

Algorithmique et programmation en Java

Vincent Granet

Maître de conférences à Polytech'Sophia
de l'université Nice-Sophia-Antipolis

5^e édition

DUNOD

Toutes les marques citées dans cet ouvrage
sont des marques déposées par leurs propriétaires respectifs.



Pour aller plus loin et mettre toutes les chances de votre côté,
des ressources complémentaires sont disponibles sur le site
www.dunod.com.

Connectez-vous à la page de l'ouvrage (grâce aux menus déroulants, ou en saisissant
le titre, l'auteur ou l'ISBN dans le champ de recherche de la page d'accueil).
Sur la page de l'ouvrage, sous la couverture, cliquez sur le lien « LES + EN LIGNE ».

© Dunod, 2018

11 rue Paul Bert, 92240 Malakoff
www.dunod.com

ISBN 978-2-10-078667-1

Le Code de la propriété intellectuelle n'autorisant, aux termes de l'article L. 122-5, 2° et 3° a), d'une part, que les « copies ou reproductions strictement réservées à l'usage privé du copiste et non destinées à une utilisation collective » et, d'autre part, que les analyses et les courtes citations dans un but d'exemple et d'illustration, « toute représentation ou reproduction intégrale ou partielle faite sans le consentement de l'auteur ou de ses ayants droit ou ayants cause est illicite » (art. L. 122-4).

Cette représentation ou reproduction, par quelque procédé que ce soit, constituerait donc une contrefaçon sanctionnée par les articles L. 335-2 et suivants du Code de la propriété intellectuelle.

à Maud

Table des matières

AVANT-PROPOS	XVII
CHAPITRE 1 • INTRODUCTION	1
1.1 Environnement matériel	1
1.2 Environnement logiciel	4
1.3 Les langages de programmation	6
1.4 Construction des programmes	12
1.5 Démonstration de validité	13
CHAPITRE 2 • ACTIONS ÉLÉMENTAIRES	17
2.1 Lecture d'une donnée	17
2.2 Exécution d'une routine prédéfinie	18
2.3 Écriture d'un résultat	19
2.4 Affectation d'un nom à un objet	19
2.5 Déclaration d'un nom	20
2.5.1 Déclaration de constantes	20
2.5.2 Déclaration de variables	21
2.6 Règles de déduction	21
2.6.1 L'affectation	21
2.6.2 L'appel de routine	22
2.7 Le programme sinus écrit en Java	22
2.8 Exercices	24
CHAPITRE 3 • TYPES ÉLÉMENTAIRES	25
3.1 Le type entier	26
3.2 Le type réel	27

3.3	Le type booléen	30
3.4	Le type caractère	31
3.5	Constructeurs de types simples	33
3.5.1	Les types énumérés	33
3.5.2	Les types intervalles	34
3.6	Exercices	34
CHAPITRE 4 • EXPRESSIONS		37
4.1	Évaluation	38
4.1.1	Composition du même opérateur plusieurs fois	38
4.1.2	Composition de plusieurs opérateurs différents	38
4.1.3	Parenthésage des parties d'une expression	39
4.2	Type d'une expression	39
4.3	Conversions de type	40
4.4	Un exemple	40
4.5	Exercices	43
CHAPITRE 5 • ÉNONCÉS STRUCTURÉS		45
5.1	Énoncé composé	45
5.2	Énoncés conditionnels	46
5.2.1	Énoncé choix	46
5.2.2	Énoncé si	48
5.3	Résolution d'une équation du second degré	49
5.4	Exercices	53
CHAPITRE 6 • ROUTINES		55
6.1	Intérêt	55
6.2	Déclaration d'une routine	56
6.3	Appel d'une routine	57
6.4	Transmission des paramètres	58
6.4.1	Transmission par valeur	59
6.4.2	Transmission par résultat	59
6.5	Retour d'une routine	59
6.6	Localisation	60
6.7	Règles de déduction	62
6.8	Exemples	63
6.8.1	Équation du second degré	63
6.8.2	Date du lendemain	64
6.9	Exercices	67

CHAPITRE 7 • PROGRAMMATION PAR OBJETS	69
7.1 Objets et classes	69
7.1.1 Création des objets	70
7.1.2 Destruction des objets	71
7.1.3 Accès aux attributs	71
7.1.4 Attributs de classe partagés	72
7.1.5 Les classes en Java	72
7.2 Les méthodes	73
7.2.1 Accès aux méthodes	74
7.2.2 Constructeurs	74
7.2.3 Constructeurs en Java	75
7.2.4 Les méthodes en Java	75
7.3 Assertions sur les classes	77
7.4 Exemples	78
7.4.1 Équation du second degré	78
7.4.2 Date du lendemain	81
7.5 Exercices	84
CHAPITRE 8 • ÉNONCÉS ITÉRATIFS	87
8.1 Forme générale	87
8.2 L'énoncé tant que	88
8.3 L'énoncé répéter	89
8.4 Finitude	90
8.5 Exemples	90
8.5.1 Factorielle	90
8.5.2 Minimum et maximum	91
8.5.3 Division entière	92
8.5.4 Plus grand commun diviseur	93
8.5.5 Multiplication	93
8.5.6 Puissance	94
8.6 Exercices	95
CHAPITRE 9 • LES TABLEAUX	97
9.1 Déclaration d'un tableau	97
9.2 Dénotation d'un composant de tableau	98
9.3 Modification sélective	99
9.4 Opérations sur les tableaux	99
9.5 Les tableaux en Java	99
9.6 Un exemple	101
9.7 Les chaînes de caractères	103
9.8 Exercices	105

