

2020 / 2021

Présentation

INFO 0101

Quentin JULLIARD
URCA

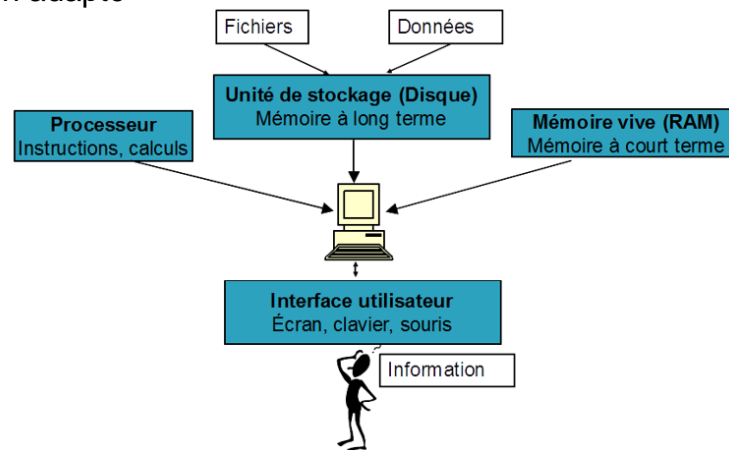
Table des matières

Préliminaire :	2
Qu'est-ce qu'un ordinateur ?	2
La structure d'un ordinateur	2
Objectif général du cours	3
Introduction à l'algorithmique	4
Qu'est-ce que l'algorithme ?	4
Les informations élémentaires manipulées par un algorithme	4
Les types de données élémentaires	5
Pourquoi les types de données ?	5
Les principaux types de données.....	5
Les instructions élémentaires	6
Affectation	6
Lecture	6
Écriture :.....	7
Génération d'un nombre aléatoire.....	7
Présentation d'un algorithme	7
Opérateurs et expression de base.....	8
Opérateur et expression.....	8
Opérateurs arithmétiques.....	8
Opérateurs relationnels.....	9
Opérateurs logiques.....	10
Instructions structurées.....	11
Structure de sélection (conditionnelles)	11
Si... Alors... Sinon.....	11
Cas... Parmi.....	12
Avant d'aller plus loin : Trace d'exécution d'un algorithme.....	12
Structures itératives (Boucles)	13
Boucle Tantque	13
Boucle POUR (1ere explication):	14
Boucle POUR (2eme explication):	15
Boucle POUR (3eme explication):	15
Boucle Faire ... Tantque	15
Synthèse :	17

Préliminaire :

Qu'est-ce qu'un ordinateur ?

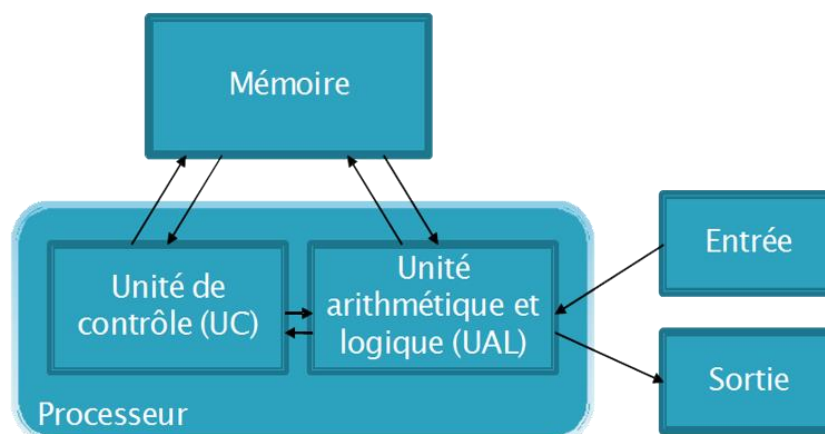
Un ordinateur est une Machine capable d'exécuter automatiquement et fidèlement une série **d'opérations simples** qu'on lui a indiquée et/ou manipuler rapidement et sans erreur un grand nombre d'informations
Pour résoudre un problème à l'aide d'un ordinateur, il faut Analyser ce problème, Déterminer une méthode de résolution, suite des opérations à effectuer (algorithme) pour obtenir la solution, Traduire l'algorithme dans un langage de programmation adapté



