

# ***ALGORITHME : INTRODUCTION***

## Sommaire

Présentation générale du développement d'un logiciel.....	1
L'algorithme .....	1
Le codage .....	1
Définition de l'algorithme .....	2
Double problématique de l'algorithmique .....	2
Différences entre « Algorithme » et « programme » .....	2
Représentation et Structures algorithmiques .....	3
Structure linéaire.....	3
Structures alternatives .....	4
Structure SI...ALORS...SINON.....	4
Structures répétitives (ou itératives).....	5
Structure FAIRE...JUSQU'À .....	5
Structure TANT QUE...FAIRE .....	5
Structure POUR...FAIRE.....	6
Choix d'un langage de programmation .....	6

## Présentation générale du développement d'un logiciel

---

Vous voilà face à votre client et à son problème.

La première étape consiste à vous mettre d'accord avec le client sur le travail à fournir. Il est notoire qu'une fois le problème bien compris, un pas décisif vers la solution est fait dans la mesure où il existe une solution informatique au problème posé.

Cette phase est l'analyse fonctionnelle.

L'étape suivante consiste à concevoir l'application. Cela veut dire modéliser l'application, la décomposer de manière descendante, proposer une (des) solution(s) et les moyens à mettre en œuvre.

Cette phase est la conception préliminaire.

La phase suivante est celle qui nous intéresse ici : la conception détaillée.

Il s'agit de présenter de manière détaillée les éléments de la solution choisie.

A ce stade du développement, le futur logiciel est architecturé en plusieurs unités de traitement (les composants) réalisant les fonctions à implémenter. Le traitement de chaque composant sera modélisé par un algorithme.

Pour cela il existe plusieurs formalismes:

- **Graphiques** : Organigrammes
- **Textuels** : Pseudo-code (Langage de description d'algorithme).

## L'algorithme

---

L'algorithme s'élabore en se centrant sur la nature du travail. C'est-à-dire sans souci des spécificités dues à la machine ou au langage de programmation.

*L'algorithme représente les opérations réalisées.*

Une fois l'algorithme terminé, il reste la phase de réalisation (codage) au cours de laquelle les algorithmes sont transcrits dans le langage de programmation retenu (C#, Java, PHP, etc..).

La traduction de cet algorithme en un programme se fait de manière quasiment automatique, sauf pour les points faisant appel aux spécificités de la machine ou du langage de programmation utilisé.

Ainsi l'algorithme a permis de traiter séparément les problèmes dus à la conception du produit de ceux dus à son implémentation. C'est pourquoi l'algorithmique est une étape indispensable à la réalisation d'applications informatiques.

## Le codage

---

En observant les mêmes règles dans la programmation que dans l'algorithme, le programmeur aboutit à une application conviviale et maintenable (c'est à dire compréhensible et modifiable par une tierce personne) à condition que les documents des phases antérieures soient complets.

Puis chaque composant est vérifié à l'aide des tests unitaires à partir d'un jeu d'essai (données sur lesquelles sont effectués les tests et qui permettent de couvrir tous les cas traités).

Après le constat du bon fonctionnement de chaque composant, il est procédé aux tests d'intégration ou tests d'ensemble.

## Définition de l'algorithme

---

**Algorithme** : Suite finie d'opérations élémentaires constituant un schéma de calcul ou de résolution d'un problème.

**Algorigramme** : Traduction graphique de l'algorithme. Également appelé Ordinogramme ou Organigramme.

**Syntaxe** : Règles d'écriture d'un langage donné.

Un algorithme est une suite 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, et cela indépendamment des données.

## Double problématique de l'algorithmique

---

- 1) Trouver une méthode de résolution (du problème à traiter)
- 2) Trouver une méthode **efficace**.

Savoir résoudre un problème est une chose, le résoudre efficacement en est une autre. La suite du parcours vous permettra de vous en rendre compte.