CHAPITRE 10 • L'ÉNONCÉ ITÉRATIF POUR	107
10.1 Forme générale	107
10.2 Forme restreinte	108
10.3 Les énoncés pour de Java	108
10.4 Exemples	110
10.4.1 Le schéma de HORNER	110
10.4.2 Exemple en Java : nombres binaires	111
10.4.3 Un tri interne simple	112
10.4.4 Confrontation de modèle	114
10.5 Complexité des algorithmes	117
10.6 Exercices	119
CHAPITRE 11 • LES TABLEAUX À PLUSIEURS DIMENSIONS	123
11.1 Déclaration	123
11.2 Dénotation d'un composant de tableau	124
11.3 Modification sélective	124
11.4 Opérations	124
11.5 Tableaux à plusieurs dimensions en Java	125
11.6 Exemples	125
11.6.1 Initialisation d'une matrice	125
11.6.2 Ligne de somme maximale	126
11.6.3 Matrice symétrique	127
11.6.4 Demi-matrice carrée	128
11.6.5 Produit de matrices	129
11.6.6 Carré magique	130
11.7 Exercices	132
CHAPITRE 12 • HÉRITAGE	137
12.1 Classes héritières	137
12.2 Redéfinition de méthodes	140
12.3 Recherche d'un attribut ou d'une méthode	141
12.4 Polymorphisme et liaison dynamique	142
12.5 Classes abstraites	144
12.6 Héritage simple et multiple	144
12.7 Héritage et assertions	145
12.7.1 Assertions sur les classes héritières	145
12.7.2 Assertions sur les méthodes	145
12.8 Relation d'héritage ou de clientèle	146
12.9 L'héritage en Java	147
12.10 Exercices	150

CHAPITRE 13 • FONCTIONS ANONYMES	153
13.1 Paramètres fonctions	154
13.1.1 Fonctions anonymes en Java	155
13.1.2 Filtrage, foreach et map	157
13.1.3 Continuation	159
13.2 Fonctions anonymes en résultat	161
13.2.1 Composition	161
13.2.2 Curryfication	162
13.3 Fermeture	164
13.3.1 Fermeture en Java	164
13.3.2 Lambda récursives	166
13.4 Exercices	167
CHAPITRE 14 • LES EXCEPTIONS	169
14.1 Émission d'une exception	169
14.2 Traitement d'une exception	170
14.3 Le mécanisme d'exception de Java	171
14.3.1 Traitement d'une exception	171
14.3.2 Émission d'une exception	172
14.4 Exercices	173
CHAPITRE 15 • LES FICHIERS SÉQUENTIELS	175
15.1 Déclaration de type	176
15.2 Notation	176
15.3 Manipulation des fichiers	177
15.3.1 Écriture	177
15.3.2 Lecture	178
15.4 Les fichiers de Java	179
15.4.1 Fichiers d'octets	179
15.4.2 Fichiers d'objets élémentaires	181
15.4.3 Fichiers d'objets structurés	185
15.5 Les fichiers de texte	186
15.6 Les fichiers de texte en Java	186
15.7 Automate et reconnaissance de mots	188
15.8 Exercices	193
CHAPITRE 16 • RÉCURSIVITÉ	195
16.1 Récursivité des actions	196
16.1.1 Définition	196
16.1.2 Finitude	196
16.1.3 Écriture récursive des routines	196
16.1.4 La pile d'évaluation	199
16.1.5 Quand ne pas utiliser la récursivité ?	200

16.1.6	Réversibilité directe et croisée	202
16.1.7	Zéro d'une fonction	204
16.2	Réversibilité des objets	205
16.3	Exercices	208
CHAPITRE 17 • STRUCTURES DE DONNÉES		211
17.1	Définition d'un type abstrait	212
17.2	L'implémentation d'un type abstrait	214
17.3	Utilisation du type abstrait	216
CHAPITRE 18 • STRUCTURES LINÉAIRES		219
18.1	Les listes	219
18.1.1	Définition abstraite	220
18.1.2	L'implémentation en Java	221
18.1.3	Énumération	233
18.2	Les piles	237
18.2.1	Définition abstraite	237
18.2.2	L'implémentation en Java	238
18.3	Les files	241
18.3.1	Définition abstraite	242
18.3.2	L'implémentation en Java	242
18.4	Les dèques	243
18.4.1	Définition abstraite	244
18.4.2	L'implémentation en Java	244
18.5	Exemple d'utilisation d'une Pile	245
18.6	Exercices	247
CHAPITRE 19 • GRAPHES		251
19.1	Terminologie	252
19.2	Définition abstraite d'un graphe	253
19.3	L'implémentation en Java	254
19.3.1	Matrice d'adjacence	256
19.3.2	Listes d'adjacence	258
19.4	Parcours d'un graphe	260
19.4.1	Parcours en profondeur	260
19.4.2	Parcours en largeur	261
19.4.3	Programmation en Java des parcours de graphe	262
19.5	Exemple	263
19.6	Exercices	263

