CARNUS Enseignement supérieur RODEZ

Algorithmique et Programmation

Prof : Kamal Boudjelaba

25 novembre 2024

2024–2025

BTS CIEL

Carnus Enseignement Supérieur
Avenue de Saint Pierre - Bourran - 12000 RODEZ, France

IR

Table des matières

Liste des figures

1. Introduction : Définitions de l'académie

INFORMATIQUE *n. f. et adj. XXe siècle. Dérivé d'information sur le modèle de mathématique, électronique.*
1. N. f. Science du traitement rationnel et automatique de l'information ; l'ensemble des applications de cette science.
2. Adj. Qui se rapporte à l'informatique. Système informatique, ensemble des moyens qui permettent de conserver, de traiter et de transmettre l'information.

INSTRUCTION *n. f. XIVe siècle. Emprunté du latin instructio, « action d'adapter, de ranger », puis « instruction ». Ordre, indication qu'on donne à quelqu'un pour la conduite d'une affaire ; directive, consigne. Le plus souvent au pluriel. Des instructions verbales, écrites. Donner des instructions à ses collaborateurs. Instructions pour la mise en marche d'un appareil. Par anal. INFORM. Consigne formulée dans un langage de programmation, selon un code.*

LOGICIEL *n. m. XXe siècle. Dérivé de logique. INFORM. Ensemble structuré de programmes remplissant une fonction déterminée, permettant l'accomplissement d'une tâche donnée.*

MATÉRIEL *adj. et n. XIIIe siècle. Emprunté du latin materialis, de même sens. INFORM. Ensemble des éléments physiques employés pour le traitement des données, par opposition aux logiciels.*

ORDINATEUR *n. m. XVe siècle, au sens de « celui qui institue quelque chose » ; XXe siècle, au sens actuel. Emprunté du latin ordinator, « celui qui règle, met en ordre ; ordonnateur ». Équipement informatique comprenant les organes nécessaires à son fonctionnement autonome, qui assure, en exécutant les instructions d'un ensemble structuré de programmes, le traitement rapide de données codées sous forme numérique qui peuvent être conservées et transmises.*

Remarque 1.1

Le mot INFORMATIQUE n'a pas vraiment d'équivalent aux Etats-Unis où l'on parle de *Computing Science* (science du calcul) alors que *Informatics* est admis par les Britanniques.

2. Ordinateur

L'architecture matérielle d'un ordinateur repose sur le modèle de VON NEUMANN qui distingue classiquement 4 parties (figure 1) :

1. L'unité arithmétique et logique, ou unité de traitement, effectue les opérations de base.
2. L'unité de contrôle séquence les opérations.
3. La mémoire contient à la fois les données et le programme qui dira à l'unité de contrôle quels calculs faire sur ces données. La mémoire se divise entre mémoire vive volatile (programmes et données en cours de fonctionnement) et mémoire de masse permanente (programmes et données de base de la machine).
4. Les entrées-sorties sont des dispositifs permettant de communiquer avec le monde extérieur (écran, clavier, souris...).

Les 2 premières parties sont souvent rassemblées au sein d'une même structure : le micro-processeur (la « puce »), unité centrale de l'ordinateur.

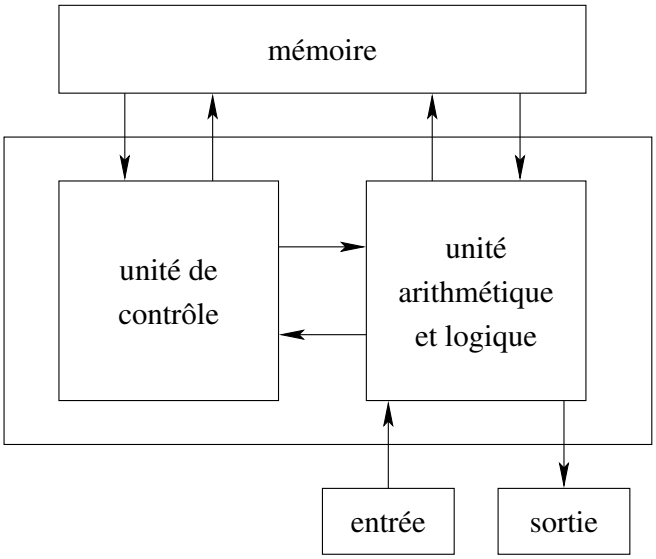

Figure 1. Architecture de VON NEUMANN

3. Algorithmique

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.*
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.

