INTRODUCTION A L'ALGORITHME OU PSEUDO CODE

TABLE DES MATIÈRES

Introduction....1
But de l'algorithme....1
DÉfinition....1
Les concepts....1
GÉNÉRALITÉs....2
Les diffÉrentes phases du dÉveloppement logiciel :....2
Analyse et conception....2
Conception....2
Codage....3
Tests....3
rÈgles ALGORITHMIQUES....3
Pourquoi faire de l'algorithmique et pas simplement de la programmation ?....4
Conclusion....4
COMPLÉMENTS de cours....5
ExÉcution d'un programme....5

INTRODUCTION

BUT DE L'ALGORITHME

Un algorithme c'est une solution (parmi d'autres) permettant de résoudre un problème informatique. Cela suppose qu'il faut parfaitement comprendre le problème, pour pouvoir y apporter la meilleure solution.

L'objectif est d'obtenir de la machine qu'elle effectue un travail à notre place de façon automatique.

C'est la définition même d'Informatique = Information + automatique : traitement automatique de l'information

Le problème est : comment expliquer à la machine comment elle doit s'y prendre.

DÉFINITION

Un algorithme est une suite finie de règles à appliquer dans un ordre déterminé sur un nombre fini de données, pour arriver à un certain résultat, et cela indépendamment des données.

Cette définition quelque peu abstraite n'est que le reflet de chose que l'on fait quotidiennement:

- Lorsque vous achetez des planches de bois dans un magasin suédois, vous suivez une notice de montage, pour obtenir un meuble en kit.
- À partir de la farine, œufs, chocolat, et en appliquant une recette, on obtient un bon dessert au chocolat.

Dans ses 2 exemples, les algorithmes correspondent à la notice de montage ou à la recette.

La difficulté dans le cas d'un algorithme pour résoudre un problème informatique, est que celui qui va résoudre le problème n'est pas un être humain, mais l'ordinateur. Et qu'un ordinateur ne fonctionne pas du tout de la même façon qu'un humain. Il faut donc comprendre le fonctionnement d'un ordinateur pour trouver la solution.

LES CONCEPTS

GÉNÉRALITÉS

L'informatique est en perpétuelle évolution : programmation procédurale, événementielle, et objet. Si **l'analyse** d'un projet informatique et l'architecture des systèmes ont beaucoup changé, la phase de **conception** détaillée doit toujours répondre à la même question :

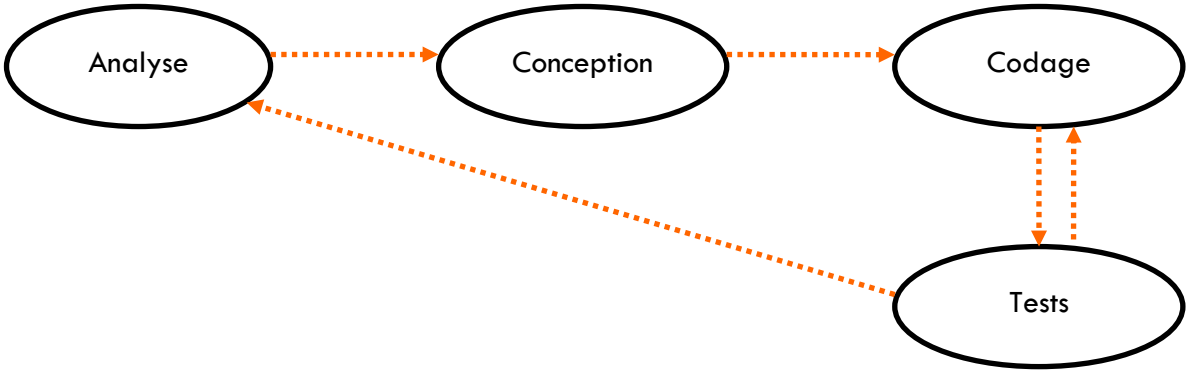
Comment décrire de façon claire et simple le comportement d'un module (procédure, module, objet) pour l'implanter dans une machine informatique.

Cette description doit bien sûr être indépendante de la machine et des langages de programmation.

L'algorithme sera donc décrit dans un langage naturel (le français), et n'est donc pas un langage de programmation. On parle de **pseudocode**. Celui-ci est **codé** dans un langage de programmation (VB, Pascal, ADA, C, JavaScript) pour être exécuté puis **testé**. (Voir figure suivante).

