Claude Delannoy

S'initier à la PROGRAMMATION et à L'ORIENTÉ OBJET

Avec des exemples en C, C++, C#, Python, Java et PHP

2e édition

Code source des exemples du livre

EYROLLES

S'initier à la PROGRAMMATION et à L'ORIENTÉ OBJET

2e édition

Acquérir rapidement une parfaite maîtrise des techniques de programmation et savoir s'adapter facilement à tout nouveau langage

Conçu pour les débutants en programmation, cet ouvrage commence par un apprentissage progressif et rigoureux des notions de programmation procédurale communes à tous les langages (types de données, variables, opérateurs, instructions de contrôle, fonctions, tableaux...), avant d'aborder les notions propres aux langages orientés objet.

L'auteur utilise, pour faciliter l'assimilation des concepts, un pseudo-code complet mais simple d'accès, qui évite de se perdre dans les spécificités de tel ou tel langage. Chaque notion est d'abord présentée à l'aide du pseudo-code, avant d'être illustrée d'exemples d'implémentation en langages C, C++, C#, Java, PHP et, nouveauté de cette 2e édition, en langage Python. De nombreux exercices corrigés permettent au lecteur de contrôler ses connaissances à chaque étape de l'apprentissage.

À qui s'adresse ce livre ?

- Aux étudiants en 1ère année de cursus informatique (BTS, DUT, licences, écoles d'ingénieur).
- Aux autodidactes ou professionnels de tous horizons souhaitant s'initier à la programmation.
- A tous ceux qui ont appris un langage « sur le tas » et ressentent le besoin d'approfondir leurs connaissances pour gagner en efficacité et en qualité et s'adapter plus facilement à de nouveaux langages.
- Aux enseignants et formateurs à la recherche d'une méthode pédagogique et d'un support de cours structuré pour enseigner la programmation à des débutants.

Sur le site www.editions-eyrolles.com

- Dialoguez avec l'auteur
- Téléchargez le code source des exemples du livre

Ingénieur informaticien au CNRS, **Claude Delannoy** possède une grande pratique de la formation continue et de l'enseignement supérieur. Réputés pour la qualité de leur démarche pédagogique, ses ouvrages sur les langages et la programmation totalisent plus de 500 000 exemplaires vendus.

Au sommaire

Chaque chapitre comporte une rubrique « Côté langage », qui montre comment les concepts introduits à l'aide du pseudo-code s'expriment en C, C++, C#, Java, Python et PHP, et une rubrique « Exemples langages », qui propose plusieurs programmes complets écrits dans ces différents langages.

www.editions-eyrolles.com
Groupe Eyrolles | Diffusion Geodif

Code éditeur : G11826
ISBN : 978-2-212-11826-1

S'initier à la PROGRAMMATION et à L'ORIENTÉ OBJET

AUX EDITIONS EYROLLES

Du même auteur

C. Delannoy. – **Programmer en Java**. *Java 8*.
N°14007, 9e édition, 2014, 940 pages.

C. Delannoy. – **Exercices en Java**.
N°14009, 4e édition, 2014, 360 pages.

C. Delannoy. – **Programmer en langage C++.**
N°14008, 8e édition, 2011, 820 pages (réédition avec nouvelle présentation, 2014).

C. Delannoy. – **Programmer en langage C**. *Cours et exercices corrigés*.
N°11825, 5e édition, 2009, 276 pages (réédition avec nouvelle présentation, 2016).

C. Delannoy. – **Exercices en langage C**.
N°11105, 1997, 260 pages.

C. Delannoy. – **Le guide complet du langage C.**
N°14012, 2014, 844 pages.

Autres ouvrages

G. Dowek, et coll. – **Informatique et sciences du numérique – Edition spéciale Python.**
Manuel de spécialité ISN en terminale - Avec des exercices corrigés et des idées de projets
N°13676, 2013, 342 pages.

B. Wack, et coll. – **Informatique pour tous en classes préparatoires aux grandes écoles.**
Manuel d'algorithmique et programmation structurée avec Python – Voies MP, PC, PSI, PT, TPC et TSI
N°13700, 2013, 420 pages.