Définition 3.1 : Algorithme

Un algorithme est une suite ordonnée d'instructions qui indique la démarche à suivre pour résoudre une série de problèmes équivalents.

Dans ce cours, nous apprendrons à définir des algorithmes pour qu'ils soient compréhensibles – et donc exécutables – par un ordinateur.

Exemple 3.1 : Recette de cuisine

Une recette de cuisine peut être réduite à un algorithme si on peut réduire sa spécification aux éléments constitutifs :

- des entrées (les ingrédients, le matériel utilisé) ;
- des instructions élémentaires simples (frire, flamber, rissoler, braiser, blanchir, etc.) dont les exécutions dans un ordre précis amènent au résultat voulu ;
- un résultat : le plat préparé.

Cependant, les recettes de cuisine ne sont en général pas présentées rigoureusement sous forme non ambiguë : il est d'usage d'y employer des termes vagues laissant une liberté d'appréciation à l'exécutant, alors qu'un algorithme doit être précis et sans ambiguïté.

Source : Wikipédia

Définition 3.2 : Algorithmique

L'algorithmique est la science des algorithmes.

L'algorithmique s'intéresse à l'art de construire des algorithmes ainsi qu'à caractériser leur validité, leur robustesse, leur réutilisabilité, leur complexité ou leur efficacité.

Définition 3.3 : Caractéristiques d'un algorithmique

VALIDITÉ D'UN ALGORITHME : La validité d'un algorithme est son aptitude à réaliser exactement la tâche pour laquelle il a été conçu.

ROBUSTESSE D'UN ALGORITHME : La robustesse d'un algorithme est son aptitude à se protéger de conditions anormales d'utilisation.

RÉUTILISABILITÉ D'UN ALGORITHME : La réutilisabilité d'un algorithme est son aptitude à être réutilisé pour résoudre des tâches équivalentes à celle pour laquelle il a été conçu.

COMPLEXITÉ D'UN ALGORITHME : La complexité d'un algorithme est le nombre d'instructions élémentaires à exécuter pour réaliser la tâche pour laquelle il a été conçu.

EFFICACITÉ D'UN ALGORITHME : L'efficacité d'un algorithme est son aptitude à utiliser de manière optimale les ressources du matériel qui l'exécute.

- Pour construire un algorithme, l'essentiel est de percevoir les éléments clés d'un processus de calcul, et d'imaginer les suites d'opérations les plus astucieuses et les plus efficaces pour le mettre en œuvre de façon performante.

— Il est écrit en utilisant le langage naturel et des mots clés : Début, si, alors, tant que, jusqu'à, fin...
— L'algorithme est le squelette abstrait du programme informatique, indépendant du mode de codage particulier (langage utilisé) qui permettra sa mise en œuvre effective au sein d'un ordinateur.

Algorithm 1 EXEMPLE

Entrée(s) n entier, $n > 1$
  b prend pour valeur n
  pour c prenant les valeurs entières de 2 **á** $E\left(\frac{n}{2}\right)$ **faire**
    tant que c divise b **faire**
      afficher la valeur de c
      b prend pour valeur $\frac{b}{c}$
    fin du tant que
  fin du pour
Sortie(s) Les diviseurs de n

4. Organigramme

— L'organigramme est utile pour élaborer un programme, il permet de repérer plus facilement les erreurs, d'organiser le travail notamment si le programme est complexe et que le travail est réalisé en équipe. Il est construit, généralement, à partir d'un algorithme ;
— L'organigramme est normalisé, il suit des normes, pour que chacun comprenne la même chose.
  ⋆ La forme ovale indique le début et la fin d'un programme.
  ⋆ Un processus (initialisations, calculs) est représenté à l'aide d'un symbole rectangle.
  ⋆ La vérification d'une condition est représentée par un symbole de diamant.
  ⋆ Le passage d'une étape à l'autre est représenté par une ligne fléchée.
— ...