La structure d'un ordinateur

Le processeur, Charge une instruction à exécuter, Décode l'instruction, Localise dans la mémoire les données utilisées par l'instruction, Charge les données si nécessaire, Exécute l'instruction, Sauvegarde les résultats à leurs destinations respectives, Passe à l'instruction suivante

Basé sur le modèle de Von Neumann



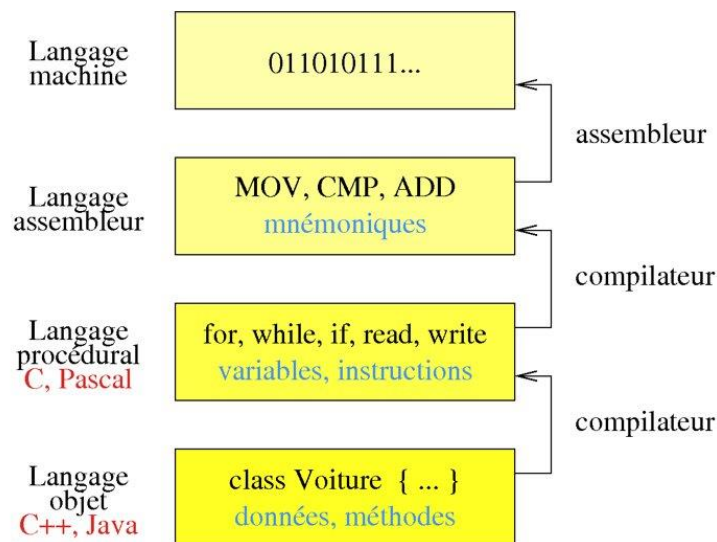
La mémoire Stocke les données et les instructions et Formée d'un certain nombre de cellules, ou cases, contenant chacune une information. Le processeur peut y accéder à n'importe quel moment, en lecture : consultation, en écriture : modification. Les entrées/sorties échange avec l'environnement externe (utilisateur), avec Périphériques de communication clavier, souris, imprimante.

Objectif général du cours

Le but de l'Algorithmique est de décomposer un problème en une suite d'instructions simples pouvant être exécutées par un ordinateur (théorique). L'ordinateur n'est pas intelligent, il faut lui dire quoi faire dans un langage qu'il comprend !

L'Algorithmique formalisme indépendant de tout langage de programmation.

La programmation a pour but d'écrire un algorithme dans un langage de programmation spécifique afin de pouvoir l'exécuter sur un ordinateur réel.



Introduction à l'algorithmique

Qu'est-ce que l'algorithme ?

Le problème principal du concepteur ou du programmeur est qu'à partir des informations fournies, décrire la suite des actions élémentaires permettant de résoudre un problème et de produire les résultats attendus.

Algorithmique est l'étude et production de règles et techniques impliquées dans la conception d'algorithmes.

Algorithme (Encyclopaedia Universalis) est une suite finie de règles à appliquer dans un ordre déterminé à un nombre fini de données pour arriver, en un nombre fini d'étapes, à un certain résultat.

Le rôle de l'algorithme est fondamental en informatique sans algorithme, pas de programme. Un programme n'est que sa traduction dans un langage compréhensible par l'ordinateur.

Un algorithme est indépendant à la fois du langage de programmation dans lequel il est traduit et de l'ordinateur sur lequel le programme doit être exécuté.

Les informations élémentaires manipulées par un algorithme

Une information élémentaire est désignée par un nom (invariable) -> pour l'utiliser, un type (invariable) -> pour l'interpréter et d'une valeur qui est modifiable (variable) ou non modifiable (constante).

Section de déclarations permet de définir les informations élémentaires nécessaires au début de l'algorithme.

Exemple 1: algorithme Info0101

Les algorithmes doivent pouvoir manipuler des données de différentes natures. Pour les représenter adéquatement, il faut connaître les **types de données** élémentaires.

