Algorithmique

J. Landré - jerome.landre@univ-reims.fr

I.U.T. Troyes

Table des matières

Table des matières

Table des matières

Table des matières

Définition

Définition

Algorithme : n. m. XIIIe siècle, augorisme. Altération, sous l'influence du grec arithmos, « nombre », d'algorisme, qui, par l'espagnol, remonte à l'arabe Al-Khuwarizmi, surnom d'un mathématicien.

MATH. Méthode de calcul qui indique la démarche à suivre pour résoudre une série de problèmes équivalents en appliquant dans un ordre précis une suite finie de règles. (D'après le dictionnaire de l'académie française).

L'algorithme est avant tout une notion mathématique.

Définition

Définition

Algorithme : n. m. XIIIe siècle, augorisme. Altération, sous l'influence du grec arithmos, « nombre », d'algorisme, qui, par l'espagnol, remonte à l'arabe Al-Khuwarizmi, surnom d'un mathématicien.

MATH. Méthode de calcul qui indique la démarche à suivre pour résoudre une série de problèmes équivalents en appliquant dans un ordre précis une suite finie de règles. (D'après le dictionnaire de l'académie française).

L'algorithme est avant tout une notion mathématique.

Algorithmes célèbres

Mathématiques

- Algorithme d'Euclide d'Alexandrie (325–265 av. J.-C.) : calcul du plus grand commun diviseur (P.G.C.D.) de deux nombres.
- Algorithme de Madhava de Sangamagrama (1350–1425) : calcul du nombre π par une série mathématique : $\pi = 4.\sum_{k=0}^{\infty} \frac{(-1)^k}{2k+1}$
- Algorithme du simplexe de Georges Dantzig (1914–2005) : résolution de problèmes d'optimisation avec contraintes.

Informatique

- Algorithme de tracé de segment de Bresenham : dessiner une ligne sur un espace discret à partir de coordonnées continues (dessin d'une ligne sur un écran d'ordinateur).
- Algorithme Lempel-Ziv-Welch (LZW) : compression de données utilisé dans le format zip.
- Algorithmes de tri (tri à bulle, tri fusion, tri rapide, ...) : trier les éléments d'un tableau dans l'ordre croissant ou décroissant.

Algorithmes célèbres

Mathématiques

- Algorithme d'Euclide d'Alexandrie (325–265 av. J.-C.) : calcul du plus grand commun diviseur (P.G.C.D.) de deux nombres.
- Algorithme de Madhava de Sangamagrama (1350–1425) : calcul du nombre π par une série mathématique : $\pi = 4.\sum_{k=0}^{\infty} \frac{(-1)^k}{2k+1}$
- Algorithme du simplexe de Georges Dantzig (1914–2005) : résolution de problèmes d'optimisation avec contraintes.

Informatique

- Algorithme de tracé de segment de Bresenham : dessiner une ligne sur un espace discret à partir de coordonnées continues (dessin d'une ligne sur un écran d'ordinateur).
- Algorithme Lempel-Ziv-Welch (LZW) : compression de données utilisé dans le format zip.
- Algorithmes de tri (tri à bulle, tri fusion, tri rapide, ...) : trier les éléments d'un tableau dans l'ordre croissant ou décroissant.

Table des matières

Historique

- Vers 820 : Travaux du mathématicien perse **Al-Khuwarizmi** (né vers 780 – mort vers 850) en mathématiques
- En 1833, **Charles Babbage** (1791–1871) et **Ada Lovelace** (1815–1852) inventent la machine analytique composée d'un moulin (unité de calcul), d'un magasin (mémoire) et d'un dispositif de contrôle. Cet ensemble était utilisé avec des cartes opérations, des cartes variables et des cartes nombres. C'est le modèle exact du fonctionnement d'un ordinateur moderne.
- L'histoire retiendra que Ada Lovelace (née Augusta Ada King) fut l'inventeur de la programmation. En l'honneur du mathématicien Al-Khuwarizmi, elle appelle algorithme le processus logique permettant l'exécution d'un programme.

Historique

- En 1854, **Georges Boole** introduit le calcul binaire qui structure la logique avec deux états 1/0, oui/non, vrai/faux.
- En 1934, **Johannes Von Neumann** définit l'architecture qui porte son nom et qui est à l'origine de l'ordinateur actuel.
- En 1936, **Alan Türing** définit la notion formelle d'algorithme et de complexité.

Table des matières

Avant-propos

Avant de définir l'algorithmique (ou algorithmie), il faut définir le concept d'informatique.

informatique

n.f. Science du traitement rationnel et automatique de l'information (d'après le dictionnaire de l'académie française).

Ordinateur

Machine capable de résoudre des problèmes en appliquant une suite d'instructions préalablement définies.

But de l'informatique : modéliser et résoudre des problèmes du monde réel dans le monde numérique.

Avant-propos

Avant de définir l'algorithmique (ou algorithmie), il faut définir le concept d'informatique.

informatique

n.f. Science du traitement rationnel et automatique de l'information (d'après le dictionnaire de l'académie française).

Ordinateur

Machine capable de résoudre des problèmes en appliquant une suite d'instructions préalablement définies.

But de l'informatique : modéliser et résoudre des problèmes du monde réel dans le monde numérique.

Avant-propos

Avant de définir l'algorithmique (ou algorithmie), il faut définir le concept d'informatique.

informatique

n.f. Science du traitement rationnel et automatique de l'information (d'après le dictionnaire de l'académie française).

Ordinateur

Machine capable de résoudre des problèmes en appliquant une suite d'instructions préalablement définies.

But de l'informatique : modéliser et résoudre des problèmes du monde réel dans le monde numérique.