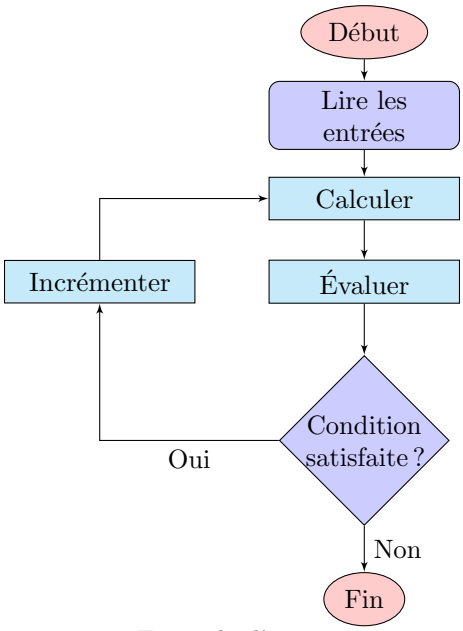

Figure 2. Exemple d'organigramme

L'algorithmique permet ainsi de passer d'un problème à résoudre à un algorithme qui décrit la démarche de résolution du problème. La programmation a alors pour rôle de traduire cet algorithme dans un langage « compréhensible » par l'ordinateur afin qu'il puisse exécuter l'algorithme automatiquement.

5. Programmation

Un algorithme exprime la structure logique d'un programme informatique et de ce fait est indépendant du langage de programmation utilisé. Par contre, la traduction de l'algorithme dans un langage particulier dépend du langage choisi et sa mise en œuvre dépend également de la plateforme d'exécution.
La programmation d'un ordinateur consiste à lui « expliquer » en détail ce qu'il doit faire, en sachant qu'il ne « comprend » pas le langage humain, mais qu'il peut seulement effectuer un traitement automatique sur des séquences de 0 et de 1, appelé langage machine. Un programme n'est rien d'autre qu'une suite d'instructions, encodées en respectant de manière très stricte un ensemble de conventions fixées à l'avance par un langage informatique. La machine décode alors ces instructions en associant à chaque « mot » du langage informatique une action précise. Le programme que nous écrivons dans le langage informatique à l'aide d'un éditeur (une sorte de traitement de texte spécialisé) est appelé programme source (ou code source).
Le seul « langage » que l'ordinateur puisse véritablement « comprendre » est donc très éloigné de ce que nous utilisons nous-mêmes. C'est une longue suite de 0 et de 1 (les « bits », binary digit) traités par groupes de 8 (les « octets », byte), 16, 32 et 64.

LANGAGE *n. m. XIIe siècle. Dérivé de langue. Système de signes, de symboles, élaboré à partir des langues naturelles et constituant un code qui les remplace dans certains cas déterminés (on parle aussi de langage symbolique). Le langage mathématique. Langage logique, fondé sur la logique formelle. Langage informatique. Langage de programmation.*

BIT *n. m. XXe siècle. Mot anglo-américain, contraction de binary digit, « chiffre binaire ». Chacun des deux chiffres, 0 et 1, de la numération binaire. En informatique, le bit est l'unité élémentaire d'information appelée aussi élément binaire.*

OCTET *n. m. XXe siècle. Dérivé savant du latin octo, « huit ». Unité d'information composée de huit bits.*

Définition 5.1 : Bit, Octet

Un bit est un chiffre binaire (0 ou 1). C'est l'unité élémentaire d'information.
Un octet est une unité d'information composée de 8 bits.

Exemple 5.1 : Stockage

Soit une clé USB d'une capacité de stockage de 32 Go (gigaoctet). Le préfixe « giga » est un multiplicateur décimal qui vaut 10^9 mais s'il est bien adapté au calcul décimal, il l'est moins au calcul binaire car il ne correspond pas à une puissance entière de 2.
En effet, la puissance x de 2 telle que $2^x = 10^9$ vaut $x = 9 \cdot \dfrac{\log(10)}{\log(2)} \approx 29.89$. On choisit alors la puissance entière de 2 immédiatement supérieure ($2^{30} = 1\,073\,741\,824$) pour définir le Go : 1 Go = 1 073 741 824 octets.

Pour « parler » à un ordinateur, il nous faudra donc utiliser des systèmes de traduction automatiques, capables de convertir en nombres binaires des suites de caractères formant des mots-clés (anglais en général) qui seront plus significatifs pour nous. Le système de traduction proprement dit s'appellera interpréteur ou bien compilateur, suivant la méthode utilisée pour effectuer la traduction (figure 3).

Définition 5.2 : Compilateur

Un compilateur est un programme informatique qui traduit un langage, le langage source, en un autre, appelé le langage cible.