G. Swinnen. – **Apprendre à programmer avec Python 3.** *Avec 60 pages d'exercices corrigés*
N°13434, 3e édition, 2012, 430 pages.

J. Engels. – **HTML5 et CSS3 : cours et exercices.**
N°13400, 2012, 550 pages.

R. Rimelé. – **HTML5.**
N°14365, 3e édition, 750 pages environ, à paraître au 4e trimestre 2016.

J. Engels. – **PHP 5 : cours et exercices.**
N°13725, 3e édition, 2013, 630 pages.

E. Daspet, C. Pierre de Geyer, F. Hardy. – **PHP 7 avancé**.
N°14357, 700 pages environ, à paraître au 4e trimestre 2016.

P. Roques. – **UML 2 par la pratique**
N°12565, 7e édition, 2009, 396 pages.

Claude Delannoy

S'initier à la PROGRAMMATION et à L'ORIENTÉ OBJET

Avec des exemples en C, C++, C#, Python, Java et PHP

2e édition

Deuxième tirage 2016, avec nouvelle présentation

EYROLLES

ÉDITIONS EYROLLES
61, bd Saint-Germain
75240 Paris Cedex 05
www.editions-eyrolles.com

La seconde édition du présent ouvrage est parue en 2014 sous l'ISBN 978-2-212-14067-5. À l'occasion de ce deuxième tirage, elle bénéficie d'une nouvelle couverture. Le texte reste inchangé.

 ISBN : 978-2-212-11826-1.

Avant-propos

Objectif de l'ouvrage

Ce livre se propose de vous apprendre à programmer en exprimant les concepts fondamentaux à l'aide d'un « pseudo-code ». Cela vous permet de rédiger des programmes en privilégiant l'aspect algorithmique, sans être pollué par la complexité et la technicité d'un langage donné. Par ailleurs, l'ouvrage montre comment ces concepts fondamentaux se traduisent dans six langages très usités (C, C++, Java, C#, Python et PHP) et fournit des exemples complets. Il prépare ainsi efficacement à l'étude d'un langage réel.

Forme de l'ouvrage

L'ouvrage a été conçu sous forme d'un cours, avec une démarche très progressive. De nombreux exemples complets, écrits en pseudo-code et accompagnés du résultat fourni par leur exécution, viennent illustrer la plupart des concepts fondamentaux. Des exercices appropriés proposent la rédaction de programmes en pseudo-code, permettant ainsi la mise en pratique des acquis. Plutôt que de les regrouper classiquement en fin de chapitre, nous avons préféré les placer aux endroits jugés opportuns pour leur résolution. Une correction est fournie en fin de volume ; nous vous encourageons vivement à ne la consulter qu'après une recherche personnelle et à réfléchir aux différences de rédaction qui ne manqueront pas d'apparaître.

Chaque chapitre se termine par :

- Une rubrique « Côté langages » qui montre comment les concepts exposés préalablement s'expriment dans les six langages choisis ; elle constitue une sorte de guide de traduction du

pseudo-code dans un véritable langage. Notez que le langage C n'étant pas orienté objet, il n'est pris en compte que jusqu'au chapitre 8.

- Une rubrique « Exemples langages » qui propose des programmes complets, traduction de certains des exemples présentés en pseudo-code.

À qui s'adresse cet ouvrage

Cet ouvrage s'adressse aux débutants en programmation et aux étudiants du premier cycle d'université. Il peut également servir :

- à ceux qui apprennent à programmer directement dans un langage donné : il leur permettra d'accompagner leur étude, en dégageant les concepts fondamentaux et en prenant un peu de recul par rapport à leur langage ;
- à ceux qui maîtrisent déjà la programmation dans un langage donné et qui désirent « passer à un autre langage » ;
- à ceux qui connaissent déjà la programmation procédurale et qui souhaitent aborder la programmation orientée objet.

Enfin, sa conception permet à l'ouvrage d'être facilement utilisé comme « support de cours ».