En résumé : Un **algorithme**, traduit dans un langage compréhensible par l'ordinateur (ou langage de programmation), donne un **programme**, qui peut ensuite être exécuté, pour effectuer le **traitement** souhaité.

LES DIFFÉRENTES PHASES DU DÉVELOPPEMENT LOGICIEL :



ANALYSE ET CONCEPTION

Dans cette phase, il s'agit de comprendre le problème. Il faut analyser quelles sont les informations à traiter. Réfléchir aux traitements à réaliser.

Il y a plusieurs façons d'analyser:

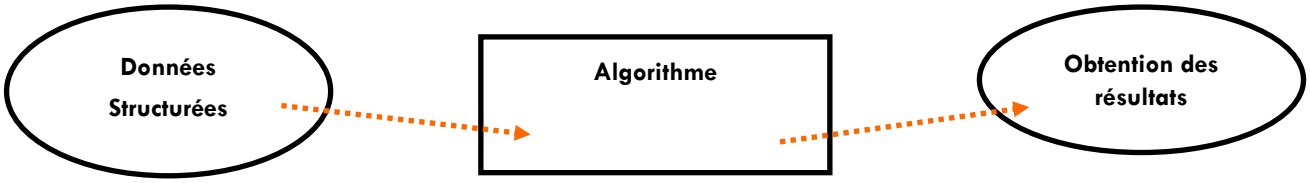
- Si le problème est d'ordre mathématique, on définira des **règles de calculs** permettant de passer des données aux résultats.
- Si le problème n'est pas d'ordre mathématique, on fera une **analyse par cas**. Le but étant de regrouper les cas ayant un même traitement.
- Combiner analyse par cas et règles de calculs.

Cette étape donne lieu à un **dictionnaire de données** précisant pour chaque donnée leur nom, et leur nature (entier, caractère, chaîne de caractère, réel …)

CONCEPTION

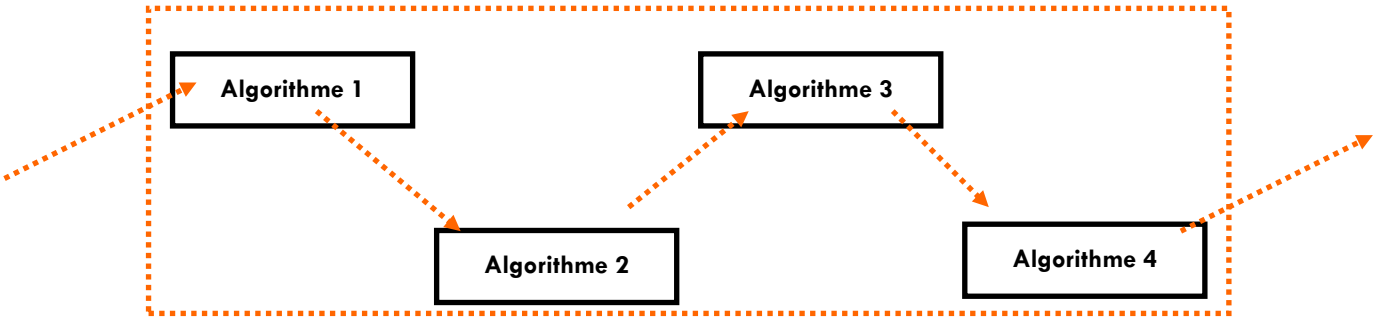
Il s'agit de **l'algorithme** lui-même.

Toute portion, algorithme ou activité, peut apparaître comme un bloc avec un point d'entrée et un point de sortie et contenant une suite structurée d'instructions



Attention, si l'algorithme est trop compliqué, on le décomposera en sous-algorithme. C'est le principe même de la programmation structurée : pouvoir décrire le comportement d'un problème de façon descendante (du plus global au plus détaillé) et de décrire à chaque niveau qu'une quantité d'informations limitée afin d'en permettre une compréhension rapide.

Le problème final sera vu comme un enchaînement d'algorithmes



CODAGE

C'est la traduction des différents algorithmes dans un langage de programmation. Si l'algorithme est décrit dans un langage naturel (le français), le codage se fait avec respect des syntaxes des langages. Le compilateur est là pour vérifier ce point.