Avant-propos

Avant de définir l'algorithmique (ou algorithmie), il faut définir le concept d'informatique.

informatique

n.f. Science du traitement rationnel et automatique de l'information (d'après le dictionnaire de l'académie française).

Ordinateur

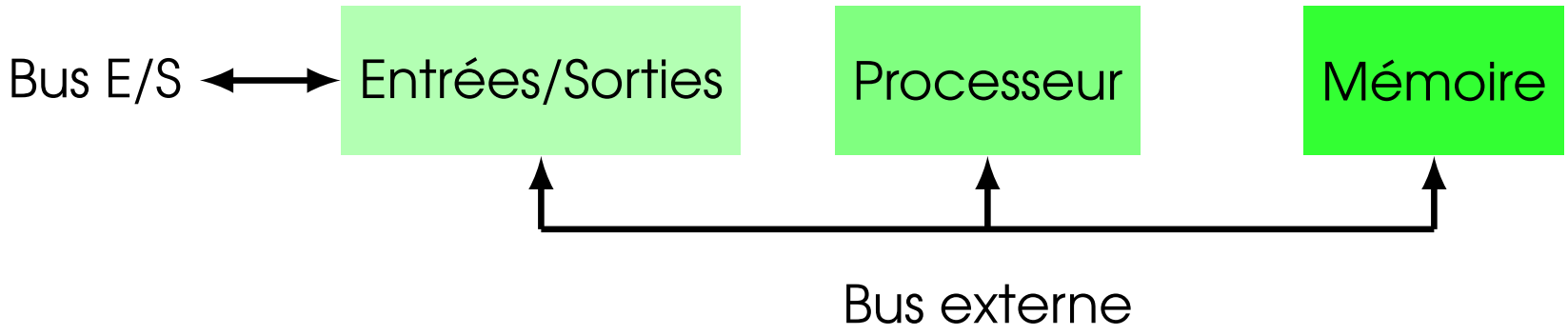
Machine capable de résoudre des problèmes en appliquant une suite d'instructions préalablement définies.

But de l'informatique : modéliser et résoudre des problèmes du monde réel dans le monde numérique.

Architecture de Von Neumann

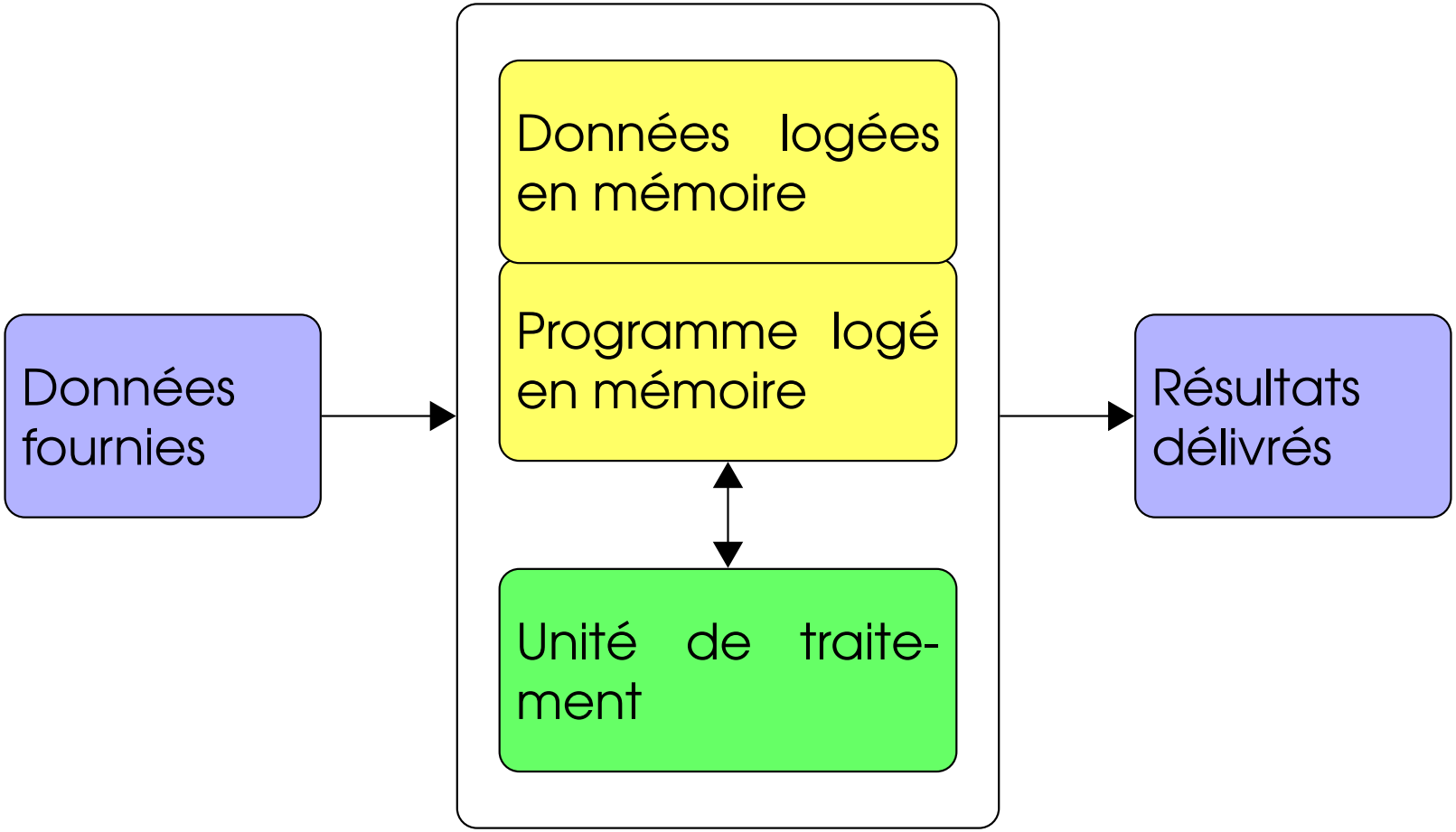
Un ordinateur est composé :

- d'un processeur,
- de mémoire,
- d'une unité d'entrées-sorties.



