Algorithmie et Programmation

Introduction.

---

1. Le cours.

Objectifs

- Etre capable d'analyser un problème et de conceptualiser la solution sous la forme d'un algorithme.
- Implémenter une solution structurée, lisible, commentée, sûre et efficace à l'aide d'un langage de programmation.
- Maîtriser les formants algorithmiques de base (séquence, sélection, itération).
- Manipuler les algorithmes classiques de recherche et de tri de tableaux de données.
- Concevoir et mettre en œuvre les structures de données tableaux et enregistrements

Phases

- Les concepts fondamentaux de la programmation (variables, affections, contrôles sélectif et itératif, procédures et fonctions).
- Les structures de données tableaux et enregistrement.
- La méthode d'analyse descendante.
- Les fonctions principales (compilation, exécution, deboguage) d'un environnement de développement.

2. Définition d'un algorithme.

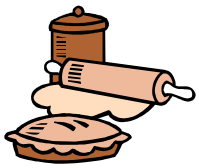
Le mot ***algorithme*** est dérivé du nom d'un mathématicien perse qui a vécu au IX^{ème} siècle, Mohammed al-Khwârizmî (en latin Algorismus). Il a proposé un ensemble d'opérations élémentaires à exécuter séquentiellement, pour additionner, soustraire, multiplier et diviser des nombres décimaux.

Voici donc une définition :
Un algorithme consiste en la description d'un certain nombre d'opérations élémentaires selon un ordre logique permettant de résoudre un problème sur des données en un nombre fini d'étapes.

Les algorithmes ne sont pas réservés au seul domaine de l'informatique, ils font partie de notre vie quotidienne. Suivre une recette d'un livre de cuisine, déchiffrer un mode d'emploi d'installation consistent à **exécuter** des algorithmes.

Avez-vous déjà indiqué un chemin à un touriste égaré, demandé à un correspondant téléphonique de chercher un objet, écrit une lettre anonyme en indiquant comment procéder à une remise de rançon ? Si oui, vous avez déjà **fabriqué** et fait **exécuter** des algorithmes. Un algorithme c'est donc une suite d'instructions, qui une fois exécutée correctement conduit à un résultat donné. Si l'algorithme est juste, le résultat est celui voulu, et le touriste se trouve là où il voulait aller. Si l'algorithme est faux, le résultat est disons comme aléatoire et le touriste se perd.