Les types de données élémentaires

Pourquoi les types de données ?

Un ordinateur fonctionne en langage binaire, cela signifie qu'il ne reconnaît que les suites de bits (0 et 1).

La même suite de bits peut représenter une instruction ou une donnée (nombre, caractère, etc...).

Exemple : 0110000. Il peut-être représenter comme un entier donc 97 ou comme un caractère (ascii) : « a » .

Le typage d'une variable permet d'interpréter sa valeur, de lui donner du sens.

Les principaux types de données

Le premier type de données est l'entier, il Permet de représenter des valeurs numériques entières positives ou négatives, 0, 12, 150, -43, ...

Déclarations variables => a : entier

Le second type de données est le flottant, il permet de représenter (de manière approchée) une partie des nombres réels, notation décimale (1,346 6567,4552 - 4,0), notation exponentielle (1346E-3 6,5674552e3 -4e0).

Déclarations variables => a : réel

Le troisième type de données est le caractère, il permet de représenter un seul caractère et il est noté entre apostrophes 'a', 'i', ' ', '+', 'Z', ...

Déclarations variables => a : caractère

Le quatrième type de données est le booléen sert à représenter une valeur logique du type vrai ou faux. De plus, il ne peut prendre que 2 valeurs : Vrai (ou 1) ou Faux (ou 0)

Déclarations variables => a : booléen

Afin de manipuler les données dans un algorithme, il faut utiliser des **instructions**.

Les instructions élémentaires

Affectation

L'affectation a pour but d'attribuer une valeur donnée à une variable. Il a pour notation **variable ← valeur**. La valeur et la variable doivent être du même type. Après affectation, l'ancienne valeur de la variable est perdue. La valeur peut-être le résultat d'une expression.

Exemple d'algorithme :

Déclarations

Variables

a : entier

b : caractère

Début

a ← 2

b ← 'a'

Fin

Attention aux variables non initialisées !

a ← 2

b ← 2 + a

a ← a + 1

a ← 2 * b + e

Valeur de e??

Exemple 2: Echange des valeurs de 2 variable

Lecture

L'attribution d'une valeur à une variable par l'intermédiaire d'un périphérique d'entrée (souris, clavier, etc...).

Il se note :

variable1 ← lire ()

variable2 ← lire ("Entrez une valeur")

Proche de l'affectation, mais la valeur est à l'initiative de l'utilisateur.

Exemple :

a ← 2

a ← lire ()

Écriture :

Imprimer (afficher) une valeur ou le contenu d'une variable sur un périphérique de sortie (écran, imprimante, etc...).

Elle a pour notation :

écrire (variable)

écrire (variable1 + variable2)

écrireLn (variable1 + " et " + variable2)

Il ne modifie pas la valeur de la variable !

Exemple :

a ← 2

écrire ("Entrez une valeur")

b ← lire()

c ← a + b

écrire (c)

Génération d'un nombre aléatoire

Il génère un nombre réel au (pseudo) hasard dans l'intervalle [0,1[

Il a pour notation variable ← aleatoire()

La variable prendra une valeur aléatoire réelle entre 0 et 0,999999...

Exemple :

Déclarations

Variables

val : réel

Début

val ← aleatoire()

/* val prend une valeur aléatoire réelle dans l'intervalle [0, 1 [*/

val ← aleatoire() * 10

/* val prend une valeur aléatoire réelle dans l'intervalle [0, 10 [*/

val ← aleatoire() * 10 - 5

/* val prend une valeur aléatoire réelle dans l'intervalle [-5, 5 [*/

Fin

Présentation d'un algorithme

Il est explicite les actions principales par des commentaires entre /* et */

Il faut mettre en évidence les mots clés en évidence et utilise une indentation qui facilite la lecture.