CHAPITRE 20 • STRUCTURES ARBORESCENTES	265
20.1 Terminologie	266
20.2 Les arbres	267
20.2.1 Définition abstraite	268
20.2.2 L'implémentation en Java	269
20.2.3 Algorithmes de parcours d'un arbre	272
20.3 Arbre binaire	273
20.3.1 Définition abstraite	274
20.3.2 L'implémentation en Java	275
20.3.3 Parcours d'un arbre binaire	278
20.4 Représentation binaire des arbres généraux	280
20.5 Exercices	281
CHAPITRE 21 • TABLES	285
21.1 Définition abstraite	286
21.1.1 Ensembles	286
21.1.2 Description fonctionnelle	286
21.1.3 Description axiomatique	286
21.2 Représentation des éléments en Java	286
21.3 Représentation par une liste	288
21.3.1 Liste non ordonnée	288
21.3.2 Liste ordonnée	290
21.3.3 Recherche dichotomique	292
21.4 Représentation par un arbre ordonné	294
21.4.1 Recherche d'un élément	295
21.4.2 Ajout d'un élément	295
21.4.3 Suppression d'un élément	296
21.5 Les arbres AVL	298
21.5.1 Rotations	299
21.5.2 Mise en œuvre	302
21.6 Arbres 2-3-4 et bicolores	306
21.6.1 Les arbres 2-3-4	306
21.6.2 Mise en œuvre en Java	309
21.6.3 Les arbres bicolores	311
21.6.4 Mise en œuvre en Java	316
21.7 Tables d'adressage dispersé	321
21.7.1 Le problème des collisions	323
21.7.2 Choix de la fonction d'adressage	323
21.7.3 Résolution des collisions	325
21.8 Exercices	329

CHAPITRE 22 • FILES AVEC PRIORITÉ	333
22.1 Définition abstraite	333
22.2 Représentation avec une liste	334
22.3 Représentation avec un tas	334
22.3.1 Premier	335
22.3.2 Ajouter	336
22.3.3 Supprimer	337
22.3.4 L'implémentation en Java	338
22.4 Exercices	341
CHAPITRE 23 • ALGORITHMES DE TRI	343
23.1 Introduction	343
23.2 Tris internes	344
23.2.1 L'implantation en Java	345
23.2.2 Méthodes par sélection	345
23.2.3 Méthodes par insertion	350
23.2.4 Tri par échanges	355
23.2.5 Comparaisons des méthodes	359
23.2.6 Tri sans comparaison de clés	361
23.3 Tris externes	363
23.4 Exercices	366
CHAPITRE 24 • ALGORITHMES SUR LES GRAPHS	369
24.1 Composantes connexes	369
24.2 Fermeture transitive	371
24.3 Plus court chemin	374
24.3.1 Algorithme de Dijkstra	374
24.3.2 Algorithme de Bellman-Ford	378
24.3.3 Algorithme A*	379
24.3.4 Algorithme IDA*	381
24.4 Tri topologique	383
24.4.1 L'implémentation en Java	384
24.4.2 Existence de cycle dans un graphe	386
24.4.3 Tri topologique inverse	386
24.4.4 L'implémentation en Java	386
24.5 Exercices	387
CHAPITRE 25 • ALGORITHMES DE RÉTRO-PARCOURS	391
25.1 Écriture récursive	391
25.2 Le problème des huit reines	393
25.3 Écriture itérative	395
25.4 Problème des sous-suites	396
25.5 Jeux de stratégie	398

25.5.1	Stratégie <i>MinMax</i>	398
25.5.2	Coupure α - β	401
25.5.3	Profondeur de l'arbre de jeu	404
25.6	Exercices	405
CHAPITRE 26 • INTERFACES GRAPHIQUES		409
26.1	Systèmes interactifs	409
26.2	Conception d'une application interactive	411
26.3	Environnements graphiques	413
26.3.1	Système de fenêtrage	414
26.3.2	Caractéristiques des fenêtres	416
26.3.3	Boîtes à outils	418
26.3.4	Générateurs	420
26.4	Interfaces graphiques en Java	420
26.4.1	Une simple fenêtre	421
26.4.2	Convertisseur Celsius-Fahrenheit	423
26.4.3	Un composant graphique pour visualiser des couleurs	427
26.4.4	Applets	430
26.5	Exercices	432
BIBLIOGRAPHIE		435
INDEX		439

Avant-propos

Pour cette cinquième édition d'*Algorithmique et Programmation en Java*, l'ensemble de l'ouvrage a été révisé. Il a été corrigé et actualisé pour tenir compte de l'évolution des technologies depuis quatre ans. De nouveaux algorithmes sont présentés, et de nouveaux exercices sont proposés. Des diagrammes de classes au format UML ont été introduits pour donner au lecteur une vision plus claire des relations entre les classes qui implémentent les types abstraits présentés dans la seconde partie de l'ouvrage.

Contrairement à la version 8 qui avait introduit le paradigme fonctionnel, la version 9¹ de JAVA n'apporte pas de grands bouleversements au niveau du langage lui-même. La grande nouveauté est l'introduction de la notion de *module* et le découpage en modules de toute l'API. Cette notion définit un nouveau niveau d'encapsulation et sera évoquée au chapitre 12. Toutefois, dans le cadre de cet ouvrage, elle reste marginale et sera présentée succinctement.

