INITIATION AUX LANGAGES

INFORMATIQUES ET A LA PROGRAMMATION

Enseignant: KWETTE NOUMSI Loïc Bastien

ISTIC

December 4, 2018

AVANT-PROPOS

Ce cours permettra aux étudiants d'aborder la programmation et l'écriture des algorithmes pour résoudre un problème en automatisant l’exécution de ses tâches sur un ordinateur via un langage de programmation.

SOMMAIRE

CHAPITRE I - INTRODUCTION GENERALE

1. Contexte scientifique
    1.1. Informatique

Le terme informatique est un néologisme proposé en 1962 par Philippe Dreyfus pour caractériser le traitement automatique de l'information : il est construit sur la contraction de l'expression « information automatique ». Ce terme a été accepté par l'Académie française en avril 1966, et l'informatique devint alors officiellement la science du traitement automatique de l'information, où l'information est considérée comme le support des connaissances humaines et des communications dans les domaines techniques, économiques et sociaux.

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.

Définition : Informatique

L'informatique est la science du traitement automatique de l'information. L'informatique traite de deux aspects complémentaires : les programmes immatériels (logiciel, software) qui d´écrivent un traitement à réaliser et les machines (matériel, hardware) qui exécutent ce traitement. Le matériel est donc l'ensemble des éléments physiques (microprocesseur, mémoire, écran, clavier, disques durs. . .) utilisés pour traiter les données. Dans ce contexte, l'ordinateur est un terme générique qui désigne un équipement informatique permettant de traiter des informations selon des séquences d'instructions (les programmes) qui constituent le logiciel. Le terme ordinateur a été proposé par le philologue Jacques Perret en avril 1955 en réponse à une demande d'IBM France qui estimait le mot « calculateur » (computer) bien trop restrictif en regard des possibilités de ces machines.

Définition : Matériel

Le matériel informatique est un ensemble de dispositifs physiques utilisés pour traiter automatiquement des informations.

Définition : Logiciel

Le logiciel est un ensemble structuré d'instructions décrivant un traitement d'informations à faire réaliser par un matériel informatique.

Un ordinateur n'est rien d'autre qu'une machine effectuant des opérations simples sur des séquences de signaux électriques, lesquels sont conditionnés de manière à ne pouvoir prendre que deux états seulement (par exemple un potentiel électrique maximum ou minimum). Ces s´séquences de signaux obéissent à une logique binaire du type « tout ou rien » et peuvent donc être considérés conventionnellement comme des suites de nombres ne prenant jamais que les deux valeurs 0 et 1 : un ordinateur est donc incapable de traiter autre chose que des nombres binaires. Toute information d'un autre type doit être convertie, ou codée, en format binaire. C'est vrai non seulement pour les données que l'on souhaite traiter (les nombres, les textes, les images, les sons, les vidéos, etc.), mais aussi pour les programmes, c'est-`a-dire les séquences L'architecture matérielle d'un ordinateur repose sur le modèle de Von Neumann (figure 1.2) qui distingue classiquement 4 parties (figure 1.3) :