Plan de l'ouvrage

Le **chapitre 1** présente le rôle de l'ordinateur, les grandes lignes de son fonctionnement et la manière de l'utiliser. Il dégage les importantes notions de langage, de programme, de données et de résultats, de système d'exploitation et d'environnement de programmation.

Le **chapitre 2** introduit les concepts de variable et de type, et la première instruction de base qu'est l'affectation. Il se limite à trois types de base : les entiers, les réels et les caractères. Il présente les erreurs susceptibles d'apparaître dans l'évaluation d'une expression et les différentes façons dont un langage donné peut les gérer. On y inclut les notions d'expression mixte et d'expression constante.

Le **chapitre 3** est consacré aux deux autres instructions de base que sont la lecture et l'écriture. Il nous a paru utile de les placer à ce niveau pour permettre, le plus rapidement possible, de présenter et de faire écrire des programmes complets. On situe ces instructions par rapport aux différents modes de communication entre l'utilisateur et le programme : mode console, programmation par événements, mode batch, programmation Internet.

Le **chapitre 4** étudie la structure de choix, en présentant la notion de condition et en introduisant le type booléen. On y aborde les choix imbriqués. L'existence de structures de choix multiple (instruction `switch` des six langages examinés) est évoquée dans la partie « Côté langages ».

Le **chapitre 5** aborde tout d'abord les structures de répétition conditionnelle. Il présente la notion de compteur, avant d'examiner les structures de répétition inconditionnelle (ou « avec compteur ») et les risques inhérents à la modification intempestive du compteur.

Le **chapitre 6** présente les « algorithmes élémentaires » les plus usuels : comptage, accumulation, recherche de maximum, imbrication de répétitions. Il donne un aperçu de ce qu'est l'itération.

Le **chapitre 7** traite des tableaux, à une ou deux dimensions. Il se place a priori dans un contexte de gestion statique des emplacements mémoire correspondants et il décrit les contraintes qui pèsent alors sur la taille d'un tableau. Les autres modes de gestion (automatique et dynamique) sont néanmoins évoqués en fin de chapitre, ainsi que la notion de tableau associatif (utilisé par exemple par PHP) qui est comparée à celle de tableau indicé. Les situations de débordement d'indice sont examinées, avec leurs conséquences potentielles dépendantes du langage.

Le **chapitre 8** est consacré aux fonctions. Il présente les notions de paramètres, de variable locale et de résultat, et distingue la transmission par valeur de la transmission par référence (par adresse), en examinant le cas particulier des tableaux. Il aborde la durée de vie des variables locales, ce qui amène à traiter du mode de gestion automatique correspondant (et du concept de pile qu'il utilise souvent). Il dégage le concept de « programme principal » ou de « fonction principale ». Enfin, il donne un aperçu de ce qu'est la récursivité.

Le **chapitre 9** introduit les notions de classe, d'attribut, d'objet, de méthode, d'encapsulation des données et de constructeur. Il fournit quelques éléments concernant les deux modes de gestion possibles des objets, à savoir par référence ou par valeur. Il étudie les possibilités d'amendement du principe d'encapsulation par modification des droits d'accès aux attributs ou aux méthodes.

Le **chapitre 10** examine l'incidence du mode de gestion des objets (par référence ou par valeur) sur l'affectation d'objets et sur la durée de vie des objets locaux. Il aborde les objets transmis en paramètre et il convient, comme c'est le cas dans la plupart des langages objet, que « l'unité d'encapsulation est la classe et non l'objet ». Il analyse le cas des objets fournis en résultat. Puis, il étudie les attributs et les méthodes de classe, et traite sommairement des tableaux d'objets et des problèmes qu'ils posent dans l'appel des constructeurs, ainsi que des situations « d'auto-référence ».

Le **chapitre 11** est consacré à la composition des objets, c'est-à-dire au cas où un attribut d'une classe est lui-même de type classe. Il examine les problèmes qui peuvent alors se poser au niveau des droits d'accès et dans la nature de la relation qui se crée entre les objets concernés. Il présente la distinction entre copie profonde et copie superficielle d'un objet. Il montre également comment résoudre un problème fréquent, à savoir réaliser une classe à instance unique (singleton).