## Différences entre « Algorithme » et « programme »

---

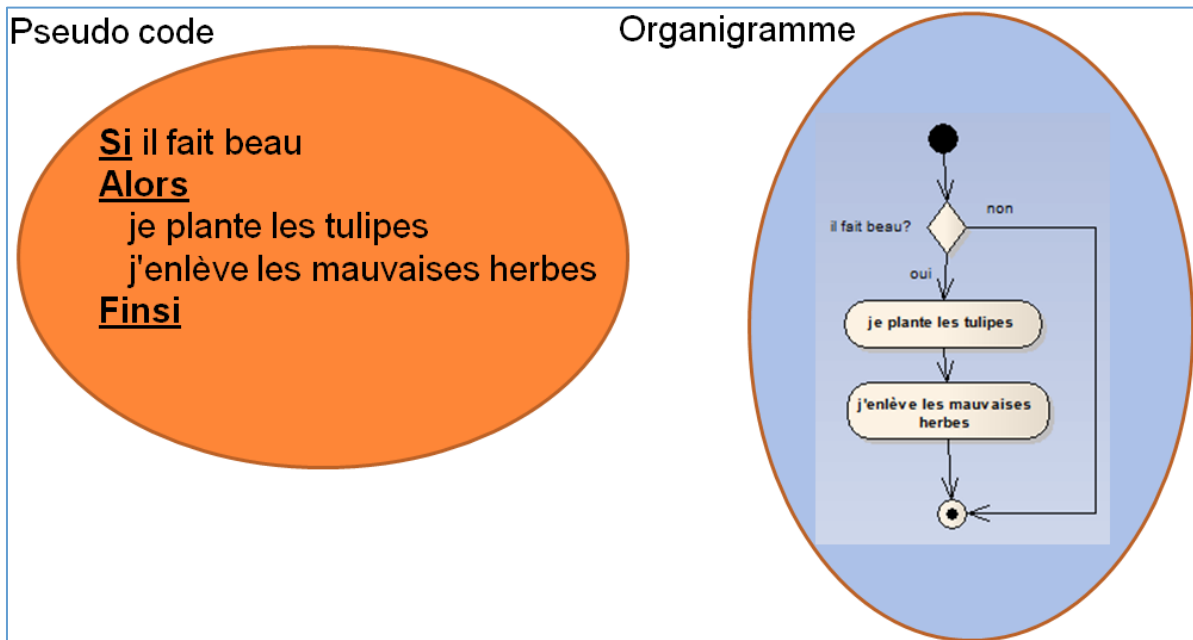
Un programme est la réalisation (l'implémentation) d'un algorithme au moyen d'un langage donné (sur une architecture donnée). Il s'agit de la mise en œuvre du principe.

Par exemple, lors de la programmation, on s'occupera par exemple de la gestion de la mémoire et de la persistance des données qui sont tous deux des problèmes d'implémentation ignorés au niveau algorithmique.

## Représentation et Structures algorithmiques

Un algorithme est composé d'un ensemble de structures ordonnant à un processeur de réaliser dans un ordre précis un nombre de tâches élémentaires dans le but de résoudre un problème technique donné.

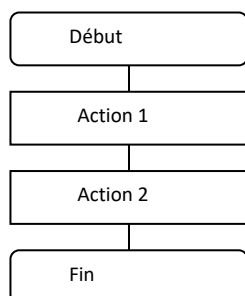
Un algorithme peut être décrit sous forme graphique (Algorithme ou Organigramme) ou sous forme littérale (notation algorithmique, pseudo code).



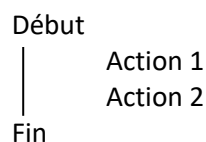
## Structure linéaire

On exécute successivement une suite d'action dans l'ordre de leur énoncé.

Algorithme



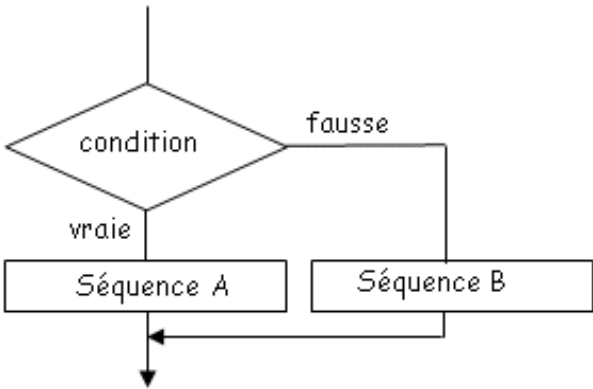
Notation algorithmique



## Structures alternatives

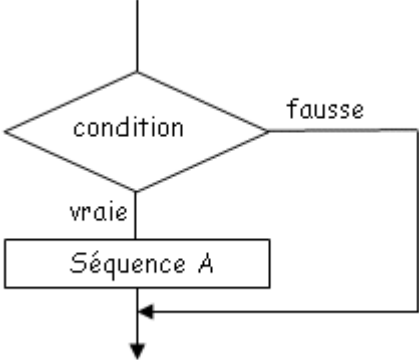
### Structure SI...ALORS...SINON...

Cette structure offre le choix entre deux séquences s'excluant mutuellement.

<p>Algorithme</p>  <pre> graph TD     Start(( )) --&gt; Condition{condition}     Condition -- vraie --&gt; SeqA[Séquence A]     Condition -- fausse --&gt; SeqB[Séquence B]     SeqA --&gt; Exit(( ))     SeqB --&gt; Exit     </pre>	<p><b><u>Notation algorithmique</u></b></p> <p><b>Si</b> condition <b>Alors</b>              Séquence A  <b>Sinon</b>              Séquence B  <b>Fin Si</b></p> <p><b><u>Exemple de Code</u></b></p> <pre> if ( condition ) {     Séquence A; } else {     Séquence B; } </pre>
--	--

#### **Remarque :**

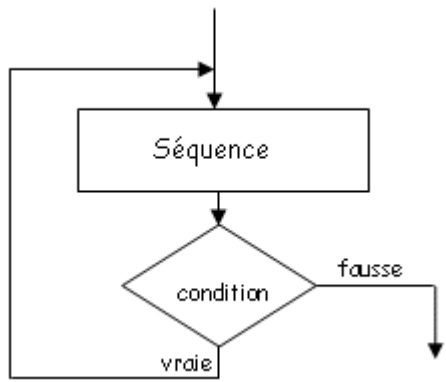
La structure peut se limiter à SI...ALORS, si la condition est vraie on exécute la séquence A si elle est fausse on quitte la structure sans exécuter de séquence.