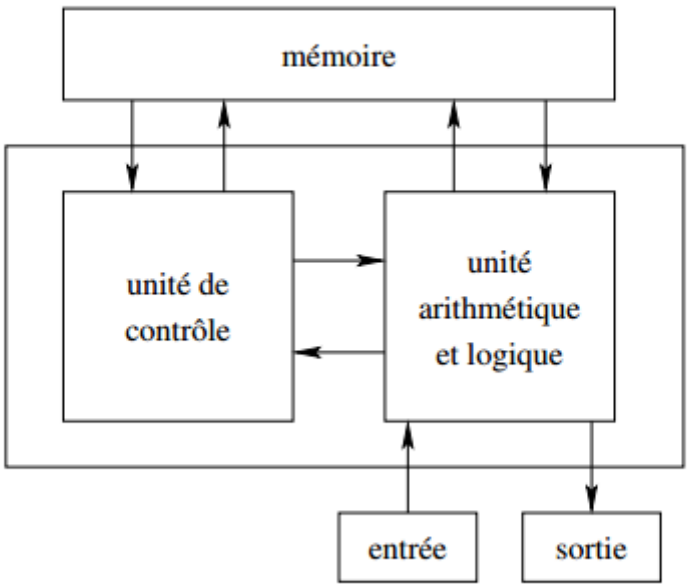
1. L'unité arithmétique et logique, ou unité de traitement, effectue les opérations de base.
2. L'unité de contrôle séquence les opérations.
3. La mémoire contient à la fois les données et le programme qui dira à l'unité de contrôle quels calculs faire sur ces données. La mémoire se divise entre mémoire 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 microprocesseur (la « puce »), unité centrale de l'ordinateur. Dans ce cours, nous ne nous intéresserons qu'aux aspects logiciels de l'informatique.

Fig. 1.2 : JOHN VON NEUMANN (1903-1957)

Mathématicien américain d'origine hongroise : il a donné son nom à l'architecture de von Neumann utilisée dans la quasi totalité des ordinateurs modernes (voir figure 1.3 ci-dessous).

Fig. 1.3 : ARCHITECTURE DE VON NEUMANN



1.2. **Algorithmique**

Exemple 1.1 : **Mode d'emploi d'un télécopieur**
Extrait du mode d'emploi d'un télécopieur concernant l'envoi d'un document.

1. *Insérez le document dans le chargeur automatique ;*
2. *Composez le numéro de fax du destinataire à l'aide du pavé numérique ;*
3. *Enfoncez la touche envoi pour lancer l'émission*.

Ce mode d'emploi précise comment envoyer un fax. Il est composé d'une suite ordonnée d'instructions (insérez. . ., composez. . ., enfoncez. . .) qui manipulent des données (document, chargeur *automatique, numéro de fax, pavé numérique, touche envoi) pour réaliser la tâche désirée (envoi d'un document).*

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

Exemple 1.2 : Trouver son chemin
Extrait d'un dialogue entre un touriste égaré et un autochtone.

- Pourriez-vous m'indiquer le chemin de la gare, s'il vous plait ?
- Oui bien sûr : vous allez tout droit jusqu'au prochain carrefour, vous prenez à gauche au carrefour et ensuite la troisième à droite, et vous verrez la gare juste en face de vous ;
- Merci.

Dans ce dialogue, la réponse de l'autochtone est la description d'une suite ordonnée d'instructions (allez tout droit, prenez à gauche, prenez la troisième à droite) qui manipulent des données (carrefour, rues) pour réaliser la tâche désirée (aller à la gare). Ici encore, chacun a déjà été confronté à ce genre de situation et donc, consciemment ou non, a déjà construit un algorithme dans sa tête (c-à-d définir la suite d'instructions pour réaliser une tâche). Mais quand on définit un algorithme, celui-ci ne doit contenir que des

instructions compréhensibles par celui qui devra l'exécuter (des humains dans les 2 exemples précédents).

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

Définition : 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 : Validité d'un algorithme

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

Si l'on reprend l'exemple 1.2 de l'algorithme de recherche du chemin de la gare, l'étude de sa validité consiste à s'assurer qu'on arrive effectivement à la gare en exécutant scrupuleusement les instructions dans l'ordre annoncé.

Définition : Robustesse d'un algorithme

La robustesse d'un algorithme est son aptitude à se protéger de conditions anormales d'utilisation. Dans l'exemple 1.2, la question de la robustesse de l'algorithme se pose par exemple si le chemin.

Dans l'exemple 1.2, la question de la robustesse de l'algorithme se pose par exemple si le chemin proposé a été pensé pour un piéton, alors que le « touriste égaré » est en voiture et que la « troisième à droite » est en sens interdit.