Le **chapitre 12** présente la notion d'héritage ou de classe dérivée et son incidence sur les droits d'accès aux attributs et aux méthodes. Il fait le point sur la construction des objets dérivés avant de traiter de la redéfinition des méthodes. Il aborde les situations de dérivations

successives et décrit succinctement les possibilités de modification des droits d'accès lors de la définition d'une classe dérivée.

Le **chapitre 13** expose les notions de base du polymorphisme, à savoir la compatibilité par affectation et la ligature dynamique. Il en examine les conséquences dans plusieurs situations et montre quelles sont les limites de ce polymorphisme, ce qui conduit, au passsage, à parler de valeurs de retour covariantes présentes dans certains langages.

Le **chapitre 14** traite enfin de concepts moins fondamentaux que l'héritage ou le polymorphisme, parfois absents de certains langages, mais qui peuvent faciliter la conception des logiciels. Il s'agit des notions de classes abstraites (ou retardées), d'interface et d'héritage multiple.

Justifications de certains choix

Voici quelques éléments justifiant les choix que nous avons opéré dans la conception de cet ouvrage.

- Nous présentons la programmation procédurale avant d'introduire la programmation objet, pour différentes raisons :
  - la plupart des langages objet actuels offrent des possibilités de programmation procédurale ;
  - la programmation orientée objet s'appuie sur les concepts de la programmation procédurale ; la seule exception concerne la notion de fonction indépendante qui peut être absente de certains langages objet mais qui se retrouve quand même sous une forme très proche dans la notion de méthode ;
  - sur un plan pédagogique, il est difficile d'introduire directement des méthodes dans une classe si l'on n'a pas encore étudié l'algorithmique procédurale.
- Dans le choix des concepts fondamentaux, nous avons évité de nous limiter à un sous-ensemble commun à tous les langages car cela aurait été trop réducteur à notre sens. Nous avons choisi les concepts qu'il nous a paru important de maîtriser pour pouvoir ensuite aborder la programmation dans n'importe quel langage.

  Ces choix font ainsi du pseudo-code, non pas la « matrice » de tous les langages, mais plutôt un langage à part entière, simple, mais renfermant la plupart des concepts fondamentaux de la programmation – certains pouvant ne pas exister dans tel ou tel langage. C'est précisément le rôle de la partie « Côté langages » que de monter en quoi les langages réels peuvent différer les uns des autres et ce au-delà de leur syntaxe (les six langages choisis possèdent la même syntaxe de base et jouissent pourtant de propriétés différentes). Ainsi, le lecteur est non seulement amené à programmer en pseudo-code mais, en même temps, il est préparé à affronter un vrai langage. Voici par exemple quelques points sur lesquels les langages peuvent se différencier les uns des autres :

  - mode de traduction : compilation (C, C++), interprétation (PHP ou Python) ou traduction dans un langage intermédiaire (Java) ;
  - mode de gestion de la mémoire : statique, automatique, dynamique ;
  - nature des expressions constantes et des expression mixtes ;
  - gestion des tableaux (sous forme indicée ou sous forme de tableau associatif comme en PHP) ;
  - mode de transmission des paramètres d'une fonction : par valeur, par référence ;
  - utilisation pour les objets d'une « sémantique de valeur » ou d'une « sémantique de référence » (Java n'utilise que la deuxième, tandis que C++ utilise les deux) ;
  - mode standard de recopie des objets : copie superficielle ou copie profonde.
- Nous n'avons pas introduit de type chaîne car son implémentation varie fortement suivant les langages (type de base dans certains langages procéduraux, type hybride en C, type classe dans les langages objet...). Sa gestion peut se faire par référence ou par valeur. Dans certains langages, ces chaînes sont constantes (non modifiables), alors qu'elles sont modifiables dans d'autres...