Définition 5.3 : Interpréteur

Un interpréteur est un outil informatique (logiciel ou matériel) ayant pour tâche d'analyser et d'exécuter un programme écrit dans un langage source.

On appellera langage de programmation un ensemble de mots-clés (choisis arbitrairement) associé à un ensemble de règles très précises indiquant comment on peut assembler ces mots pour former des « phrases » que l'interpréteur ou le compilateur puisse traduire en langage machine (binaire).
Un langage de programmation se distingue du langage mathématique par sa visée opérationnelle (ie. il doit être exécutable par une machine), de sorte qu'un langage de programmation est toujours un compromis entre sa puissance d'expression et sa possibilité d'exécution.

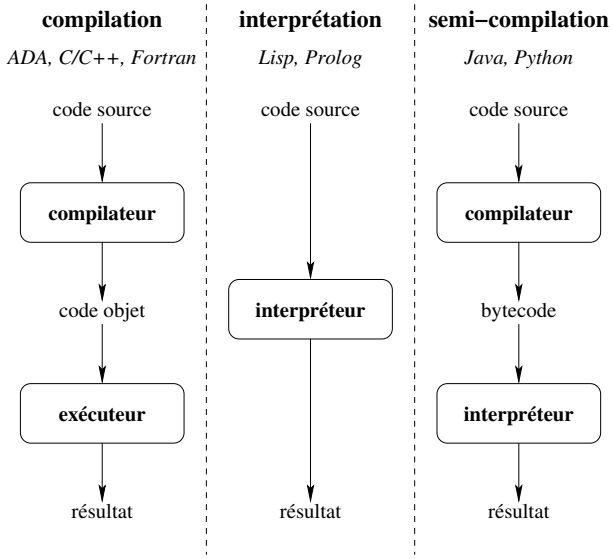

Figure 3. Du code source à son exécution

Définition 5.4 : Langage de programmation

Un langage de programmation est un langage informatique, permettant à un humain d'écrire un code source qui sera analysé par un ordinateur.

Le code source subit ensuite une transformation ou une évaluation dans une forme exploitable par la machine, ce qui permet d'obtenir un programme. Les langages permettent souvent de faire abstraction des mécanismes bas niveaux de la machine, de sorte que le code source représentant une solution puisse être rédigé et compris par un humain.

Définition 5.5 : Programmation

La programmation est l'activité de rédaction du code source d'un programme.

Compilation : La compilation consiste à traduire la totalité du code source en une fois. Le compilateur lit toutes les lignes du programme source et produit une nouvelle suite de codes appelé programme objet (ou code objet). Celui-ci peut désormais être exécuté indépendamment du compilateur et être conservé tel quel dans un fichier (« fichier exécutable »). Les langages C, C++ et FORTRAN sont des exemples de langages compilés.

Interprétation : L'interprétation consiste à traduire chaque ligne du programme source en quelques instructions du langage machine, qui sont ensuite directement exécutées au fur et à mesure (« au fil de l'eau »). Aucun programme objet n'est généré. L'interpréteur doit être utilisé chaque fois que l'on veut faire fonctionner le programme. Les langages LISP et PROLOG sont des exemples de langages interprétés.

L'interprétation est idéale lorsque l'on est en phase d'apprentissage du langage, ou en cours d'expérimentation sur un projet. Avec cette technique, on peut en effet tester immédiatement toute modification apportée au programme source, sans passer par une phase de compilation qui demande toujours du temps. Mais lorsqu'un projet comporte des fonctionnalités complexes qui doivent s'exécuter rapidement, la compilation est préférable : un programme compilé fonctionnera toujours plus vite que son homologue interprété, puisque dans cette technique l'ordinateur n'a plus à (re)traduire chaque instruction en code binaire avant qu'elle puisse être exécutée.

Semi-compilation : Certains langages tentent de combiner les deux techniques afin de tirer le meilleur de chacune. C'est le cas des langages PYTHON et JAVA par exemple. De tels langages commencent par compiler le code source pour produire un code intermédiaire, similaire à un langage machine (mais pour une machine virtuelle), que l'on appelle bytecode, lequel sera ensuite transmis à un interpréteur pour l'exécution finale. Du point de

vue de l'ordinateur, le bytecode est très facile à interpréter en langage machine. Cette interprétation sera donc beaucoup plus rapide que celle d'un code source.