Modélisation

A partir du programme, des données en mémoire et des données fournies (entrées), l'unité de traitement calcule les résultats (sorties).



Ordinateur $\rightarrow$ Machine électronique $\rightarrow$ Binaire

100110110010101101010110010001010111010100101
000001010011010101110101111010110101010101001
011001010101010101010101010101010101010101101
001101010101010101001001010001010010101010101
010011010110100110000110101000101010101010101
010010100100101011010101010101001010101010010
101010101010101001010101010111010101101010101
001101010110010101010101010010101010101001010
101000010011010111010111111010101010100110010

Table des matières

Langage machine

- L'ordinateur est une machine électronique dont les circuits ne peuvent exécuter qu'un nombre réduit d'instructions simples : additionner deux nombres, voir si un nombre est égal à zéro, lire et écrire des nombres en mémoire, ...
- Langage machine : ensemble des instructions compréhensibles directement par l'ordinateur.
- Problème : L'ordinateur étant une machine électronique, son langage machine L1 est en binaire, chaque instruction représente une opération électronique au niveau des registres (mémoire interne) du micro-processeur, le code est donc très difficile à lire.
- Exemple :
  00000001 10000101 // charger registre A avec adresse 133
  00000010 00000000 // incrémenter valeur registre A
  ...

Machines virtuelles

Solution :

- On construit un langage L2 plus facile à comprendre : on associe chaque instruction de L2 avec un ensemble d'instructions de L1.
- Exemple :
  LOAD A,(133) // charger registre A avec adresse 133
  INC A // incrémenter valeur registre A
  ...

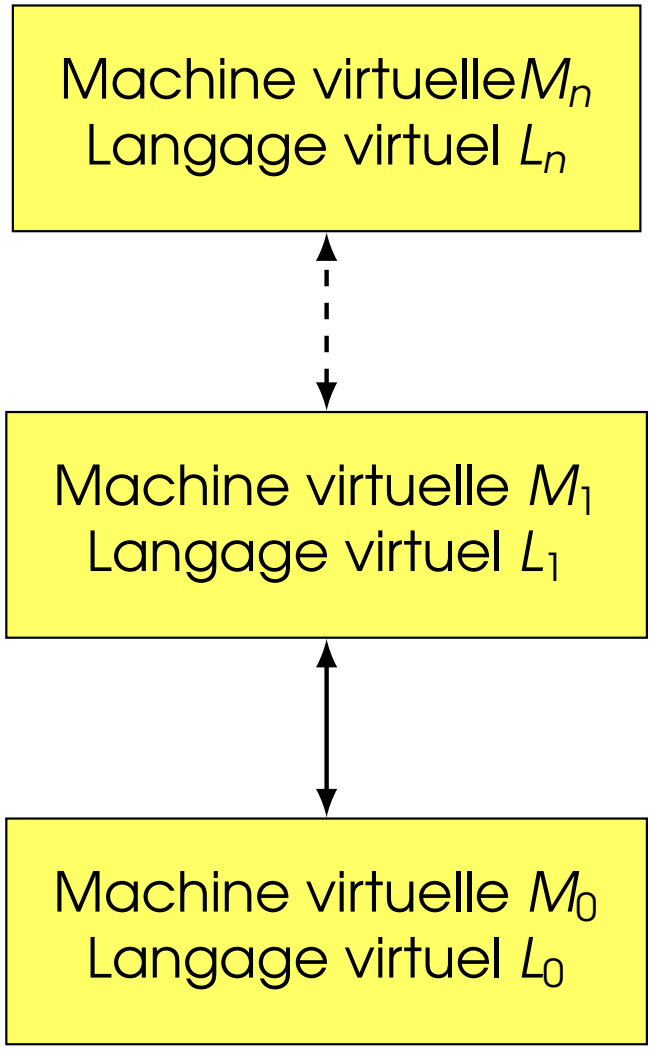
On a donc créé une machine virtuelle L2 capable de comprendre le langage L2 et de le transformer en langage L1.
Problème : Le langage L2 reste difficile à comprendre.

Solution :

- On construit un langage L3 plus facile à comprendre : on associe chaque instruction de L3 avec un ensemble d'instructions de L2.
- Exemple :
  int a ;
  a=mem(133) ; // charger registre A avec adresse 133
  a++ ; // incrémenter valeur registre A
  ...

On a donc créé une machine virtuelle L3 capable de comprendre le langage L3 et de le transformer en langage L2.
Problème : Le langage L3 reste difficile à comprendre.