Table des matières

1

Ordinateur, programme et langage

Ce chapitre expose tout d'abord les notions de programme et de traitement de l'information. Nous examinerons ensuite le rôle de l'ordinateur et ses différents constituants. Nous aborderons alors l'importante notion de langage de programmation et nous vous indiquerons succinctement quels sont les concepts fondamentaux que l'on rencontre dans la plupart des langages actuels, ce qui nous permettra d'introduire la démarche que nous utiliserons dans la suite de l'ouvrage.

1 Le rôle de l'ordinateur

1.1 La multiplicité des applications

Les applications de l'ordinateur sont très nombreuses. En voici quelques exemples :

- accès à Internet ;
- envoi de courrier électronique ;
- création de sites Web ;
- lecture de CD-Rom ou de DVD ;
- archivage et retouche de photos ;
- jeux vidéo ;

- bureautique : traitement de texte, tableur, gestion de bases de données... ;
- gestion et comptabilité : facturation, paye, stocks... ;
- analyse numérique ;
- prévisions météorologiques ;
- aide à la conception électronique (CAO) ou graphique (DAO) ;
- pilotage de satellites, d'expériences...

1.2 Le programme : source de diversité

Si un ordinateur peut effectuer des tâches aussi variées, c'est essentiellement parce qu'il est possible de le programmer. Effectivement, l'ordinateur est capable de mettre en mémoire un programme qu'on lui fournit ou, plus souvent qu'on lui désigne (en général, on fournira le moyen de trouver le programme, plutôt que le programme lui-même) puis de l'exécuter.

Plus précisément, un ordinateur possède un répertoire limité d'opérations élémentaires qu'il sait exécuter très rapidement. Un programme est alors constitué d'un ensemble de directives, nommées instructions, qui spécifient :

- les opérations élémentaires à exécuter ;
- la manière dont elles s'enchaînent.

En définitive, la vitesse d'exécution de l'ordinateur fait sa puissance ; le programme lui donne sa souplesse. En particulier, nous verrons que certaines des instructions permettent soit de répéter plusieurs fois un ensemble donné d'instructions, soit de choisir entre plusieurs ensembles d'instructions.

1.3 Les données du programme, les résultats

Supposez qu'un enseignant dispose d'un ordinateur et d'un programme de calcul de moyennes de notes. Pour fonctionner, un tel programme nécessite qu'on lui fournisse les notes dont on cherche la moyenne. Nous les appellerons informations données ou plus simplement données. En retour, le programme va fournir la moyenne cherchée. Nous l'appellerons information résultat ou plus simplement résultat. Si le programme a été prévu pour cela, il peut fournir d'autres résultats tels que le nombre de notes supérieures à 10.

De la même manière, un programme de paye nécessite des données telles que le nom des différents employés, leur situation de famille, leur numéro de sécurité sociale et différentes informations permettant de déterminer leur salaire du mois. Parmi les résultats imprimés sur les différents bulletins de salaire, on trouvera notamment : le salaire brut, les différentes retenues légales, le salaire net...

Un programme de réservation de billet d'avion par Internet nécessitera des données telles que votre nom, votre numéro de carte d'identité ainsi que le choix du vol.

1.4 Communication ou archivage

D'où proviennent les données ? Que deviennent les résultats ? Les réponses les plus naturelles sont : les données sont communiquées au programme par l'utilisateur ; les résultats sont communiqués à l'utilisateur par le programme.

Cela correspond effectivement à une situation fort classique dans laquelle l'ordinateur doit être en mesure de communiquer avec l'homme. Cependant, les données ne sont pas toujours fournies manuellement. Par exemple, dans le cas d'un programme de paye, il est probable que certaines informations relativement permanentes (noms, numéros de sécurité sociale...) auront été préalablement archivées dans un fichier ou dans une base de données. Le programme y accédera alors directement.

Dans le cas de la réservation d'un billet d'avion, vous n'aurez pas à fournir explicitement les caractéristiques du vol souhaité (aéroport de départ, aéroport d'arrivée, heure de départ, numéro de vol...). Vous effectuerez un choix parmi une liste de possibilités que le programme aura trouvées, là encore, dans une base de données, en fonction de certains critères que vous aurez exprimés.