 <pre> graph TD     Start(( )) --&gt; Condition{condition}     Condition -- vraie --&gt; SeqA[Séquence A]     Condition -- fausse --&gt; Exit(( ))     SeqA --&gt; Exit     </pre>	<p><b><u>Notation algorithmique</u></b></p> <p><b>Si</b> condition <b>Alors</b>              Séquence A  <b>Fin Si</b></p> <p><b><u>Exemple de Code</u></b></p> <pre> if ( condition ) {     Séquence A ; } </pre>
---	--

## Structures répétitives (ou itératives)

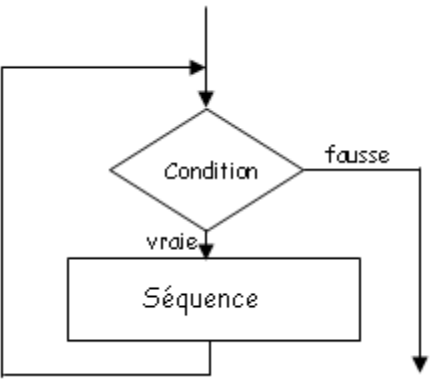
### Structure FAIRE...JUSQU'À

La séquence est exécutée au moins une fois, elle est répétée tant que la condition est vraie.

Algorithme	La traduction en algorithme peut se faire de 2 façons	
 <pre> graph TD     Start(( )) --&gt; Seq[Séquence]     Seq --&gt; Cond{condition}     Cond -- vraie --&gt; Seq     Cond -- fausse --&gt; Exit(( )) </pre>	<b>Faire</b> Séquence <b>Tant Que</b> (condition vraie)	<b>Faire</b> Séquence <b>Jusqu'à</b> (condition fausse)
	<b>Exemple de Code</b>  <pre> do {     Séquence ; } while (condition vraie) </pre>	

### Structure TANT QUE...FAIRE

On teste d'abord la condition. La séquence est exécutée tant que la condition est vraie.

Algorithme	Traduction en algorithme	
 <pre> graph TD     Start(( )) --&gt; Cond{Condition}     Cond -- vraie --&gt; Seq[Séquence]     Seq --&gt; Cond     Cond -- fausse --&gt; Exit(( )) </pre>	<b>Tant que</b> (condition vraie) Séquence <b>Fin Tant que</b>	
	<b>Exemple de Code</b>  <pre> while (condition vraie) {     Séquence; } </pre>	

## Structure POUR...FAIRE

On connaît le nombre d'itérations à réaliser.

Algorithme	Traduction en algorithme
<pre> graph TD     Start(( )) --&gt; Init[i = 0]     Init --&gt; Decision{i &lt;= N}     Decision -- vraie --&gt; Seq[Séquence]     Seq --&gt; Inc[i = i + 1]     Inc --&gt; Decision     Decision -- fausse --&gt; Exit(( ))         </pre>	<b>Pour</b> i = 0 à N <b>Faire</b> Séquence <b>Fin Pour</b>
	<b>Exemple de Code</b>  <pre> int N = 10 ;  for ( int i = 0; i &lt;= N; i++ ) {     Séquence; }         </pre>

## Choix d'un langage de programmation

Avant de se demander quel langage utiliser pour réaliser telle application , il faut se poser la question « comment vais-je résoudre mon problème algorithmique ».

La difficulté première est d'élaborer le bon algorithme pour résoudre un problème : cela demande de la recherche, beaucoup de réflexion et ce d'autant plus que le problème à résoudre est complexe.

Une fois l'algorithme mis en place, on peut passer à la phase « codage », c'est-à-dire à la phase de traduction dans un langage donné : C, C++, C#, Java, PHP... Cela ne présente pas de difficulté majeure si ce n'est la connaissance de la syntaxe des différents langages (Une fois que vous connaissez un langage dérivé du C, apprendre un autre dérivé du C prend peu de temps).

Un bon programmeur est avant tout un bon algorithmicien qui saura ensuite exploiter au mieux tel ou tel langage de programmation pour réaliser le programme demandé.

Le php ou le C# ASP.NET sont particulièrement pratique pour le développement d'applications sur internet car ils disposent d'une bibliothèque de ressources importantes. Java et C# sont un bon choix pour les applications de bureau et mobile. Les langages plus bas niveau (dont la syntaxe s'approche du langage machine) sera réservé aux applications dont les performances et la réactivité sont les critères principaux. De plus, il existe des langages destinés à des contextes et domaines très ciblés (recherche scientifique, recherche et développement, cryptographie...).

Le choix du langage de programmation se fera donc en fonction de critères pratiques : facilité de codage (bibliothèque d'instructions prédéfinies), rapidité d'exécution, disponibilité du langage pour le processeur ou le serveur ou encore affinités des équipes de développeurs avec tel ou tel outil.

--- FIN DU DOCUMENT ---

<http://www.arfp.asso.fr>