3. Différence entre algorithme et programme.

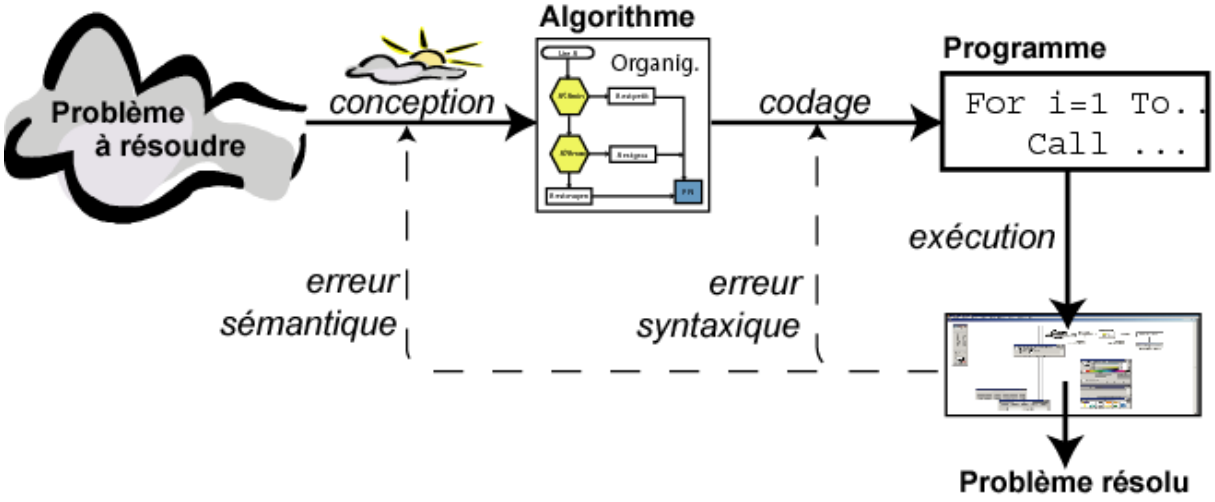


Si une recette de cuisine correspond à un algorithme, la mise en œuvre de cette recette est le programme. En effet, pour faire une tarte aux myrtilles, il faut toujours de la pâte et des myrtilles, mais l'algorithme ne vous dit pas si vous devez réaliser aussi la pâte ou si vous devez l'acheter dans le commerce déjà prête, de même il ne vous dit pas si vous devez cueillir les myrtilles vous-même ou les acheter congelées. Ainsi, un programme est un algorithme dans le sens où c'est la description d'une méthode mécanique de résolution d'un problème donné. Exécuté par un ordinateur, un programme tient en plus compte de l'environnement informatique de programmation et d'exécution comme le langage, le(s) processeur(s), l'interface graphique, la capacité mémoire… Un programme adapte voire optimise l'algorithme à cet environnement.

Cette phase de transformation de l'algorithme en un programme exécutable est dite d'implémentation. Au final, la performance du programme finale ne dépend pas seulement de l'efficacité de l'algorithme mais aussi de la qualité de la mise en œuvre et la convivialité de l'interface homme-machine. Une tarte aux myrtilles préparée avec des myrtilles fraichement cueillies est souvent meilleure, agrémentée d'une pointe de chantilly et c'est carrément le paradis !

4. Fabrication d'un programme.

La chaîne de fabrication d'un programme se décompose comme suit :



Détaillons ces phases :

1. **Analyse et Conception** : on établit précisément le problème à résoudre, les données de départ et ce que l'on souhaite obtenir en résultat, puis, on raisonne sur la manière de représenter les données du problème, et sur une méthode de résolution. Cette méthode est ensuite énoncée sous forme de suite de pas à accomplir pour aboutir aux solutions : c'est *l'algorithme de résolution du problème*.
    - La représentation des données définit le *modèle des données.*
    - L'énoncé pas à pas des traitements détermine *l'algorithme*
2. **Codage** : il s'agit de la phase d'implémentation, traduire l'algorithme en programme au moyen d'un langage de programmation. Ce programme prend la forme d'un fichier texte appelé *code source*. Pour s'exprimer l'homme utilise un langage que la machine ne comprend pas. La machine comprend le binaire (deux états) que l'homme ne parle pas. Le langage de programmation permet à l'homme de « transmettre » sa logique à la machine. Mais la machine l'applique bêtement.

3. **Mise au point** : comprend le plus souvent plusieurs étapes répétées jusqu'a ce que le programme semble satisfaisant :

    - Compilation : un langage de programmation de haut niveau n'est pas directement compréhensible par la machine. Il faut donc traduire le *code source* du programme vers le *langage natif* de la machine (parfois virtuelle). Le résultat est un nouveau fichier écrit en langage machine, et appelé *code machine (ou objet)*. Il est prêt pour l'exécution.
    - Tests : exécution du code machine avec divers cas typiques des entrées par des *jeux de tests*. C'est là ou la plupart des erreurs apparaissent.
    - Correction d'erreurs : la première fois que l'on exécute son programme, on n'aboutit pas toujours aux résultats voulus. En cas d'erreur, il faudra donc modifier ce programme. L'erreur peut alors être de deux types, sémantique ou syntaxique. Dans le premier cas, le programme s'exécute, mais les résultats ne sont pas ceux attendus : le sens de l'algorithme est alors à remettre en cause, c'est la réflexion qu'il faut reprendre car la qualité du travail préparatoire n'était pas à la hauteur. Dans le second cas, le programme ne s'exécute même pas, vous obtenez un message d'erreur du type: *« Syntax error on line 848 »*. Dans ce cas, c'est simplement une question de syntaxe à rectifier. Après toute correction on recommence le cycle compilation, exécution et tests, etc. Jusqu'à que cela fonctionne.

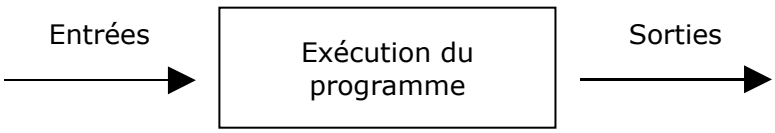
4. **Maintenance** : il s'agit le plus souvent de la correction des erreurs apparues après la mise en service, mais aussi de la modification du programme pour s'adapter à des nouvelles spécifications du problème.

Un algorithme donne le **sens** du programme. L'implémentation consiste à donner la **forme** au programme grâce à la syntaxe du langage choisi.

5. Analyse des problèmes.

Un programme est donc la description d'une méthode mécanique (applicable à une machine) de résolution d'un problème donné. La description est donnée par une suite d'instructions d'un langage de programmation, et ces instructions ont pour but de traiter et transformer les données du problème à résoudre, jusqu'à aboutir à une solution.