Machines multi-couches



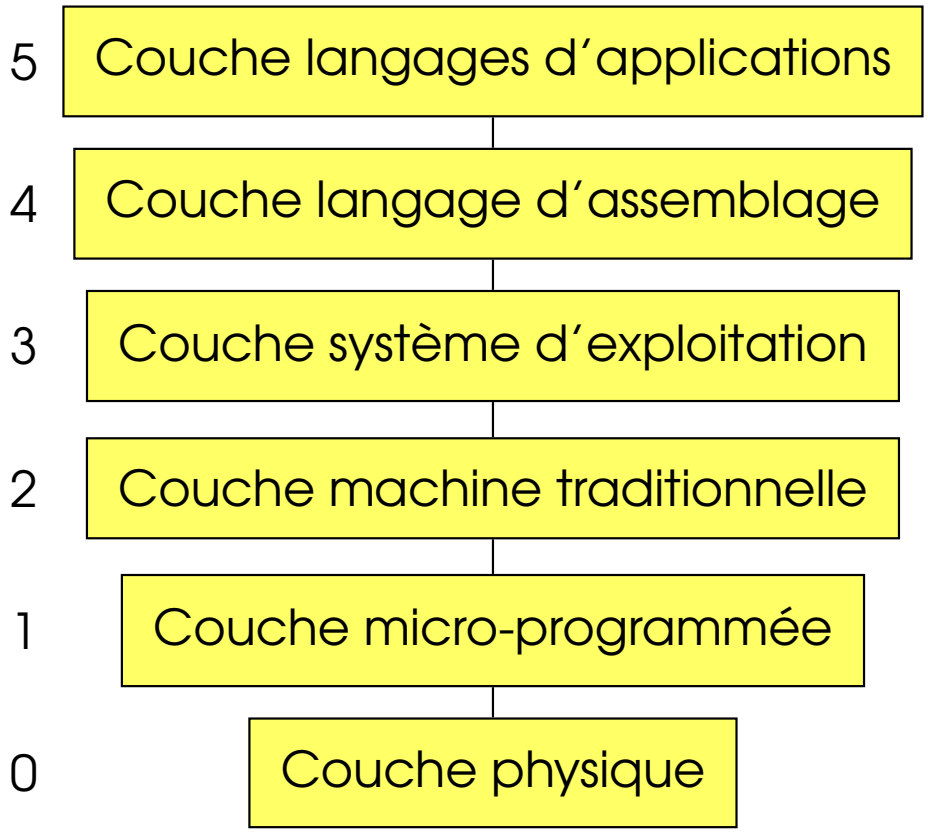
Un ordinateur est donc une machine réelle électronique comprenant un langage machine à laquelle on a ajouté des couches de machines virtuelles comprenant chacune un langage virtuel.

Compilation/interprétation

Au niveau k, il y a deux façons de voir le langage de niveau k+1 :

- Dans sa globalité : le programme virtuel écrit dans le langage L(k+1) est transformé entièrement en langage Lk : c'est la compilation, réalisée par un compilateur,
- Ligne par ligne : le programme virtuel écrit dans le langage (k+1) est lu ligne par ligne en effectuant à chaque fois les instructions correpondantes du langage Lk : c'est l'interprétation réalisée grâce à un interpréteur.

Architecture à six niveaux



- Niveau 5 : langages de programmation évolués (C, C++, BASIC, ADA, LISP, ...)
- Niveau 4 : langage assembleur
- Niveau 3 : services du système d'exploitation
- Niveau 2 : micro-programme évolué
- Niveau 1 : micro-programme physique
- Niveau 0 : portes logiques numériques

Définition

Algorithme

Séquence d'instructions de base nécessaires à la résolution d'un problème utilisant des données en entrée et calculant un résultat en sortie.

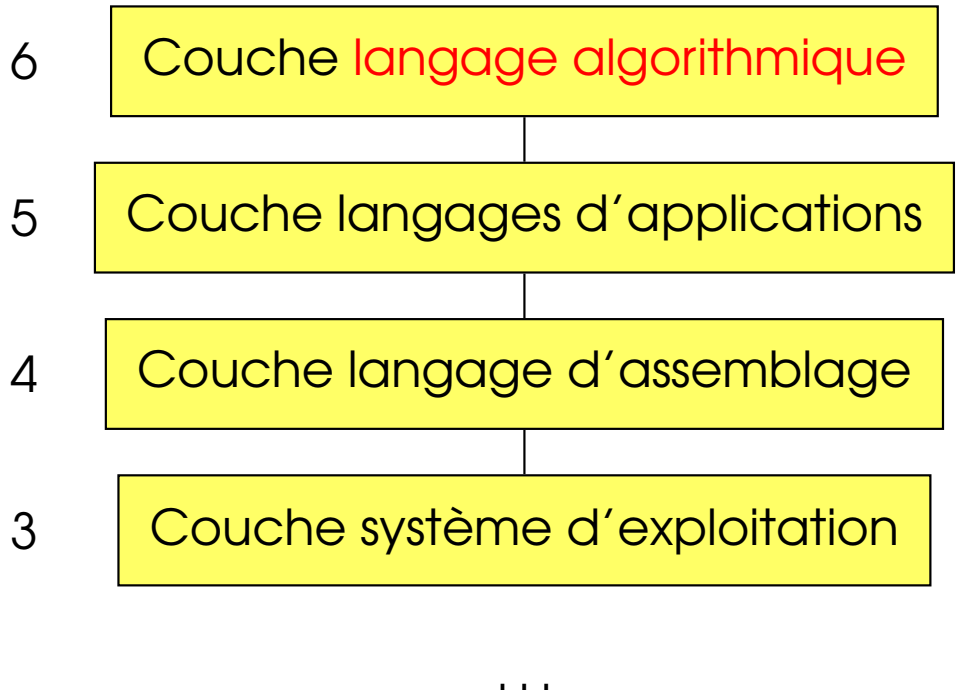
Écrire un algorithme, c'est décrire avec une séquence d'instructions très simples le moyen d'arriver au résultat qu'on souhaite à partir des données du problème.

Programme

Séquence des instructions écrites dans un langage de programmation décrivant la façon dont l'ordinateur doit effectuer un certain travail.