L'informatique est une science mais aussi une technologie et un ensemble d'outils. Ces trois composantes ne doivent pas être confondues, et l'enseignement de l'informatique ne doit pas être réduit au seul apprentissage des logiciels. Ainsi, l'activité de programmation ne doit pas se confondre avec l'étude d'un langage de programmation particulier. Même si l'importance de ce dernier ne doit pas être sous-estimée, il demeure un simple outil de mise en œuvre de concepts algorithmiques et de programmation généraux et fondamentaux. L'objectif de cet ouvrage est d'enseigner au lecteur des méthodes et des outils de construction de programmes informatiques valides et fiables.

L'étude de l'algorithmique et de la programmation est un des piliers fondamentaux sur lesquels repose l'enseignement de l'informatique. Ce livre s'adresse principalement aux étudiants des cycles informatiques et élèves ingénieurs informaticiens, mais aussi à tous ceux qui ne se destinent pas à la carrière informatique mais qui seront certainement confrontés au développement de programmes informatiques au cours de leur scolarité ou dans leur vie professionnelle.

1. À noter qu'en attendant la version stable numéro 11 du langage, une version 10 intermédiaire propose l'inférence de type des variables locales.

Les seize premiers chapitres présentent les concepts de base de la programmation *impérative* en s'appuyant sur une *méthodologie objet*. Le chapitre 13 est quant à lui dédié à la *programmation fonctionnelle*. Tous mettent l'accent sur la notion de preuve des programmes grâce à la notion d'*affirmations* (antécédent, conséquent, invariant) dont la vérification formelle garantit la validité de programmes. Ils introduisent aussi la notion de *complexité* des algorithmes pour évaluer leur performance.

Les onze derniers chapitres étudient en détail les structures de données abstraites classiques (liste, graphe, arbre...) et de nombreux algorithmes fondamentaux (recherche, tri, jeux et stratégie...) que tout étudiant en informatique doit connaître et maîtriser. D'autre part, un chapitre est consacré aux interfaces graphiques et à leur programmation avec SWING.

La présentation des concepts de programmation cherche à être indépendante, autant que faire se peut, d'un langage de programmation particulier. Les algorithmes seront décrits dans une notation algorithmique épurée. Pour des raisons pédagogiques, il a toutefois bien fallu faire le choix d'un langage pour programmer les structures de données et les algorithmes présentés dans cet ouvrage. Ce choix s'est porté sur le langage à objets JAVA, non pas par effet de mode, mais plutôt pour les qualités de ce langage, malgré quelques défauts. Ses qualités sont en particulier sa relative simplicité pour la mise en œuvre des algorithmes, un large champ d'application et sa grande disponibilité sur des environnements variés. Ce dernier point est en effet important ; le lecteur doit pouvoir disposer facilement d'un compilateur et d'un interprète afin de résoudre les exercices proposés à la fin des chapitres. Enfin, JAVA est de plus en plus utilisé comme langage d'apprentissage de la programmation dans les universités. Pour les défauts, on peut par exemple regretter l'absence de l'héritage multiple, et la présence de constructions archaïques héritées du langage C. Ce livre n'est toutefois pas un ouvrage d'apprentissage du langage JAVA. Même si les éléments du langage nécessaires à la mise en œuvre des notions d'algorithmique et de programmation ont été introduits, ce livre n'enseignera pas au lecteur les finesses et les arcanes de JAVA, pas plus qu'il ne décrira les nombreuses classes de l'API. Le lecteur intéressé pourra se reporter aux très nombreux ouvrages qui décrivent le langage en détail, comme par exemple [GR15, Del17].

Les corrigés de la plupart des exercices, ainsi que des applets qui proposent une vision graphique de certains programmes présentés dans l'ouvrage sont accessibles sur le site web de l'auteur à l'adresse :

<http://www.polytech.unice.fr/~vg/index-algoprog.html>

Cet ouvrage doit beaucoup à de nombreuses personnes. Tout d'abord, aux auteurs des algorithmes et des techniques de programmation qu'il présente. Il n'est pas possible de les citer tous ici, mais les références à leurs principaux textes sont dans la bibliographie. À Olivier Lecarme et Jean-Claude Boussard, mes professeurs à l'université de Nice qui m'ont enseigné cette discipline au début des années 1980. Je tiens tout particulièrement à remercier ce dernier qui fut le tout premier lecteur attentif de cet ouvrage alors qu'il n'était encore qu'une ébauche, et qui m'a encouragé à poursuivre sa rédaction. À mes collègues et mes étudiants qui m'ont aidé et soutenu dans cette tâche ardue qu'est la rédaction d'un livre.

Enfin, je remercie toute l'équipe Dunod, et particulièrement Jean-Luc Blanc, pour leur aide précieuse et leurs conseils avisés qu'ils m'ont apportés pour la publication des cinq éditions de cet ouvrage.

Sophia Antipolis, avril 2018.