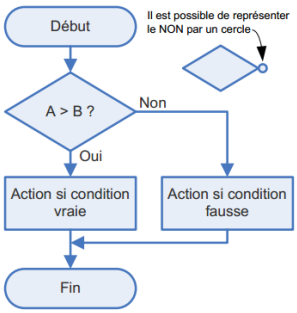
Il existe plusieurs modes de programmation : procédural ou séquentiel, par objet, événementiel. Quel que soit le mode de programmation choisi, **la rédaction d'un algorithme est primordiale**.

TESTS

Cette phase consiste à vérifier si les résultats obtenus sont conformes à ceux attendus dans l'étape d'analyse. Si ce n'est pas le cas, il faut reprendre l'étape d'analyse.

RÈGLES ALGORITHMIQUES

Les algorithmes peuvent être représentés par divers formalismes : représentation graphique (avec des carrés, des losanges ... = organigrammes), ou bien textuelle dans un langage proche du langage naturel.



Aujourd'hui, cette représentation est quasiment abandonnée, pour 2 raisons :

1- Dès que l'algorithme commence à grossir un peu, ce n'est plus pratique du tout du tout.

2- Cette représentation favorise le glissement vers un certain type de programmation, dite non structuré que l'on tente d'éviter

- Même s'il n'existe aucun standard pour les représenter, chaque activité (ou algorithme) sera représenté de façon **textuelle** ou **pseudocode** dont la longueur ne doit pas dépasser une page. Elle doit être facile à lire et à comprendre. Il faut donc utiliser des termes clairs et parlants pour décrire une activité, remplacer tout développement confus par une sous activité. Le module principal apparaît comme une suite d'appels à des activités claires. Ce pseudocode est susceptible de varier légèrement d'un ouvrage (ou d'un enseignant) à l'autre. Le pseudocode est purement conventionnel : aucune machine n'est censée le reconnaître telle quelle

  Exemple :

```
ACTIVITE Principale
      VAR a, b, c : Entiers
DEBUT
      a ← 5
      b ← 3
      c ← a + b
      a ← 7
      c ← b – a
FIN
```

- Chaque activité à un seul point d'entrée et un seul point de sortie. Chaque activité est décrite exclusivement à partir des 3 structures fondamentales de contrôle:

- **SÉQUENCE : suite d'opérations (affectation, lecture, écriture …)**
- **ITÉRATION : les boucles**
- **ALTERNATIVE : les tests**

- La description algorithmique sera accompagnée d'un **dictionnaire de données** comportant toutes les données utilisées dans la description.
- Pour chaque activité la description commencera par son nom, suivi de la déclaration des données locales utilisées.

POURQUOI FAIRE DE L'ALGORITHMIQUE ET PAS SIMPLEMENT DE LA PROGRAMMATION ?

Pourquoi apprendre l'algorithmique pour apprendre à programmer ?

En quoi a-t-on besoin d'un langage spécial, distinct des langages de programmation compréhensibles par les ordinateurs ?

L'algorithmique exprime les instructions résolvant un problème donné indépendamment des particularités de tel ou tel langage.

Apprendre l'algorithmique, c'est apprendre à manier la structure logique d'un programme informatique. Cette dimension est présente, quel que soit le langage de programmation.

Quand on programme dans un langage, on doit en plus gérer les problèmes de syntaxe, ou de types d'instructions, propres à ce langage.

Apprendre l'algorithmique de manière séparée, c'est donc dissocier les difficultés pour mieux les vaincre.

De nombreux programmeurs, souvent autodidactes ont directement appris à programmer dans tel ou tel langage. La conséquence est qu'ils ne font pas mentalement clairement la différence entre ce qui relève :

- Des règles fondamentales de l'algorithmique
- Du langage particulier qu'ils ont appris.

Ces programmeurs, non seulement ont beaucoup plus de mal à passer ensuite à un langage différent, mais encore écrivent bien souvent des programmes qui même s'ils sont justes, restent laborieux.

Enfin c'est un moyen de communication entre informaticiens et non-informaticiens.

L'algorithmique, l'art d'écrire des algorithmes, permet de se focaliser sur la procédure de résolution du problème sans avoir à se soucier des spécificités d'un langage particulier.

CONCLUSION

Le fait de bien savoir manier un langage de programmation ne vous permettra pas de résoudre un problème si vous ne savez pas comment le résoudre a priori.