Langage algorithmique

Afin de décrire la séquence d'instructions nécessaires à la résolution du programme, on va définir un langage algorithmique situé au niveau d'abstraction le plus haut de notre hiérarchie de langages.



Langages de programmation

Il existe une multitude de langages de programmation, ayant chacun leurs caractéristiques, leurs avantages et leurs inconvénients.

Il existe des langages :

généralistes : on peut programmer tous les types d'applications, par exemple C/C++/C#, Java, Python, etc.

spécialisés : ils sont réservés à un type donné d'application, par exemple Prolog, HTML, javascript, LaTeX, etc.

Il existe des langages :

déclaratifs dans lesquels on doit déclarer les variables avant de les utiliser, par exemple C/C++/C#, Java, actionscript3, etc.

non-déclaratifs dans lesquels le type de variable est défini dynamiquement, comme PHP, Python, etc.

Langages de programmation

Il existe une multitude de langages de programmation, ayant chacun leurs caractéristiques, leurs avantages et leurs inconvénients.

Il existe des langages :

généralistes : on peut programmer tous les types d'applications, par exemple C/C++/C#, Java, Python, etc.

spécialisés : ils sont réservés à un type donné d'application, par exemple Prolog, HTML, javascript, LaTeX, etc.

Il existe des langages :

déclaratifs dans lesquels on doit déclarer les variables avant de les utiliser, par exemple C/C++/C#, Java, actionscript3, etc.

non-déclaratifs dans lesquels le type de variable est défini dynamiquement, comme PHP, Python, etc.

Langages de programmation

ABC Ada ADL Algol 60
Algol 68 APL AppleScript
ARB Assembly Awk
BASIC Befunge BETA
Bigwig Bistro Blue
Brainfuck **C** **C++** Caml
Cecil Cg CHILL Clarion
Clean Clipper CLU
Cobol CobolScript
Cocoa Component
Pascal **C-sharp** Curl D
DATABUS Delphi DOS
Batch Dylan E Eiffel
ElastiC Erlang Euphoria
Forth Fortran Fortress FP
Frontier GLSL Goedel
Groovy Haskell HLSL
HTML HTMLScript
HyperCard ICI Icon IDL
Intercal Io Jal **Java**
JavaScript Jovial
LabVIEW Lagoona LaTeX
Leda Limbo Lisp Logo
Lua m4 Maple
Mathematica MATLAB
Mercury Miranda Miva
ML Modula-2 Modula-3
Moto Mumps Oberon
Objective Caml
Objective-C Obliq
Occam Oz Pascal Perl
PHP Pike PL Pliant PL-SQL
POP-11 PostScript
PowerBuilder Prograph
Prolog Proteus Python R
REBOL Refal Rexx Rigal
RPG Ruby SAS Sather
Scheme Self SETL SGML
Simkin Simula Sisal
S-Lang Smalltalk Snobol
SQL Squeak TADS Tcl-Tk
Tempo TeX TOM TRAC
Transcript Turing T3X UML
VBScript Verilog VHDL
Visual Basic Visual
DialogScript Visual
FoxPro Water **XML** XOTcl
YAFL Yorick Z

Liste non-exhaustive !

Un exemple simple. . .

Pour calculer le plus grand commun diviseur (P.G.C.D.) entre deux nombres, on utilise l'algorithme d'Euclide.

Euclide (né vers -325, mort vers -265) a proposé dans "Les éléments" une méthode de calcul du plus grand commun diviseur (P.G.C.D.) de deux nombres entiers a et b.

1) On divise a par b, si le reste r de la division est nul, le P.G.C.D. est égal à b.

2) Sinon, on remplace a par b et b par r et on recommence à l'étape 1.

Exemple :

$$\begin{array}{r|l} 81 & 21 \\ \cline{2-2} 18 & 3 \end{array} \qquad \begin{array}{r|l} 21 & 18 \\ \cline{2-2} 3 & 1 \end{array} \qquad \begin{array}{r|l} 18 & 3 \\ \cline{2-2} 0 & 6 \end{array}$$

Le P.G.C.D. de 81 et 21 est donc 3 (dernier diviseur ayant conduit au reste nul).

Un exemple simple. . .

Pour calculer le plus grand commun diviseur (P.G.C.D.) entre deux nombres, on utilise l'algorithme d'Euclide.

Euclide (né vers -325, mort vers -265) a proposé dans "Les éléments" une méthode de calcul du plus grand commun diviseur (P.G.C.D.) de deux nombres entiers a et b.

1) On divise a par b, si le reste r de la division est nul, le P.G.C.D. est égal à b.

2) Sinon, on remplace a par b et b par r et on recommence à l'étape 1.

Exemple :

8 1	2 1
1 8	3

2 1	1 8
3	1

1 8	3
0	6

Le P.G.C.D. de 81 et 21 est donc 3 (dernier diviseur ayant conduit au reste nul).

Un exemple simple. . .

Pour calculer le plus grand commun diviseur (P.G.C.D.) entre deux nombres, on utilise l'algorithme d'Euclide.

Euclide (né vers -325, mort vers -265) a proposé dans "Les éléments" une méthode de calcul du plus grand commun diviseur (P.G.C.D.) de deux nombres entiers a et b.

1) On divise a par b, si le reste r de la division est nul, le P.G.C.D. est égal à b.

2) Sinon, on remplace a par b et b par r et on recommence à l'étape 1.

Exemple :

8 1	2 1
1 8	3

2 1	1 8
3	1

1 8	3
0	6

Le P.G.C.D. de 81 et 21 est donc 3 (dernier diviseur ayant conduit au reste nul).

Un exemple simple. . .

Pour calculer le plus grand commun diviseur (P.G.C.D.) entre deux nombres, on utilise l'algorithme d'Euclide.