Un programme est en quelque sorte une méthode à suivre pour trouver les solutions, mais n'est pas une solution en soi. L'exécution du programme permet de trouver les solutions du problème :



Les traitements décrits par les instructions sont appliqués aux données (entrées). Et les données ainsi transformées permettent d'aboutir aux solutions recherchées (sorties).

Voici une démarche possible pour fabriquer un programme à partir de l'énoncé d'un problème :

1. *Déterminer le problème à résoudre :*
   - Quelles sont les données d'entrée et leur nature ?
   - Quelles sont les données attendues en sortie et leur nature ?

2. *Déterminer la méthode :*
   - Comment modéliser les données (d'entrée, de sortie, intermédiaires) ?
   - Quels sont les différents cas des entrées à traiter, et le cas échéant, quels sont les traitements de contrôle associés.
   - Exprimer sous-forme d'algorithme la manière d'obtenir le résultat final à partir des données d'entrée.

3. *Correction, complétude et lisibilité*
   - A-t-on bien prévu tous les cas des entrées et sorties ?
   - Obtient-on dans chaque cas ce que l'on voulait calculer ?
   - Notre algorithme est-il compréhensible par quelqu'un d'autre ?

4. *Tests*
   - Elaborer un jeu de tests représentatif de tous les cas possibles (univers) des entrées. Un *jeu de tests* consiste en une suite de « valeurs typiques » pour chacune des entrées, accompagnées des valeurs attendues comme résultat dans ce cas.
   - Confronter le fonctionnement de l'algorithme avec le jeu de tests proposé.

Un exemple : fabriquer un programme de conversion de monnaie de CHF vers €.

- Données du problème :

Donnée en entrée : une somme à convertir (ex: 55CHF).
Donnée en sortie : une somme, résultat de la conversion (ex: 37.4€).
Donnée constante du problème : le taux de conversion (ex. :0.68€).

- Modélisation :

La donnée en entrée est une variable réelle que l'on nomme : `sommeChf`.
La donnée en sortie est une variable réelle que l'on nomme : `sommeEuro`.
La constante du taux de conversion est une constante réelle que l'on nomme : `tauxEuro`.

- Traitement :

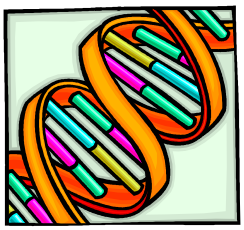
Traitement transformant les entrées pour aboutir au résultat en sortie :

```
sommeEuro = sommeChf * tauxEuro
```

- Jeux de tests :

SommeChf	SommeEuro
55	37.4
19.7	13.464
-10	-6.8

6. Les éléments de base de la programmation.



Faisons une comparaison rapide avec l'ADN, qui est en quelque sorte le programme génétique, l'algorithme à la base de la construction des êtres vivants. C'est une chaîne construite à partir de quatre éléments invariables. Ce n'est que le nombre de ces éléments et *l'ordre dans lequel ils sont arrangés* qui vont déterminer si on obtient une puce ou un éléphant. L'Homme est donc le résultat de l'exécution d'un programme comportant une suite de ces quatre briques.
Un programme informatique est une suite d'instructions. Une instruction est un ordre donné à l'ordinateur. Les ordinateurs, quels qu'ils soient, ne sont fondamentalement capables d'exécuter que quatre ordres (je n'inclus pas ici les opérations de calcul). Ces ordres sont, par extension du théorème fondamental de Böhm et Jacopini[1] :

- l'affectation de variables
- la lecture / écriture
- la sélection : un ordre à exécuter sous certaines conditions
- la répétition : un ordre à répéter plusieurs fois

Un programme informatique se ramène donc toujours au bout du compte à la combinaison de ces quatre « briques » de base. Il peut y en avoir quelques unes, quelques dizaines, et jusqu'à plusieurs centaines de milliers dans certains programmes.
La suite décrit donc tous les éléments de base de la programmation structurée, les « outils » à disposition du programmeur pour écrire un programme. Tous ces outils ont leur équivalent dans tout langage de programmation.