Définition: 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. L'algorithme de recherche du chemin de la gare est-il réutilisable tel quel pour se rendre à la mairie ? A priori non, sauf si la mairie est juste à côté de la gare.

Définition : Complexite 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. Si le «touriste égaré» est un piéton, la complexité de l'algorithme de recherche de chemin peut se compter en nombre de pas pour arriver à la gare.

Définition: Efficacité d'un algorithme

L'efficacité d'un algorithme est son aptitude à utiliser de manière optimale les ressources du matériel qui l'exécute. N'existerait-il pas un raccourci piétonnier pour arriver plus vite à la gare ?

L'algorithmique permet ainsi de passer d'un problème à résoudre à un algorithme qui d´écrit 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 (figure 1.5).

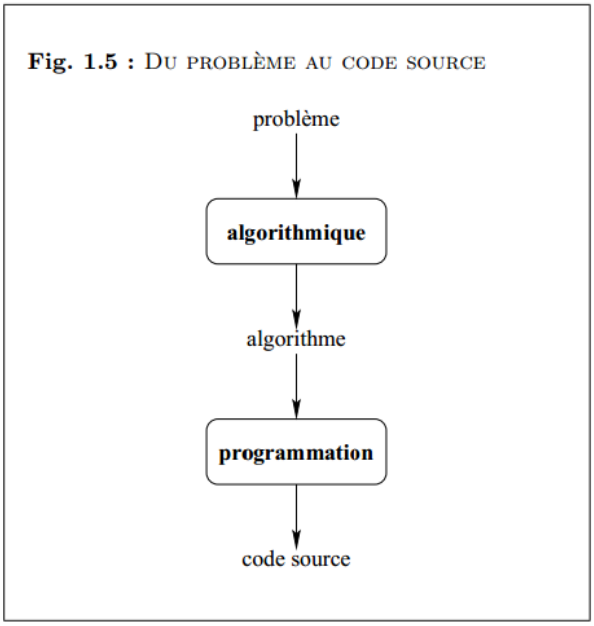


Fig. 1.5 : DU PROBLÈME AU CODE SOURCE

1.3. 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.

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 spécialise) 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, ou même 64.