Euclide (né vers -325, mort vers -265) a proposé dans "Les éléments" une méthode de calcul du plus grand commun diviseur (P.G.C.D.) de deux nombres entiers a et b.

1) On divise a par b, si le reste r de la division est nul, le P.G.C.D. est égal à b.

2) Sinon, on remplace a par b et b par r et on recommence à l'étape 1.

Exemple :

$$\begin{array}{r|l} 81 & 21 \\ \cline{2-2} 18 & 3 \end{array} \qquad \begin{array}{r|l} 21 & 18 \\ \cline{2-2} 3 & 1 \end{array} \qquad \begin{array}{r|l} 18 & 3 \\ \cline{2-2} 0 & 6 \end{array}$$

Le P.G.C.D. de 81 et 21 est donc 3 (dernier diviseur ayant conduit au reste nul).

Un exemple simple. . .

Pour calculer le plus grand commun diviseur (P.G.C.D.) entre deux nombres, on utilise l'algorithme d'Euclide.

Euclide (né vers -325, mort vers -265) a proposé dans "Les éléments" une méthode de calcul du plus grand commun diviseur (P.G.C.D.) de deux nombres entiers a et b.

1) On divise a par b, si le reste r de la division est nul, le P.G.C.D. est égal à b.

2) Sinon, on remplace a par b et b par r et on recommence à l'étape 1.

Exemple :

$$\begin{array}{r|l} 81 & 21 \\ \hline 18 & 3 \end{array} \qquad \begin{array}{r|l} 21 & 18 \\ \hline 3 & 1 \end{array} \qquad \begin{array}{r|l} 18 & 3 \\ \hline 0 & 6 \end{array}$$

Le P.G.C.D. de 81 et 21 est donc 3 (dernier diviseur ayant conduit au reste nul).

Un exemple simple...

euclide.c

```c
#include <stdio.h>

int main() {

int a=12345;
int b=67890;
int r=a % b;

while (r != 0) {
  a=b;
  b=r;
  r=a % b;    }

printf("le pgcd est %d\n", b);

return(0); }
```

Euclide.java

```java
class Euclide {

  public static void main(String() args)
  {

    int a=12345;
    int b=67890;
    int r=a % b;

    while (r != 0)      {
      a=b;
      b=r;
      r=a % b; }

    System.out.println("le pgcd est "+ b);
  }
}
```

Un exemple simple...

euclide.c

```
#include <stdio.h>

int main() {

int a=12345;
int b=67890;
int r=a % b;

while (r != 0) {
  a=b;
  b=r;
  r=a % b;    }

printf("le pgcd est %d\n", b);

return(0); }
```

Euclide.java

```
class Euclide {

  public static void main(String() args)
  {

    int a=12345;
    int b=67890;
    int r=a % b;

    while (r != 0)      {
      a=b;
      b=r;
      r=a % b; }

    System.out.println("le pgcd est "+ b);
  }
}
```

Un exemple simple. . .

euclide.py

```
a=12345
b=67890
r=a % b
while r<>0:
    a=b
    b=r
    r=a % b
print "le pgcd est %i" % b
```

Euclide.as

```
package {

import flash.display.Sprite;

public class Euclide extends Sprite {
public function Euclide() {
var a:int=12345;
var b:int=67890;
var r:int=a % b;

trace(a +" " +b+ " "+r);
while (r != 0) {
a=b;
b=r;
r=a % b;
}

trace("le pgcd est "+ b);
}
}
}
```

Un exemple simple. . .

euclide.py

```
a=12345
b=67890
r=a % b
while r<>0:
    a=b
    b=r
    r=a % b
print "le pgcd est %i" % b
```

Euclide.as

```
package {

import flash.display.Sprite;

public class Euclide extends Sprite {
public function Euclide() {
var a:int=12345;
var b:int=67890;
var r:int=a % b;

trace(a +" " +b+ " "+r);
while (r != 0) {
a=b;
b=r;
r=a % b;
}

trace("le pgcd est "+ b);
}
}
}
```

Un exemple simple...

euclide.php

```php
<?
        $a = 12345;
        $b = 67890;
        $r = $a % $b;

        while ($r != 0) {
                $a = $b;
                $b = $r;
                $r = $a % $b;
        }

        echo "le pgcd est ".$b;
?>
```

euclide.prolog

```
gcd(X, Y, G) :- X = Y, G = X.
gcd(X, Y, G) :-
   X < Y,
   Y1 is Y mod X,
   (Y1 = 0 -> G = X; gcd(X, Y1, G)).
gcd(X, Y, G) :- X > Y, gcd(Y, X, G).

?- gcd(12345,67890,X).
X = 15 .
```

Un exemple simple...

euclide.php

```
<?
        $a = 12345;
        $b = 67890;
        $r = $a % $b;

        while ($r != 0) {
                $a = $b;
                $b = $r;
                $r = $a % $b;
        }

        echo "le pgcd est ".$b;
?>
```

euclide.prolog

```
gcd(X, Y, G) :- X = Y, G = X.
gcd(X, Y, G) :-
   X < Y,
   Y1 is Y mod X,
   (Y1 = 0 -> G = X; gcd(X, Y1, G)).
gcd(X, Y, G) :- X > Y, gcd(Y, X, G).

?- gcd(12345,67890,X).
X = 15 .
```

Un exemple simple...

euclide.ruby

```
a=12345
b=67890

a, b = b, a % b   until b.zero?
print a
```

euclide.clisp

```
(setq a 12345)
(setq b 67890)
(do ((n a)) ((zerop b) (abs a))
    (shiftf n a b (mod n b)))
(print a)
```

Un exemple simple...

euclide.ruby

```
a=12345
b=67890

a, b = b, a % b   until b.zero?
print a
```

euclide.clisp

```
(setq a 12345)
(setq b 67890)
(do ((n a)) ((zerop b) (abs a))
    (shiftf n a b (mod n b)))
(print a)
```

Un exemple simple. . .