7. L'ordinateur : quelques rappels.

Composants principaux:

- Le processeur ou unité centrale (CPU) pour le traitement des données et instructions logées en mémoire centrale. Capable d'exécuter un ensemble d'instructions dites instructions machine. Il s'agit d'opérations simples telles que la lecture/écriture en mémoire centrale, opérations arithmétiques et logiques, comparaison des valeurs, branchement (saut) vers une adresse, etc. Ces instructions sont codées en binaire et diffèrent dans chaque plateforme matériel. Ainsi, une instruction qui s'exécute sur un processeur Intel/AMD/x86 ne peut pas s'exécuter sur un processeur Sun/SPARC.

- Mémoire Centrale (RAM), très rapide (accès directe), mais volatile. Cette mémoire est organisé comme une suite d'emplacements appelés mots et munis chacun d'une adresse. On doit y stocker les données à traiter, ainsi que les instructions machine à exécuter. Tout programme à exécuter et toute donnée à traiter doit être chargée en mémoire centrale au préalable.

- Périphériques. Dispositifs de communication de l'ordinateur pour l'acquisition, la production, la communication des données: clavier, écran, enceintes, ports de communication; et pour le stockage persistent tels que disques durs, disquettes, etc.

[1] Théorème fondamental : les informaticiens Böhm et Jacopini ont démontré que trois schémas de base sont nécessaires et suffisants pour décrire tout algorithme. Il s'agit de :
- La séquence : les instructions s'effectuent les unes après les autres dans l'ordre indiqué par l'algorithme.
- La sélection : des instructions s'effectuent selon un choix, soit cette instruction, soit l'autre.
- La répétition : des instructions s'effectuent de manière répétitive, tant qu'une condition est satisfaite.

8. Historique de la programmation.

Pour conclure cette introduction, un petit historique de la programmation s'impose. Cet historique n'est pas exhaustif, car on pourrait remonter en Mésopotamie. On présente donc ici quelques dates importantes et les langages les plus populaires. Pour plus de détails, le lecteur est invité à parcourir le remarquable site : http://histoire.info.online.fr.
En parallèle il est aussi utile de consulter l'historique de l'informatique.

- 1854 : George Boole publie un ouvrage dans lequel il démontre que tout processus logique peut être décomposé en une suite d'opérations logiques (ET, OU, NON) appliquées sur deux états (0-1, Oui-Non, Vrai-Faux, Allumé-Eteint).
- 1937 : George Stibitz crée le premier circuit binaire, un additionneur. Il l'appelle le Model K (pour Kitchen) car il l'a créé dans sa cuisine avec une planche à pain !
- 1938 : Thèse de Shannon qui fait le parallèle entre les circuits électriques et l'algèbre Booléenne. Il définit le chiffre binaire : bit (BInary digiT).
- 1945 : Un insecte coincé dans les circuits bloque le fonctionnement du calculateur Mark1. La mathématicienne Grace Murray Hopper décide alors que tout ce qui arrête le bon fonctionnement d'un programme s'appellera Bug. (Il semble que l'expression soit restée ;-)

  Fort de ces résultats, les ingénieurs commencèrent à écrire des programmes binaires de ce genre :

  ```
  01000110
  11000100
  10111011
  00011101
  ```

  Concevoir des programmes via le langage binaire prenait énormément de temps pour de faibles résultats.
- 1950 : Alan Turing et Maurice V. Wilkes de l'université de Cambridge ont eu la bonne idée de brancher un clavier à l'ordinateur. Grâce à cette invention, la longue suite de 1 et de 0 à saisir s'est transformée en commandes appelées: "instructions machines". Ainsi la série de bits ci-dessus est devenue :

  ```
  ADD A,6
  MOV A,OUT
  LOAD B
  SUB B,A
  ```

  Ce langage se nomme l'ASSEMBLEUR. Il est constitué de mots (ADD, MOV, etc...) appelés mnémoniques.
  Ces mnémoniques ne sont pas directement comprises par l'ordinateur. Ils doivent d'abord passer par un programme de traduction qui les transforme en 1 et 0. L'ASSEMBLEUR est considéré comme un langage de bas niveau car il est fortement lié au processeur de l'ordinateur. Chaque processeur à son jeu d'instructions.
  Pour pouvoir utiliser un même programme sur un ordinateur possédant un processeur différent, il fallait réécrire une grande partie ou la totalité du programme. C'est pour résoudre ce genre de problème que les langages de haut niveau sont arrivés.
- 1957 : Le Fortran fut créé par John Backus d'IBM. Fortran signifie FORmula TRANslator (traducteur de formules). Il est utilisé pour les applications mathématiques et scientifiques. Le Fortran est plus compréhensible que l'Assembleur car il utilise des mots anglais tels que If, Write ou Format. Il est plus proche du langage parlé que les 0 et les 1 du langage machine. Malheureusment, il offre un nombre restreint de commandes.