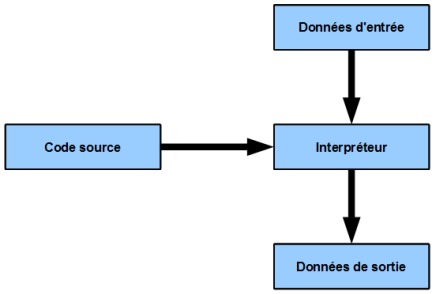
◦ Vous serez évalué sur la capacité de votre solution à résoudre le problème donné, on veut une solution « qui marche ». Et ça ne sera pas le cas si votre algorithme à la base est faux.

◦ ON NE SE LANCE PAS DANS LE CODE SANS RÉFLÉCHIR AU PRÉALABLE ET SANS SAVOIR CE QU'ON CHERCHE À FAIRE !!!

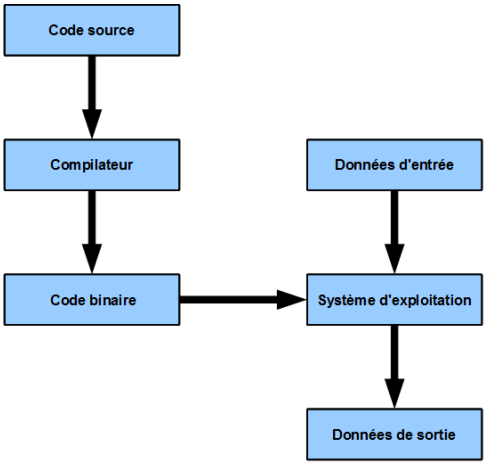
COMPLÉMENTS DE COURS

Le programme (code) établi peut ensuite être :

- Interprété : PHP, scripts Bash, Perl, JavaScript : dans ces langages, le code source (celui que vous écrivez) est interprété, par un logiciel (un programme) qu'on appelle interpréteur. Celui-ci va utiliser le code source et les données d'entrée pour calculer les données de sortie. L'interprétation du code source est un processus « pas à pas » : l'interpréteur va exécuter les lignes du code une par une, en décidant à chaque étape ce qu'il va faire ensuite
- compiler puis interpréter : Java. La compilation du code produit un "bytecode Java" qui est ensuite exécuté par une machine virtuelle Java.



- Compilé : C, Pascal. La compilation produit un fichier exécutable (.exe). Dans ces langages, le code source (celui que vous écrivez) est tout d'abord compilé, par un logiciel qu'on appelle compilateur, en un code binaire qu'un humain ne peut pas lire. C'est alors directement le système d'exploitation, plus précisément le processeur qui va utiliser le code binaire et les données d'entrée pour calculer les données de sortie.



On pourrait discuter très longtemps des avantages et inconvénients des différents types de langages, mais les deux points qui sont les plus intéressants sont les suivants :

Dans un langage interprété, le même code source pourra marcher directement sur tout ordinateur. Avec un langage compilé, il faudra (en général) tout recompiler à chaque fois ce qui pose parfois des soucis.

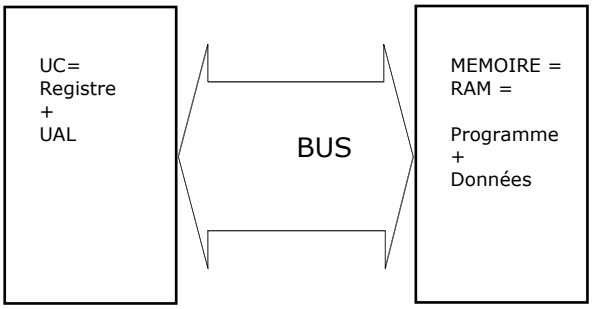
Dans un langage compilé, le programme est directement exécuté sur l'ordinateur, donc il sera en général plus rapide que le même programme dans un langage interprété.

Quel que soit le mode de programmation choisi, la rédaction d'un algorithme est primordiale.

EXÉCUTION D'UN PROGRAMME

Le programme est une suite de 0 et de 1, même chose pour les données.

L'UC (unité centrale) dialogue en permanence avec la mémoire vive (RAM) pour récupérer les instructions du programme à exécuter (1 instruction = 1 série de 0 et de 1), ou bien les données par l'intermédiaire d'un bus.



Un processeur de 3Ghz est un processeur qui est capable d'exécuter 3 x 10^9 instructions à la seconde !!!