Sur cet exemple, la façon d'écrire le programme change. Par contre, le principe de calcul reste le même, c'est l'algorithme d'Euclide.
On va donc définir un langage algorithmique qui va permettre d'écrire l'algorithme, indépendamment du langage d'implémentation choisi.

Données : **var** a, b, r : **entiers**
Résultats : Le P.G.C.D. de a et b

début
  $a \leftarrow 12345$ $b \leftarrow 67890$ $r \leftarrow a$ modulo b
  tantque $r \neq 0$ **faire**
    $a \leftarrow b$
    $b \leftarrow r$
    $r \leftarrow a$ modulo b
  fintantque
  écrire "Le pgcd est : " + b
fin

Algorithme 1 : Algorithme d'Euclide.

Commentaires

Les commentaires donnent des informations sur votre programme. L'écriture des commentaires varie selon les langages de programmation.

En algorithmique, on utilise la notation // pour marquer un commentaire qui va jusqu'à la fin de la ligne marquée.

Données : **var** *a*, *b*, *r* : **entiers**
Résultats : Le P.G.C.D. de *a* et *b*

début
  a ← 12345 *//on initialise a*
  b ← 67890 *//on initialise b*
  r ← *a* modulo *b* *//on calcule a mod b*
  tantque $r \neq 0$ **faire**
    a ← *b* *//a prend la valeur b*
    b ← *r* *//b prend la valeur r*
    r ← *a* modulo *b* *//on recalcule a mod b*
  fintantque
  écrire "Le pgcd est : " + b *//on affiche le pgcd*
fin

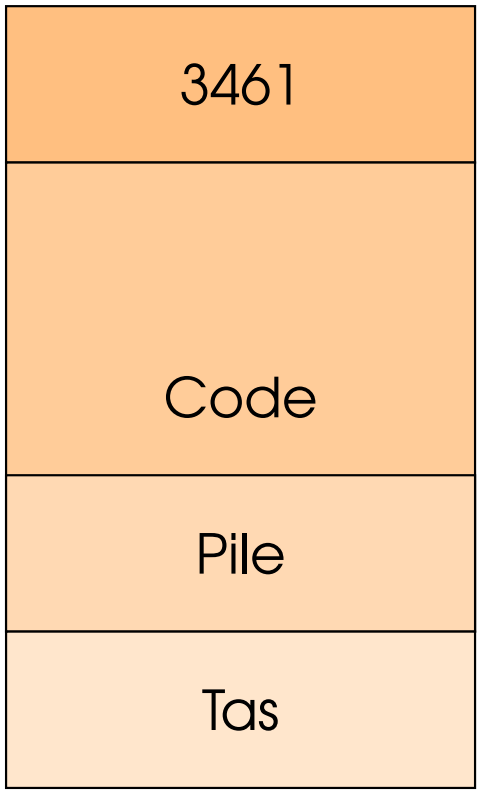
Algorithme 2 : Algorithme d'Euclide commenté.

Table des matières

Petit détour par le système d'exploitation

Le système d'exploitation gère le programme en exécution, il alloue la mémoire nécessaire et gère les ressources matérielles de l'ordinateur.

Un processus est un programme en exécution. Il est composé du code du programme, d'une pile (*stack*) et d'un tas (*heap*).



- Le processus porte le numéro 3461,
- le code est du langage machine,
- la pile contient les données d'exécution,
- le tas contient les variables du programme.

Séquencement

- Les intructions d'un algorithme sont toujours exécutées séquentiellement, une par une, à la suite les unes des autres depuis la première jusqu'à la dernière.

L'ordre des opérations est très important !

Compilation ou interprétation ?

Compilation

La liste des instructions est transformée en langage machine avant l'exécution.

Interprétation

Les instructions sont décodées et exécutées au fil de l'eau.

Table des matières

Types de données

Avant d'utiliser des données, il est obligatoire de définir leur type.
Le type de donnée donne des indications sur les valeurs qu'elles peuvent prendre.

Il existe cinq types de base :

Types de base

- Les entiers,
- les réels,
- les booléens,
- les chaînes de caractères,
- les dates.

Il existe deux types composites :

Types composites

- Les tableaux,
- les structures.

Types de base

entiers : **var** a : **entier** $\Rightarrow a \in \mathbb{Z}$
a $\leftarrow$ -7

réels : **var** b : **réel** $\Rightarrow b \in \mathbb{R}$
b $\leftarrow$ 3.14

booléens : **var** c : **booléen** $\Rightarrow c \in \{Vrai, Faux\}$
c $\leftarrow$ Vrai

chaîne de caractères : **var** d : **chaine** $\Rightarrow d \in \{[0-9a-zA-Z]*\}$
d $\leftarrow$ "Bonjour"

Types de base

entiers : **var** a : **entier** $\Rightarrow a \in \mathbb{Z}$
a $\leftarrow$ -7

réels : **var** b : **réel** $\Rightarrow b \in \mathbb{R}$
b $\leftarrow$ 3.14

booléens : **var** c : **booléen** $\Rightarrow c \in \{Vrai, Faux\}$
c $\leftarrow$ Vrai

chaîne de caractères : **var** d : **chaine** $\Rightarrow d \in \{[0-9a-zA-Z]*\}$
d $\leftarrow$ "Bonjour"