- 1960 : Arrivée du COBOL (Common Busisness Oriented langage) développé par Grace Murray Hopper, l'amiral de la marine américaine qui avait découvert le premier bogue. Elle créa avec son équipe de programmeurs un langage pour les applications commerciales. Ce langage permet de faire des inventaires, des factures ou encore des données individuelles du personnel.

  D'autres langages suivirent.

- 1964 : Naissance du BASIC (Beginner's All-purpose Symbolic Instruction Code : code d'instructions symboliques polyvalent pour débutants) créé par Thomas Kurtz et John Kemeny du Darmouth College pour leurs étudiants. Ils se sont inspirés du Fortran qui était trop complexe pour les étudiants non mathématiciens.

- 1966 : Le LOGO fut créé par une équipe chez BBN (Bolt Beranek & Newman) dirigée par Wally Fuerzeig dont faisait partie Seymour Papert. Ce langage très graphique est basé sur le principe d'une tortue que l'on pilote à l'écran en lui donnant des ordres (tourner, avancer, etc...). Le LOGO est avant tout un langage d'apprentissage de la programmation.

  Ces langages de programmations ont eu leurs heures de gloire, ils ont apporté leurs lots de solutions. Les programmes informatiques sont devenus de plus en plus gros et de plus en plus complexes posant de nouveaux problèmes. Le temps de développement de grosses applications et leur maintenance devenaient colossaux.

- 1966 : Apparition du "paradigme de programmation structurée" de Corrado Bohm et Guiseppe Jacopini, connu aussi sous le nom de "The Structure Theorem". Selon ce paradigme, n'importe quel programme peut être construit sur la base d'une combinaison de trois structures fondamentales :
  - la séquence (l'ordre dans lequel les instructions sont exécutées),
  - la sélection (exécution d'instructions sous condition),
  - la répétition (répétition contrôlée d'un bloc d'instructions).

  Ainsi, vinrent les premiers langages structurés.

- 1968 : Niklaus Wirth crée le langage PASCAL (En l'honneur du mathématicien Blaise Pascal). Ce fut le meilleur langage de son époque, il était très performant et structuré ce qui permettait aux programmes d'être plus lisibles et plus faciles à corriger en cas d'erreurs.

- 1970 : Keth Thompson, pensant qu'Unix ne serait pas complet sans un langage de programmation de haut niveau, crée le B (En référence au BCPL dont il s'inspire). Entre 1971 et 1973, Dennis Ritchie du Bell Lab d'ATT reprend le langage B et le fait évoluer. Il nomme cette nouvelle version le C. Le C est un habile mélange de langages de haut niveau (Basic, Pascal) et de bas niveau (Assembleur). Il est extrêmement rapide.

- 1983 : Le danois Bjarn Stroustrup crée une nouvelle version du C appelé C++. C++ propose une programmation orientée objet, POO ou OOP (Object Oriented Programming) qui permet aux programmeurs d'écrire des programmes plus rapidement et surtout avec moins d'erreurs.

- 1991 : Un petit groupe d'employés de SUN travaille sur un projet de réalisation d'applications électroniques commerciales baptisé "Green". La première mission du "Green Team" est de créer un appareil capable de controler plusieurs équipements électroniques à la fois. Le langage OAK ("chêne" en français) est créé pour cette intention. L'arrivée en 1993 du World Wide Web et du navigateur Mosaic de NCSA fait jaillir les idées. SUN se rend compte que Oak est le langage idéal pour un réseau informatique hétérogène (interconnexion de plusieurs systèmes différents).

- 1994 : Le "Green Team" fait de Oak un langage basé sur le système d'exploitation (indépendant du processeur). De cette manière les applications développées peuvent fonctionner sur n'importe quelle machine, Macintosh, Windows, OS/2 ou UNIX.
- 1995 : Oak est rebatisé JAVA pour des raisons commerciales. Le nom JAVA fut choisi lors d'une pause où, à cours d'idées, l'équipe de développement baptisa le projet du terme argotique signifiant café en américain ("kawa" pour les francophones).
- 2000 : Microsoft lance son Framework.Net en réponse à J2EE (Java2 Enterprise Edition, de Sun). dotNet inclut plusieurs langage orientés objet, dont C# inspiré par Java et VB.Net (version 7 de Visual Basic)
- 2006 : JAVA SE 6 (nom de code « Mustang ») sort en version finale le 12 décembre, incluant plus de 3500 classes et interfaces.

A suivre…