Chapitre 1

Introduction

Les informaticiens, ou les simples usagers de l’outil informatique, utilisent des systèmes informatiques pour concevoir ou exécuter des programmes d’application. Nous considérerons qu’un environnement informatique est formé d’une part d’un ordinateur¹ et de ses équipements externes, que nous appellerons *environnement matériel*, et d’autre part d’un système d’exploitation avec ses programmes d’application, que nous appellerons *environnement logiciel*. Les programmes qui forment le logiciel réclament des méthodes pour les construire, des langages pour les rédiger et des outils pour les exécuter sur un ordinateur.

Dans ce chapitre, nous introduirons la terminologie et les notions de base des ordinateurs et de la programmation. Nous présenterons les notions d’environnement de développement et d’exécution d’un programme, nous expliquerons ce qu’est un langage de programmation et nous introduirons les méthodes de construction des programmes.

1.1 ENVIRONNEMENT MATÉRIEL

Un *automate* est un ensemble fini de composants physiques pouvant prendre des états identifiables et reproductibles en nombre fini, auquel est associé un ensemble de changements d’états non instantanés qu’il est possible de commander et d’enchaîner sans intervention humaine.

Un *ordinateur* est un automate *déterministe* à composants électroniques. Tous les ordinateurs, tout au moins les ordinateurs monoprocesseurs, sont construits, peu ou prou, sur le mo-

1. Même si, aujourd’hui, les équipements électroniques qui permettent l’exécution de logiciels deviennent très variés (tablettes numériques, smartphones, montres connectées, processeurs embarqués, etc), dans cet ouvrage nous conserverons le terme *ordinateur* pour désigner l’équipement qui sert à développer les programmes et applications présentés dans cet ouvrage.

dèle proposé en 1944 par le mathématicien américain d'origine hongroise VON NEUMANN. Un ordinateur est muni :

- D'une *mémoire*, dite *centrale* ou *principale*, qui contient deux sortes d'informations : d'une part l'information traitante, les *instructions*, et d'autre part l'information traitée, les *données* (les ordinateurs actuels ont tendance à posséder deux mémoires séparées, une pour les données et l'autre pour les instructions). Cette mémoire est formée d'un ensemble de cellules, ou *mots*, ayant chacune une *adresse unique*, et contenant des instructions ou des données. La représentation de l'information est faite grâce à une codification *binnaire*, 0 ou 1. On appelle *longueur de mot*, caractéristique d'un ordinateur, le nombre d'éléments binaires, appelés *bits*, groupés dans une simple cellule. Les longueurs de mots usuelles des ordinateurs actuels (ou passés) sont, par exemple, 8, 16, 24, 32, 48 ou 64 bits. Cette mémoire possède une capacité *finie*, exprimée en giga-octets (Go); un *octet* est un ensemble de 8 bits, un kilo-octet (Ko) est égal à 1 024 octets, un méga-octet est égal à 1 024 Ko, un giga-octet (Go) est égal à 1 024 Mo, et enfin un téra-octet (To) est égal à 1 024 Go. Actuellement, les tailles courantes des mémoires centrales des ordinateurs individuels varient entre 4 Go à 8 Go².
- D'une *unité centrale de traitement*, formée d'une *unité de commande* (UC) et d'une *unité arithmétique et logique* (UAL). L'unité de commande extrait de la mémoire centrale les instructions et les données sur lesquelles portent les instructions; elle déclenche le traitement de ces données dans l'unité arithmétique et logique, et éventuellement range le résultat en mémoire centrale. L'unité arithmétique et logique effectue sur les données qu'elle reçoit les traitements commandés par l'unité de commande.
- De *registres*. Les registres sont des unités de mémorisation, en petit nombre (certains ordinateurs n'en ont pas), qui permettent à l'unité centrale de traitement de ranger de façon temporaire des données pendant les calculs. L'accès à ces registres est très rapide, beaucoup plus rapide que l'accès à une cellule de la mémoire centrale. Le rapport entre les temps d'accès à un registre et à la mémoire centrale est de l'ordre de 100.
- D'*unités d'échanges* reliées à des périphériques pour échanger de l'information avec le monde extérieur. L'unité de commande dirige les unités d'échange lorsqu'elle rencontre des instructions d'entrée-sortie.

Jusqu'au milieu des années 2000, les constructeurs étaient engagés dans une course à la vitesse avec des microprocesseurs toujours plus rapides. Toutefois, les limites de la physique actuelle ont été atteintes et depuis 2006 la tendance nouvelle est de placer plusieurs (le plus possible) microprocesseurs sur une même puce (le circuit-intégré). Ce sont par exemple les processeurs 64 bits Core i7 d'INTEL ou AMD Ryzen d'AMD, qui possèdent de 4 à 16 cœurs permettant l'exécution en parallèle de plusieurs programmes.

Les ordinateurs actuels possèdent aussi plusieurs niveaux de mémoire. Ils introduisent, entre le processeur et la mémoire centrale, des mémoires dites *caches* qui accélèrent l'accès aux données. Les mémoires caches peuvent être *primaires*, c'est-à-dire situées directement sur le processeur, ou *secondaires*, c'est-à-dire situées sur la carte mère. Certains ordinateurs introduisent même un troisième niveau de cache. En général, le rapport entre le temps d'accès entre les deux premiers niveaux de mémoire cache est d'environ 10 (le cache de niveau 1 est

2. Notez qu'avec 32 bits, l'espace adressage est de 4 Go mais qu'en général les systèmes d'exploitation ne permettent d'utiliser qu'un espace mémoire de taille inférieure. Les machines 64 bits actuelles, avec un système d'exploitation adapté, permettent des tailles de mémoire centrale supérieures à 4 Go.