On notera, à propos des bases de données, que les informations qu'elles renfement auront dû être préalablement archivées par d'autres programmes dont elles constituaient alors les résultats. Ceci montre la relativité de la notion de donnée ou de résultat. Une même information peut être tantôt donnée, tantôt résultat, suivant l'usage que l'on en fait.

En général, cet échange d'informations entre programme et milieu extérieur paraît assez naturel. En revanche, on voit que le programme représente lui-même une information particulière. Comme les données, il sera, soit prélevé automatiquement dans des archives (de programmes, cette fois), soit (plus rarement) communiqué à l'ordinateur par l'homme.

Qui plus est, nous verrons, dès que nous parlerons de langage et de traducteur, que l'information programme pourra, elle aussi, apparaître tantôt comme donnée, tantôt comme résultat d'un autre programme.

2 Pour donner une forme à l'information : le codage

2.1 L'ordinateur code l'information

Lorsque nous échangeons de l'information avec d'autres personnes, nous utilisons des chiffres, des lettres, des graphiques, des paroles, etc.

Or, pour des raisons purement technologiques, l'ordinateur ne peut traiter ou manipuler qu'une information exprimée sous forme binaire. On imagine souvent une telle information comme une suite de 0 et de 1, mais on pourrait en fait utiliser n'importe quel couple de symboles (comme rond blanc et rond noir, ampoule allumée et ampoule éteinte).

Quand vous transmettez une information à l'ordinateur, par exemple en tapant sur les touches d'un clavier, il est nécessaire qu'il la transforme en binaire. Nous dirons qu'il réalise un codage en binaire de cette information. De la même manière, avant de vous fournir un résultat, il devra opérer une transformation symétrique.

2.2 L'homme code l'information

En toute rigueur, l'ordinateur n'est pas le seul à coder l'information. Pour vous en convaincre, considérez cette petite phrase :

13, treize, vous avez dit XIII.

Vous constatez que la même valeur apparaît exprimée sous trois formes différentes :

13
treize
XIII

La première forme s'exprime avec deux symboles, chacun étant choisi parmi les chiffres de 0 à 9. Nous disons que nous avons utilisé deux positions d'un code à dix moments (les dix chiffres 0, 1, 2... 9).

La deuxième forme s'exprime avec six symboles (lettres), chacun étant choisi parmi les vingt-six lettres de l'alphabet. Nous disons que nous avons utilisé six positions d'un code à vingt-six moments.

La dernière forme s'exprime avec quatre positions d'un code à sept moments (les lettres représentant les chiffres romains : I, V, X, L, C, M et D).

Quant aux codes binaires employés par l'ordinateur, ce sont tout simplement des codes à deux moments puisqu'il suffit de deux symboles pour exprimer une information binaire. Le choix de ces deux symboles est purement conventionnel ; généralement on emploie les deux premiers chiffres de notre système décimal. Ainsi :

```
10011010
```

représente une information binaire utilisant huit positions. Chaque position porte le nom de bit, terme qui est donc l'équivalent, pour les codes binaires, des termes chiffres ou lettres employés par les codes rencontrés précédemment.

2.3 Ce qui différencie l'homme de l'ordinateur

En définitive, on peut se dire que l'ordinateur et l'homme diffèrent dans leur façon de représenter l'information puisque l'ordinateur ne connaît que le binaire tandis que l'homme est capable d'utiliser une très grande variété de codes. Mais est-ce bien la seule différence ?

En fait, lorsque, dans un texte, vous rencontrez « 13 » ou « bonjour », il n'est pas besoin qu'on vous précise quel code a été effectivement employé. Au vu des symboles utilisés, vous arrivez à leur attribuer une signification. Qui plus est, lorsque vous rencontrez « XIII » dans la petite phrase du paragraphe 2.2, vous reconnaissez immédiatement le code « chiffres romains » et non plus le code « lettres de l'alphabet » (et ceci, bien que les chiffres romains

soient des lettres de l'alphabet !). Dans ce cas, vous avez utilisé votre expérience, votre intelligence et le contexte de la phrase pour attribuer une signification à « XIII ».