Le fait de disposer en permanence d'un interpréteur permet de tester immédiatement n'importe quel petit morceau de programme. On pourra donc vérifier le bon fonctionnement de chaque composant d'une application au fur et à mesure de sa construction. L'interprétation du bytecode compilé n'est pas aussi rapide que celle d'un véritable code machine, mais elle est satisfaisante pour de très nombreux programmes.

Programme

- Le programme est dit correct lorsqu'il résout le problème posé.
- L'efficacité d'un algorithme se mesure notamment par sa durée de calcul, par sa consommation de mémoire vive, par la précision des résultats obtenus, sa scalabilité (son aptitude à être efficacement parallélisé, évolutif ...), etc.
- Un programme est dit performant, s'il utilise avec parcimonie les ressources dont il dispose, c'est-à-dire le temps CPU, la mémoire vive et la consommation électrique (objet de recherches récentes).

5.1 Langage de haut niveau et de bas niveau

Le langage de haut niveau est un langage éloigné du binaire (langage machine) et permet généralement de développer de façon plus souple et rapide.

Le langage de bas niveau est plus proche du langage machine, il demande en général un peu plus d'efforts et permet un contrôle plus simple des instructions.

— Programmes de haut niveau : Python, Matlab, Scilab, Java ...
— Programmes de bas niveau : C, C++ ...
— Programme machine : Assembleur.

5.2 Objectifs comportementaux

On cherche à développer trois « qualités » comportementales chez l'informaticien débutant : la rigueur, la persévérance et l'autonomie.

Rigueur : Un ordinateur est une machine qui exécute vite et bien les instructions qu'on lui a « apprises ». Mais il ne sait pas interpréter autre chose : même mineure, une erreur provoque le dysfonctionnement de la machine.

Exemple 5.2 : Erreur de syntaxe en langage C

Pour un « ; » oublié dans un programme C, le code source n'est pas compilable et le compilateur nous avertit par ce type de message :

```
fichier.c: In function 'main':
fichier.c:7: error: syntax error before "printf"
```

Même si un humain peut transiger sur le « ; », la machine ne le peut pas : l'ordinateur ne se contente pas de l'« à peu près ». La respect des consignes, la précision et l'exactitude sont donc de rigueur en informatique !

Persévérance : Face à l'intransigeance de la machine, le débutant est confronté à ses nombreuses erreurs (les siennes, pas celles de la machine !) et sa tendance naturelle est de passer à autre chose. Mais le papillonnage (ou *zapping*) est une très mauvaise stratégie en informatique : pour s'exécuter correctement, un programme doit être finalisé. L'informatique nécessite d'« aller au bout des choses ».

Autonomie : Programmer soi-même les algorithmes qu'on a définis est sans doute le meilleur moyen pour mieux assimiler les principales structures algorithmiques et pour mieux comprendre ses erreurs en se confrontant à l'intransigeante impartialité de l'ordinateur, véritable « juge de paix » des informaticiens. Par ailleurs, l'évolution continuelle et soutenue des langages de programmation et des ordinateurs nécessitent de se tenir à jour régulièrement et seule l'autoformation systématique permet de « ne pas perdre pied ».

Pratique personnelle et autoformation constituent ainsi deux piliers de l'autonomisation nécessaire de l'apprenti informaticien.

6. Exercices

Exercice 1 :

Écrire un algorithme permettant de faire la somme des N premiers entiers. L'algorithme demandé prend en entrée un nombre entier N, et renvoie un autre nombre entier (la somme demandée).

Exercice 2 :

Ecrire un algorithme pour calculer le diamètre, le périmètre et la surface d'un cercle à partir de son rayon.

Exercice 3 :

Écrire un algorithme qui échange la valeur de deux variables.
Exemple, si $a = 2$ et $b = 5$, le programme donnera $a = 5$ et $b = 2$.

Exercice 4 :

Écrire un algorithme qui demande un nombre à l'utilisateur, puis qui calcule et affiche le carré de ce nombre.

Exercice 5 :

Écrire un algorithme qui lit le prix HT d'un article, le nombre d'articles et le taux de TVA, et qui fournit le prix total TTC correspondant.

Exercice 6 :

Écrire un algorithme qui demande un nombre entier de départ, et qui calcule sa factorielle.

Remarque 6.1

La factorielle d'un nombre entier n est : $n! = 1 * 2 * 3 * \cdots * n$. Ex. la factorielle de 6, notée 6!, vaut $1 * 2 * 3 * 4 * 5 * 6 = 720$.