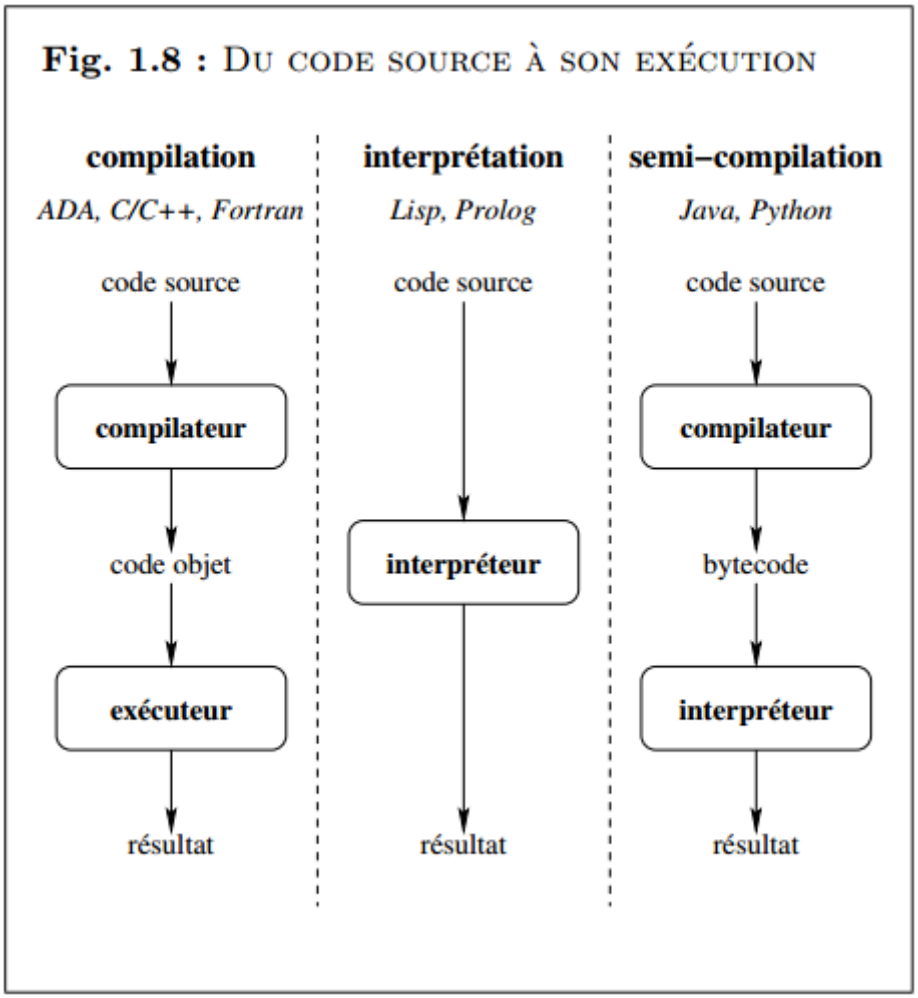
Définition: Bit

Un bit est un chiffre binaire (0 ou 1). C'est l'unité élémentaire d'information.

Définition: Octet

Un octet est une unité d'information composée de 8 bits. 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 1.8).



Définition : Compilateur

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

Définition: 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

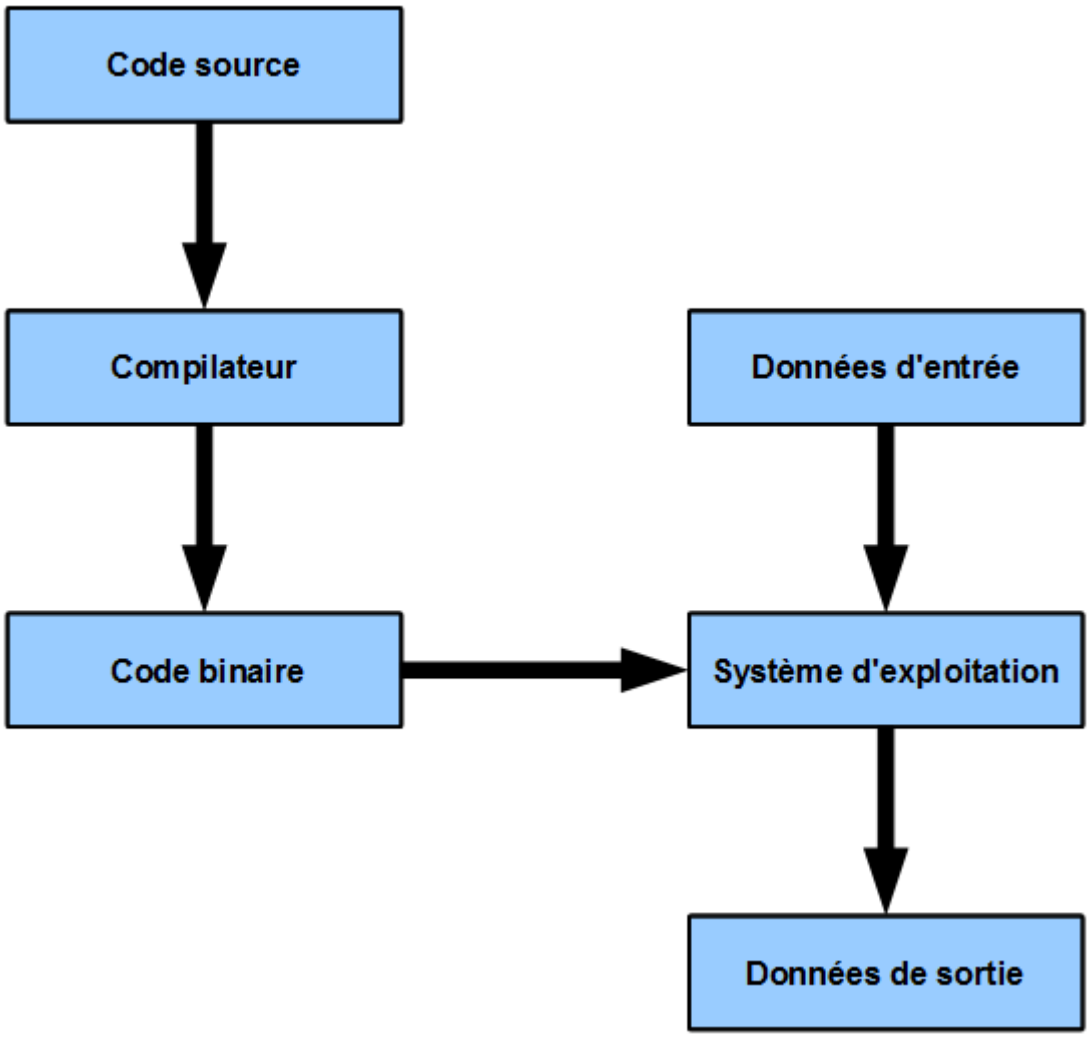
l'interpréteur ou le compilateur puisse traduire en langage machine (binaire). Un langage de programmation se distingue du langage mathématique par sa visée opérationnelle (c-à-d il doit être exécutable par une machine), de sorte qu'un langage de programmation est toujours un compromis entre sa puissance d'expression et sa possibilité d'exécution.