Algorithme Échange

Déclarations

Variables

a, b, temp : entier

Début

a ← lire() /*initialisation de a*/

b ← lire() /*initialisation de b*/

temp ← a /*sauvegarde de la valeur de a*/

a ← b /*recopie de la valeur de b dans a*/

b ← temp /*recopie de l'ancienne valeur de a dans b*/

écrire (a + " " + b) /*affichage des valeurs de a et b*/

Fin

Opérateurs et expression de base

Opérateur et expression

Les opérateurs (vus plus loin)

+, -, *, ...

Sont utilisés dans des expressions

x + b

Qui sont elles-mêmes utilisées dans des instructions

y ← a * x + b

Qui, combinées de différentes façons, constitueront un algorithme/programme

Opérateurs arithmétiques

Les opérateurs binaire (2 opérandes) :

Addition	+
Soustraction	-
Multiplication	*
Division	/
Modulo	%

Les opérateur unaires (1 seul opérandes) :

Opposé	-
Identité	+

Le fonctionnement des opérateurs binaires, à priori définis pour 2 opérandes de même type, fournissent un résultat de ce type. Les conversions implicites permettent éventuellement de spécifier des opérandes de types différents.

La priorité relative des opérateurs :

1. Opérateurs unaires + et -
2. Opérateurs binaires *, / et %
3. Opérateurs binaires + et -

La priorité identique calcule de la gauche vers la droite.

Exemple :

Déclarations

Variables

a, b, c, d, e : entier

Début

```
a ← 1 /*initialisation de la variable a*/  
b ← 2 /*initialisation de la variable b*/  
c ← a + b /*c vaut 3 : 1 + 2*/  
d ← a + b * c /*d vaut 7 : 1 + (2 x 3)*/  
e ← -b /*e vaut -2 : opposé de 2*/
```

Fin

Opérateurs relationnels

A priori définis pour des opérandes de même type, mais aussi soumis aux conversions implicites.

Retourne une valeur booléenne (vrai/faux)

Opérateur	Signification
<	Inférieur à
<=	Inférieur ou égal à
>	Supérieur à
>=	Supérieur ou égal à
==	Égal à
!=	Différent de

Exemple :

Déclarations

Variables

a, b : entier
c, d : booléen

Début

a ← 1 /*initialisation de la variable a*/
b ← 2 /*initialisation de la variable b*/
c ← a == b /*c vaut Faux : 1 n'est pas égal à 2*/
d ← a < b /*d vaut Vrai : 1 est plus petit que 2*/

Fin

Opérateurs logiques

Retourne un booléen (Vrai ou Faux)

Permet de combiner plusieurs expressions

ET

Vrai si et seulement si les 2 opérandes sont vraies

OU

Vrai si au moins un des deux opérandes est vrai

NON

Vrai si l'opérande est faux

Faux si l'opérande est vrai

Exemple :

Déclarations

Variables

a, b : entier
c, d : booléen

Début

a ← 1 /*initialisation de la variable a*/
b ← 2 /*initialisation de la variable b*/
c ← (a == b) ET (b > 0) /*c vaut Faux : Faux ET Vrai*/
d ← c OU (a > 0) /*d vaut Vrai : Faux OU Vrai*/

Fin

Utiliser des parenthèses pour spécifier l'ordre des opérations et/ou clarifier l'algorithme

Instructions structurées

Structure de sélection (conditionnelles)

Ils expriment la notion de choix

Le branchement de l'évaluation d'une condition déterminera quelles instructions seront effectuées

La condition s'exprime dans l'expression booléenne

Si... Alors... Sinon...

Si la condition est vraie, on exécute une suite d'instructions

Si la condition est fausse, on exécute une autre suite d'instructions

Notation :

Si condition **Alors**

action1

action2

...