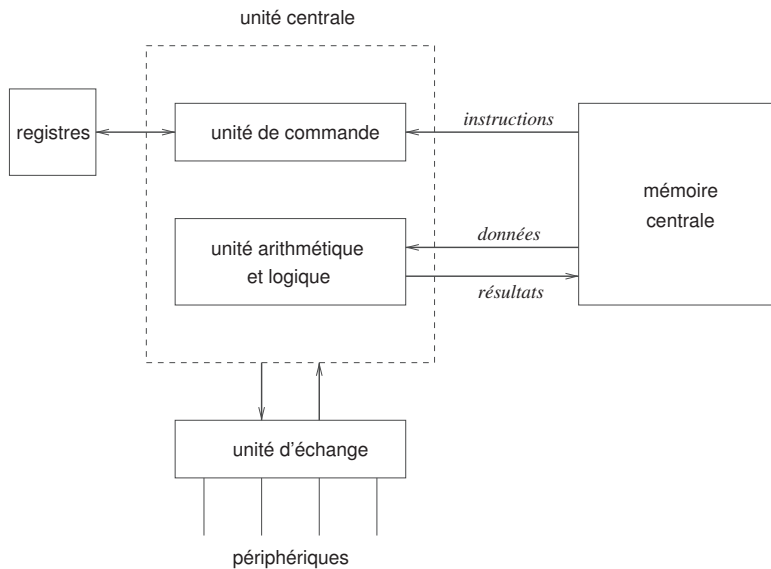


FIGURE 1.1 Structure générale d'un ordinateur.

le plus rapide et le plus petit). Le temps d'accès entre la mémoire cache secondaire et la mémoire centrale est lui aussi d'un rapport d'environ 10.

Les *équipements externes*, ou *périphériques*, sont un ensemble de composants permettant de relier l'ordinateur au monde extérieur, et notamment à ses utilisateurs humains. On peut distinguer :

- Les dispositifs qui servent pour la communication avec l'homme (clavier, écran (tactile), imprimantes, micros, haut-parleurs, scanners, etc.) et qui demandent une transcodification appropriée de l'information, par exemple sous forme de caractères alphanumériques.
- Les mémoires *secondaires*, qui permettent de conserver de l'information, impossible à garder dans la mémoire centrale de l'ordinateur faute de place, ou que l'on désire conserver pour une période plus ou moins longue après l'arrêt de l'ordinateur.

Les mémoires secondaires sont de plus en plus les disques SSD (*Solid-State drive*, disque à semi-conducteurs), et, de moins en moins, les disques durs magnétiques les clés USB, les CD-ROM, les DVD, ou les Blu-ray. Par le passé, les bandes magnétiques ou les disquettes étaient des supports très utilisés. Actuellement les bandes magnétiques ne le sont quasiment plus, la fabrication des disquettes (supports de très petite capacité et peu fiables) a été arrêtée depuis de nombreuses années, et les ordinateurs vendus aujourd'hui ne sont plus pourvus de lecteur/graveur de DVD ou Blu-ray, comme c'était le cas il y encore quelques années.

Aujourd'hui, la capacité des mémoires secondaires atteint des valeurs toujours plus importantes. Alors que certains DVD ont une capacité de 17 Go, qu'un disque Blu-ray atteint 50 Go, et que des clés USB de 64 ou 128 Go sont courantes, un seul disque SSD peut mémoriser jusqu'à 2 téraoctets, et un disque magnétique jusqu'à 10 téraoctets. Des systèmes permettent de regrouper plusieurs disques, vus comme un disque

unique, offrant une capacité de plusieurs centaines de téraoctets. Toutefois, l'accès aux informations sur les supports secondaires reste bien plus lent que celui aux informations placées en mémoire centrale. Pour les disques durs, le rapport est d'environ 10. De nos jours, avec le développement du « nuage » (*cloud computing*), la tendance est plutôt à la limitation des mémoires locales, en faveur de celles, délocalisées et bien plus vastes, proposées par les centres de données (*datacenters*) accessibles par le réseau Internet.

- Les dispositifs qui permettent l'échange d'informations sur un réseau. Pour relier l'ordinateur au réseau, il existe par exemple des connexions filaires comme celles de type *ethernet*, ou des connexions sans fil comme celles de type *WiFi* ou *Bluetooth*.

Le lecteur intéressé par l'architecture et l'organisation des ordinateurs pourra lire avec profit le livre d'A. TANENBAUM [Tan12] sur le sujet.

1.2 ENVIRONNEMENT LOGICIEL

L'ordinateur que fabrique le constructeur est une machine incomplète à laquelle il faut ajouter, pour la rendre utilisable, une quantité importante de programmes variés, qui constituent le *logiciel*.