Types de base

entiers : **var** a : **entier** $\Rightarrow a \in \mathbb{Z}$
a $\leftarrow$ -7

réels : **var** b : **réel** $\Rightarrow b \in \mathbb{R}$
b $\leftarrow$ 3.14

booléens : **var** c : **booléen** $\Rightarrow c \in \{Vrai, Faux\}$
c $\leftarrow$ Vrai

chaîne de caractères : **var** d : **chaine** $\Rightarrow d \in \{[0-9a-zA-Z]*\}$
d $\leftarrow$ "Bonjour"

Types de base

entiers : **var** a : **entier** $\Rightarrow a \in \mathbb{Z}$
a $\leftarrow$ -7

réels : **var** b : **réel** $\Rightarrow b \in \mathbb{R}$
b $\leftarrow$ 3.14

booléens : **var** c : **booléen** $\Rightarrow c \in \{Vrai, Faux\}$
c $\leftarrow$ Vrai

chaîne de caractères : **var** d : **chaine** $\Rightarrow d \in \{[0-9a-zA-Z]*\}$
d $\leftarrow$ "Bonjour"

Table des matières

Variables

Variable

Une variable est un emplacement mémoire réservé qui permet de stocker une valeur d'un certain type.

- Le type de la variable détermine les valeurs qu'elle peut contenir.
- Lorsqu'un programme définit une variable, le système d'exploitation lui alloue (réserve) un emplacement mémoire dont la taille dépend du type de la variable.
- Le codage des informations n'est pas abordé dans ce cours, voir la cours de repésentation de l'information (SCI120)...

Constantes

Constante

Une constante est une valeur qui ne change pas au cours de l'exécution d'un programme.

```
const pi : réel
pi ← 3.14159
```

Déclaration

Il faut déclarer les variables utilisées dans un algorithme au début de celui-ci (dans un langage déclaratif).

Notation

var a : **type**
var b : **type**
var c : **type**

```
var x : entier
var y : réel
var maChaine : chaîne
```

Règle générale : dans un langage déclaratif, on doit toujours déclarer une variable avant de l'utiliser !

Affectation

L'opérateur d'affectation permet de remplir une variable avec une valeur qui correspond à son type (dans un langage déclaratif).

Notation

L'affectation se note : variable $\leftarrow$ valeur

Règle générale : il est conseillé de toujours initialiser une variable avec une valeur !

```
x ← 4
y ← -34.57
chaine ← "test de chaine"
```

Bloc d'instructions

Dans l'élaboration d'un algorithme, il est nécessaire de regrouper les instructions en "blocs".

bloc d'instructions

Un bloc d'instruction est noté :

début	{
...	...
fin	}

Un bloc d'instruction représente un ensemble d'instructions liées entre elles.

Table des matières

Exécution d'un programme

Séquence d'instructions

Les lignes d'un programme sont exécutées séquentiellement une par une dans l'ordre défini par le programme (du haut vers le bas).

```
x ← 4
y ← 2
x ← 7
afficher x
```

Quel est le résultat de ce programme ?

Structures de contrôle

- Les structures de contrôle permettent d'influencer le déroulement du programme.

Il en existe trois :

- Les tests,
- les boucles,
- les appels de sous-programmes (fonctions).

Test conditionnel

Condition

Proposition mathématique qui est vraie ou fausse (booléen).

si *condition* **alors**
| bloc1
sinon
| bloc2
finsi

Si la condition est vérifiée alors c'est le bloc1 qui est exécuté, sinon c'est le bloc2.

Combinaison de conditions

Il est possible de combiner plusieurs conditions entre elles avec les opérateurs logiques et, ou et non.

si *age > 10 ou taille>1,40* **alors**
  écrire "Vous pouvez accéder à cette attraction."
sinon
  écrire "Désolé, vous ne pouvez pas accéder à l'attraction."
finsi

Enchaînement de tests conditionnels

Il est possible d'enchaîner plusieurs tests en fonction de la complexité de la condition à exprimer.

```
si cond1 alors
  | bloc1
sinon
  | si cond2 alors
  |   | bloc2
  | sinon
  |   | si cond3 alors
  |   |   | bloc3
  |   | sinon
  |   |   | bloc4
  |   | finsi
  | finsi
finsi
```

Choix conditionnel

- Le choix conditionnel permet de tester une variable et une seule et d'adapter le code à exécuter selon la valeur de cette variable.

selonque *variable* **vaut**
| **cas** *valeur1*
| | bloc1
| **fincas**
| **cas** *valeur2*
| | bloc2
| **fincas**
| **cas** *valeur3*
| | bloc3
| **fincas**
| **défaut**
| | bloc4
| **fincas**
fincas

Boucles

Une boucle permet d'effectuer plusieurs fois le même bloc d'instruction.
Chaque exécution du bloc s'appelle une itération.

Boucle pour

Dans la boucle pour, on utilise un compteur pour effectuer les itérations.

Données : var k, i : **entiers**
Résultats : Affichage des nombres doublés.

début
  $k \leftarrow 10$
  pour $i \leftarrow 1$ **à** k **faire**
    écrire $2 \times i$
  finpour
fin

Algorithme 3 : Doubler.

Quel est le résultat de cet algorithme ?

Boucle tantque

Dans la boucle tantque, on utilise une condition pour effectuer les itérations.

Données : var *numero*, *k* : **entiers**
Résultats : Affichage d'un compteur.

début
  numero $\leftarrow 0$
  k $\leftarrow 10$
  tantque *numero* $<$ *k* **faire**
    écrire *numero*
    numero $\leftarrow$ *numero* $+ 1$
  fintantque
fin

Algorithme 4 : Compteur.

Quel est le résultat de cet algorithme ?