actionn

Sinon

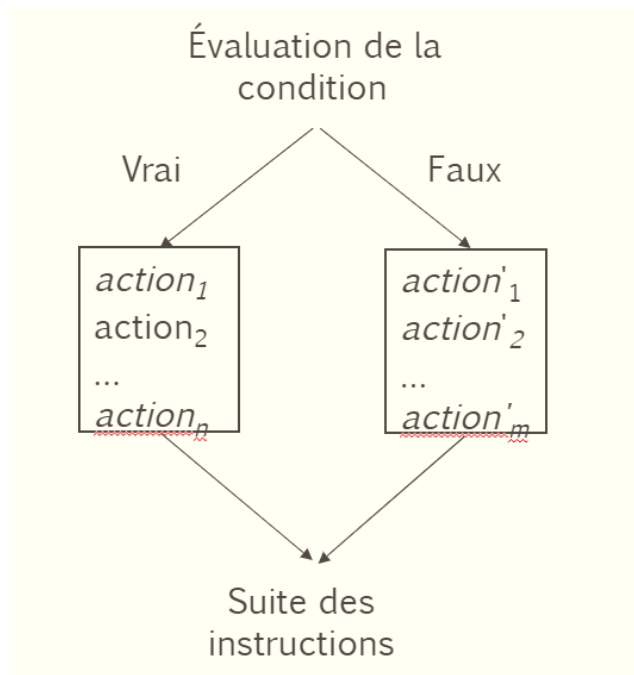
action'1

action'2

...

action'm

FinSi



La partie Sinon est facultative

S'il n'y a pas de Sinon, FinSi est placé après le premier groupe d'actions

Exemple 3 : Calcul du salaire d'un fonctionnaire

Exemple 4 : Maximum de 2 valeurs

Cas... Parmi

Quand un ensemble de conditions teste les différentes valeurs possibles d'une même expression ou variable, on peut utiliser la structure *Cas Parmi*

Cas Expression Parmi

valeur1 : action1

valeur2 : action2

valeurn : actionn

défaut : actionn+1 /*optionnel*/

FinCas

Exemple 5 : Prix d'un billet de train après réduction

Avant d'aller plus loin : Trace d'exécution d'un algorithme

Permet de suivre pas-à-pas le déroulement de l'algorithme. Il est meilleure compréhension du fonctionnement. Il permet aussi éventuellement de trouver les erreurs (bugs).

Il faut expliciter le numéro des instructions et l'évaluation de contrôle, si applicable (conditions, boucles, appels de fonction/procédure), la valeur des variables (état initial et changements) et l'affichage.

<i>Inst.</i>	<i>Contrôle</i>	<i>Variables locales</i>			
		<i>a</i>	<i>b</i>	<i>c</i>	<i>resultat</i>
	Avant	?	?	?	?
1	×	-4			
2	×		3		
3	×			-1	
4	×	4			
5	×			2	
6	$(a > c) ? \text{Vrai}$				
6a.1	×		7		
7	×				17

Exemple 6 :

Algorithme SalaireFonctionnaire

Déclarations

Variables

nbHeures : entier

tauxHoraire, salaireBrut, prime, primeSE : réel

```

Début
{1}   écrire("Entrez le taux horaire : ")
{2}   tauxHoraire ← lire ()
{3}   écrire("Entrez le nombre d'heures de travail : ")
{4}   nbHeures ← lire ()
{5}   salaireBrut ← tauxHoraire * nbHeures
{6}   Si nbHeures > 100 Alors
{6a.1}   écrire("Entrez la prime : ")
{6a.2}   prime ← lire ()
{6a.3}   salaireBrut ← salaireBrut + prime
        Sinon
{6b.1}   écrire("Entrez la prime de sous-emploi : ")
{6b.2}   primeSE ← lire ()
{6b.3}   salaireBrut ← salaireBrut + primeSE
        FinSi
{7}   écrire("Salaire brut : " + salaireBrut)
      Fin

```

Structures itératives (Boucles)

Une structure itérative permet de répéter une action ou une séquence d'actions plusieurs fois.

Il existe 3 types de boucles TantQue, Pour et Faire ... Tantque.