En général, un ordinateur est livré avec un *système d'exploitation*. Un système d'exploitation est un programme, ou plutôt un ensemble de programmes, qui assurent la gestion des ressources, matérielles et logicielles, employées par le ou les utilisateurs. Un système d'exploitation a pour tâche la gestion et la conservation de l'information (gestion des processus et de la mémoire centrale, système de gestion de fichiers); il a pour rôle de créer l'environnement nécessaire à l'exécution d'un travail, et est chargé de répartir les ressources entre les usagers. Il propose aussi de nombreux protocoles de connexion pour relier l'ordinateur à un réseau. Entre l'utilisateur et l'ordinateur, le système d'exploitation propose une interface *textuelle* au moyen d'un *interprète de commandes* et une interface *graphique* au moyen d'un *gestionnaire de fenêtres*.

Les systèmes d'exploitation des premiers ordinateurs ne permettaient l'exécution que d'une seule tâche à la fois, selon un mode de fonctionnement appelé *traitement par lots* qui assurait l'enchaînement de l'exécution des programmes. À partir des années 1960, les systèmes d'exploitation ont cherché à exploiter au mieux les ressources des ordinateurs en permettant le *temps partagé*, pour offrir un accès simultané à plusieurs utilisateurs.

Jusqu'au début des années 1980, les systèmes d'exploitation étaient dits *propriétaires*. Les constructeurs fournissaient avec leurs machines un système d'exploitation spécifique, et le nombre de systèmes d'exploitation différents était important. Aujourd'hui, ce nombre a considérablement réduit, et seuls quelques-uns sont réellement utilisés dans le monde. Citons, par exemple, WINDOWS, MACOS ou LINUX pour les ordinateurs individuels, et UNIX pour les ordinateurs multi-utilisateurs. Avec l'apparition des tablettes et smartphones, de nouveaux systèmes d'exploitation spécifiquement dédiés à ces appareils sont apparus. Citons par exemple ANDROID ou IOS.

Avec l'augmentation de la puissance des ordinateurs personnels et l'avènement des réseaux mondiaux, les systèmes d'exploitation offrent, en plus des fonctions déjà citées, une quantité extraordinaire de services et d'outils aux utilisateurs. Ces systèmes d'exploitation

modernes mettent à la disposition des utilisateurs tout un ensemble d'applications (traitement de texte, tableurs, outils multimédias, navigateurs pour le web, outils de communication, jeux...) qui leur offrent un environnement de travail pré-construit, confortable et facile d'utilisation.

Le traitement de l'information est l'exécution par l'ordinateur d'une série finie de commandes préparées à l'avance, le *programme*, qui vise à calculer et rendre des résultats, généralement, en fonction de données entrées au début ou en cours d'exécution par l'intermédiaire d'interfaces textuelles ou graphiques. Les commandes qui forment le programme sont décrites au moyen d'un *langage*. Si ces commandes se suivent strictement dans le temps, et ne s'exécutent jamais simultanément, l'exécution est dite *séquentielle*, sinon elle est dite *parallèle*.

Chaque ordinateur possède un langage qui lui est propre, appelé *langage machine*. Le langage machine est un ensemble de commandes élémentaires représentées en code binaire qu'il est possible de faire exécuter par l'unité centrale de traitement d'un ordinateur donné. Le seul langage que comprend l'ordinateur est son langage machine.

Tout logiciel est écrit à l'aide d'un ou plusieurs *langages de programmation*. Un langage de programmation est un ensemble d'énoncés déterministes, qu'il est possible, pour un être humain, de rédiger selon les règles d'une grammaire donnée, et destinés à représenter les objets et les commandes pouvant entrer dans la constitution d'un programme. Ni le langage machine, trop éloigné des modes d'expressions humains, ni les langues naturelles écrites ou parlées, trop ambiguës, ne sont des langages de programmation.

La production de logiciel est une activité difficile et complexe, et les éditeurs font payer, parfois très cher, leur logiciel dont le code source n'est, en général, pas distribué. Toutefois, tous les logiciels ne sont pas payants. La communauté internationale des informaticiens produit depuis longtemps du logiciel gratuit (ce qui ne veut pas dire qu'il est de mauvaise qualité, bien au contraire) mis à la disposition de tous. Il existe aux États-Unis³, une fondation, la FSF (*Free Software Foundation*), à l'initiative de R. STALLMAN, qui a pour but la promotion de la construction du logiciel *libre*, ainsi que celle de sa distribution. Libre ne veut pas dire nécessairement gratuit⁴, bien que cela soit souvent le cas, mais indique que le texte source du logiciel est disponible. D'ailleurs, la FSF propose une licence, GNU GPL, afin de garantir que les logiciels sous cette licence soient *libres* d'être redistribués et modifiés par tous leurs utilisateurs.

Le système d'exploitation LINUX est disponible librement depuis de nombreuses années, et aujourd'hui certains éditeurs suivent ce courant en distribuant, comme Apple par exemple, gratuitement la dernière version de leur système d'exploitation *High Sierra* (toutefois sans le code source).

3. FSF France a pour but la promotion du logiciel libre en France (<http://fsffrance.org>).

4. La confusion provient du fait qu'en anglais le mot « free » possède les deux sens.

1.3 LES LANGAGES DE PROGRAMMATION

Nous venons de voir que chaque ordinateur possède un langage qui lui est propre : le langage machine, qui est en général totalement incompatible avec celui d'un ordinateur d'un autre modèle. Ainsi, un programme écrit dans le langage d'un ordinateur donné ne pourra être réutilisé sur un autre ordinateur.