Définition: 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 rédige et compris par un humain.

Définition: Programmation

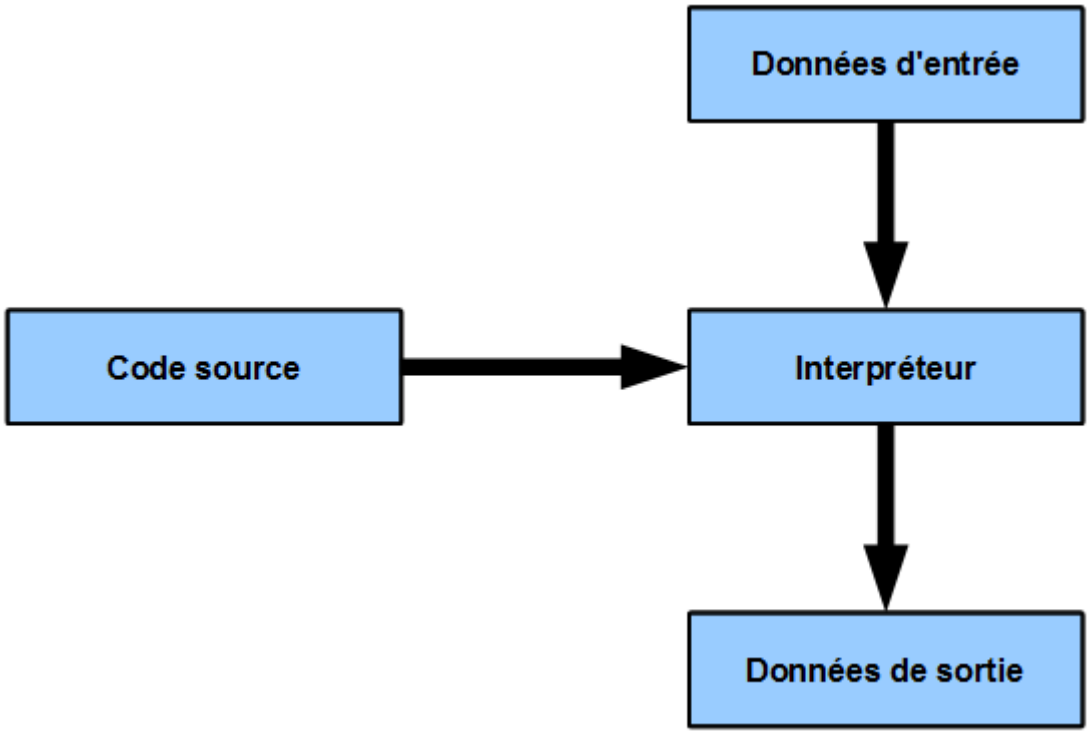
La programmation est l'activité de rédaction du code source d'un programme.
Compilation : La compilation consiste à traduire la totalité du code source en une fois. Le compilateur lit toutes les lignes du programme source et produit une nouvelle suite de codes appelé ***programme objet*** (ou ***code objet***). Celui-ci peut désormais être exécute indépendamment du compilateur et être conservé tel quel dans un fichier (« fichier exécutable »). Les langages ***Ada***, ***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.