Boucle Tantque

Permet l'exécution d'une suite d'actions tant qu'une condition de continuité est vérifiée

On l'utilise lorsqu'on ne peut pas prédire le nombre d'itérations de la boucle

Il a pour notation :

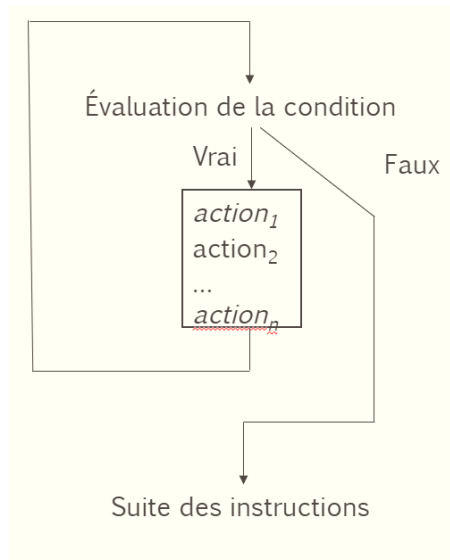
```

TantQue condition Faire
  action1
  action2
  ...
  actionn
FinTantQue

```

Condition de continuité recalculée à chaque tour de boucle

Dès que la condition est fausse, la boucle se termine



Exemple :

Exemple 7

Exemple 8

Boucle POUR (1ere explication):

Avant chaque passage éventuel dans la boucle, on teste la valeur de la variable de contrôle : on n'effectue les actions de la boucle que si elle n'a pas dépassé valeur2

Après chaque passage dans la boucle, la variable de contrôle est automatiquement incrémentée de 1

Après incrémentation, le programme retourne tester si variable_de_contrôle dépasse valeur2

La variable de contrôle est généralement entière

valeur1 et valeur2 doivent être de même type que la variable de contrôle

Pour variable_de_contrôle allant de valeur₁ à valeur₂ Faire

action₁

action₂

...

action_n

FinPour

Boucle POUR (2eme explication):

Si $\text{valeur}_1 > \text{valeur}_2$, la variable de contrôle a dès le départ une valeur supérieure à valeur_2 , donc la suite d'instructions n'est pas exécutée (même pas une fois)

À la fin d'une boucle pour, la valeur de la variable de contrôle est indéterminée. Il ne faut pas l'utiliser sans réinitialisation

Il ne faut pas modifier la valeur de la variable de contrôle à l'intérieur de la boucle

On utilise une boucle pour lorsqu'on sait exactement combien d'itérations on doit réaliser

Pour variable_de_contrôle allant de valeur₁ à valeur₂ Faire

action₁

action₂

...

action_n

FinPour

Boucle POUR (3eme explication):

Il est parfois utile de faire varier la variable de contrôle par valeurs décroissantes ou par incréments différents de l'unité. Dans ce cas, il faut utiliser la forme la plus générale de la boucle pour

Pour variable_de_contrôle allant de valeur₁ à valeur₂ par incr Faire

action₁

action₂

...

action_n

FinPour

Exemple 9 : Boucle Pour

Boucle Faire ... Tantque

Utilisé lorsque le nombre de passages est inconnu (comme pour la boucle TantQue)

Mais le corps de la boucle est toujours exécuté au moins une fois

Notation :

Faire

action1
action2
...
actionn

TantQue condition /* La condition doit être fausse */
/* pour sortir de la boucle */

L'expression logique est évaluée après l'exécution du corps de la boucle

- 1) Le corps de la boucle est exécuté
- 2) L'expression logique est évaluée
- 3) Si sa valeur est vraie
 - a. Le corps de la boucle est exécuté à nouveau
 - b. L'expression logique est réévaluée
- 4) Si sa valeur est fausse, on exécute l'instruction qui suit Tantque

Exemple 10 : Boucle Faire...Tant que

Synthèse :

Avec les types de données, opérateurs et instructions élémentaires/structurées vus dans ce cours, on peut déjà écrire des algorithmes variés

Il faut maintenant apprendre à analyser un problème, déterminer une solution adéquate à ce problème en écrivant un algorithme approprié

Un algorithme doit être efficace en exécutant le moins d'instructions possible, en utilisant le moins de mémoire possible mais il doit aussi être lisible!

Lorsqu'on dispose d'un algorithme adéquat et efficace, on peut envisager le programmer sur machine comme par exemple en Java.