Le langage *d'assemblage* est un codage alphanumérique du langage machine. Il est plus lisible que ce dernier et surtout permet un adressage relatif de la mémoire. Toutefois, comme le langage machine, le langage d'assemblage est lui aussi dépendant d'un ordinateur donné (voire d'une famille d'ordinateurs) et ne facilite pas le transport des programmes vers des machines dont l'architecture est différente. L'exécution d'un programme écrit en langage d'assemblage nécessite sa traduction préalable en langage machine par un programme spécial, l'*assembleur*.

Le texte qui suit⁵, écrit en langage d'assemblage d'un Core i7 d'Intel, correspond à l'appel, de la fonction C `printf("Bonjour\n")` qui écrit *Bonjour* sur la sortie standard (e.g. l'écran) depuis la fonction `main`.

```
.LC0:
    .string "Bonjour"
    .text
    .globl main
    .type main, @function

main:
.LFB0:
    .cfi_startproc
    pushq   %rbp
    .cfi_def_cfa_offset 16
    .cfi_offset 6, -16
    movq    %rsp, %rbp
    .cfi_def_cfa_register 6
    leaq    .LC0(%rip), %rdi
    call    puts@PLT
    movl    $0, %eax
    popq    %rbp
    .cfi_def_cfa 7, 8
    ret
    .cfi_endproc

.LFE0:
```

Le langage d'assemblage, comme le langage machine, est d'un niveau très élémentaire (une suite linéaire de commandes et sans structure) et, comme le montre l'exemple précédent, guère lisible et compréhensible. Son utilisation par un être humain est alors difficile, fastidieuse et sujette à erreurs.

Ces défauts, entre autres, ont conduit à la conception des langages de programmation dits de *haut niveau*. Un langage de programmation de haut niveau offrira au programmeur des

5. Produit par le compilateur gcc.

moyens d'expression structurés proches des problèmes à résoudre et qui amélioreront la fiabilité des programmes. Pendant de nombreuses années, les ardents défenseurs de la programmation en langage d'assemblage avançaient le critère de son efficacité. Les optimiseurs de code ont balayé cet argument depuis longtemps, et les défauts de ces langages sont tels que leurs thuriféraires sont de plus en plus rares.

Si on ajoute à un ordinateur un langage de programmation, tout se passe comme si l'on disposait d'un nouvel ordinateur (une machine abstraite), dont le langage est maintenant adapté à l'être humain, aux problèmes qu'il veut résoudre et à la façon qu'il a de comprendre et de raisonner. De plus, cet ordinateur fictif pourra recouvrir des ordinateurs différents, si le langage de programmation peut être installé sur chacun d'eux. Ce dernier point est très important, puisqu'il signifie qu'un programme écrit dans un langage de haut niveau pourra être exploité (théoriquement) sans modification sur des ordinateurs différents.

La définition d'un langage de programmation recouvre trois aspects fondamentaux. Le premier, appelé *lexical*, définit les symboles (ou caractères) qui servent à la rédaction des programmes et les règles de formation des mots du langage. Par exemple, un entier décimal sera défini comme une suite de chiffres compris entre 0 et 9. Le second, appelé *syntactique*, est l'ensemble des règles grammaticales qui organisent les mots en phrases. Par exemple, la phrase « 234 / 54 », formée de deux entiers et d'un opérateur de division, suit la règle grammaticale qui décrit une expression. Le dernier aspect, appelé *sémantique*, étudie la signification des phrases. Il définit les règles qui donnent un sens aux phrases. Notez qu'une phrase peut être syntaxiquement valide, mais incorrecte du point de vue de sa sémantique (e.g. 234/0, une division par zéro est invalide). L'ensemble des règles lexicales, syntaxiques et sémantiques définit un langage de programmation, et on dira qu'un programme appartient à un langage de programmation donné s'il vérifie cet ensemble de règles. La vérification de la conformité lexicale, syntaxique et sémantique d'un programme est assurée automatiquement par des analyseurs qui s'appuient, en général, sur des notations formelles qui décrivent sans ambiguïté l'ensemble des règles.

Comment exécuter un programme rédigé dans un langage de programmation de haut niveau sur un ordinateur qui, nous le savons, ne sait traiter que des programmes écrits dans son langage machine ? Voici deux grandes familles de méthodes qui permettent de résoudre ce problème :

- La première méthode consiste à *traduire* le programme, appelé *source*, écrit dans le langage de haut niveau, en un programme sémantiquement *équivalent* écrit dans le langage machine de l'ordinateur (voir la figure 1.2). Cette traduction est faite au moyen d'un logiciel spécialisé appelé *compilateur*. Un compilateur possède au moins quatre phases : trois phases d'analyse (lexicale, syntaxique et sémantique), et une phase de production de code machine. Bien sûr, le compilateur ne produit le code machine que si le programme source respecte les règles du langage, sinon il devra signaler les erreurs au moyen de messages précis. En général, le compilateur produit du code pour un seul type de machine, celui sur lequel il est installé. Notez que certains compilateurs, dits *multicibles*, produisent du code pour différentes familles d'ordinateurs.
- Nous avons vu qu'un langage de programmation définit un ordinateur fictif. La seconde méthode consiste à *simuler* le fonctionnement de cet ordinateur fictif sur l'ordinateur réel par *interprétation* des instructions du langage de programmation de haut niveau.