Un *langage informatique* se dit *interprété* lorsque les instructions qui le composent sont décodées les unes après les autres et exécutées aussitôt. Son fonctionnement est le suivant:

1. Lire l'instruction ;
2. Exécuter l'instruction ;
3. Passer à l'instruction suivante et recommencer.



C'est bel et bien instruction par instruction que le programme est exécuté. En réalité, il n'y a aucune traduction du programme depuis le langage informatique en langage machine interprété par l'ordinateur. L'*interpréteur* fait une *interprétation* du langage informatique pas à pas. Cela donne l'avantage d'une exécution immédiate, mais aussi d'une certaine lenteur. De plus, de nombreuses erreurs de syntaxe ne sont identifiées que lorsque le programme est exécuté, et non plus tôt, d'où la difficulté d'identifier certains bugs et erreurs.

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.

Principales différences

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.

Semi-compilation ou intermédiation : Certains langages tentent de combiner les deux techniques afin de retirer 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.

Voici une liste non exhaustive de langages informatiques existants :

Langage	Domaine d'application principal	Compilé/interprété
ADA	Le temps réél	Langage compilé
BASIC	Programmation basique à but éducatif	Langage interprété
C	Programmation système	Langage compilé
C++	Programmation système objet	Langage compilé
Cobol	Gestion	Langage compilé
Fortran	Calcul	Langage compilé
Java	Programmation orientée internet	Langage intermédiaire
MATLAB	Calcul mathématique	Langage interprété
Mathematica	Calcul mathématique	Langage interprété
LISP	Intelligence artificielle	Langage intermédiaire
Pascal	Enseignement	Langage compilé
PHP	Développement de sites web dynamiques	Langage interprété
Prolog	Intelligence artificielle	Langage interprété

Dans ce cours, nous choisirons d'exprimer directement les algorithmes dans un langage informatique opérationnel : le langage *PHP*, sans passer par un langage algorithmique intermédiaire.

2. Objectifs du cours

L'objectif principal de ce cours est l'acquisition des notions fondamentales de l'algorithmique. Plus précisément, nous étudierons successivement :

i. Les instructions de base permettant de d´écrire les algorithmes : affectation, tests, boucles ;
ii. Les procédures et les fonctions qui permettent de structurer et de réutiliser les algorithmes ; on parlera alors d'encapsulation, de préconditions, de portée des variables, de passage de paramètres, d'appels de fonctions, de récursivité et de jeux de tests ;
iii. Les structures de données linéaires : tableaux, listes, piles, files, qui améliorent la structuration des données manipulées par les algorithmes.

A cette occasion, on évaluera la complexité et l'efficacité de certains algorithmes utilisant ces structures linéaires.

Ces différentes notions seront mises en œuvre à travers l'utilisation du langage ***PHP***.

CHAPITRE II – PROGRAMMATION EN PHP