Le binaire, en revanche, est beaucoup moins naturel, non seulement pour l'homme mais également pour l'ordinateur. Pour vous en convaincre, imaginez que vous ayez besoin de savoir ce que représentent les huit bits 00101001. Certes, vous pouvez toujours dire que cela peut représenter l'écriture en binaire du nombre entier 41. Mais pourquoi cela représenterait-il un nombre ? En effet, toutes les informations (nombres, textes, instructions de programme, dessins, photos, vidéos,...) devront, au bout du compte, être codées en binaire. Dans ces conditions, les huit bits ci-dessus peuvent très bien représenter une lettre, un nombre, une instruction de programme ou tout autre chose.

En définitive, nous voyons que l'ordinateur code l'information ; l'homme agit de même. Cependant, on pourrait dire que l'ordinateur « code plus » que l'homme ; en effet, il n'est pas possible d'attribuer un sens à la seule vue d'une information binaire. Il est, en outre, nécessaire de savoir comment elle a été codée. Nous verrons qu'une conséquence immédiate de ce phénomène réside dans l'importante notion de type, lequel indique précisément le codage utilisé pour représenter une information.

3 Fonctionnement de l'ordinateur

Après avoir vu quel était le rôle de l'ordinateur, nous allons maintenant exposer succinctement ses constituants et son fonctionnement.

3.1 À chacun son rôle

Nous avons donc vu qu'un ordinateur :

- traite l'information grâce à un programme qu'il mémorise ;
- communique et archive des informations.

Ces différentes fonctions correspondent en fait à trois constituants différents :

- La mémoire centrale**,** qui permet de mémoriser les programmes pendant le temps nécessaire à leur exécution. On y trouve également les informations temporaires manipulées par ces programmes : données après leur introduction, résultats avant leur communication à l'extérieur, informations intermédiaires apparaissant pendant le déroulement d'un programme. Signalons que, parfois, on emploie le terme donnée à la place de celui d'information : on dit alors qu'en mémoire centrale, se trouvent à la fois le programme et les données manipulées par ce programme.
- L'unité centrale, qui est la partie active de l'ordinateur. Elle est chargée de prélever en mémoire, une à une, chaque instruction de programme et de l'exécuter. D'ores et déjà, nous pouvons distinguer deux sortes d'instructions :

  - celles qui agissent sur des informations situées en mémoire centrale ; ce sont elles qui permettent véritablement d'effectuer le traitement escompté ;
  - celles qui assurent la communication ou l'archivage d'informations ; elles réalisent en fait un échange d'informations entre la mémoire centrale et d'autres dispositifs nommés périphériques.
- Les périphériques (évoqués ci-dessus), qui correspondent à tous les appareils susceptibles d'échanger des informations avec la mémoire centrale. On en distingue deux sortes :
  - ceux qui assurent la communication entre l'homme et l'ordinateur : clavier, écran, imprimante, souris... ;
  - ceux qui assurent l'archivage d'informations : disque dur, disquette, CD-Rom, DVD, bandes magnétiques... Ils ont un rôle de mémorisation d'informations, au même titre que la mémoire centrale dont ils constituent en quelque sorte un prolongement ; nous verrons d'ailleurs que leur existence ne se justifie que pour des considérations de coût.

Examinons maintenant le fonctionnement de chacun des constituants de l'ordinateur.

3.2 La mémoire centrale

Comme nous l'avons déjà évoqué, il se trouve qu'actuellement ce sont les systèmes de mémorisation binaire qui sont les moins coûteux. C'est pourquoi la mémoire centrale est formée d'éléments dont chacun ne peut prendre que deux états distincts. Autrement dit, chacun de ces éléments correspondant à un bit d'information.

Pour que la mémoire soit utilisable, il faut que l'unité centrale puisse y placer une information et la retrouver. Dans toutes les machines actuelles, cela est réalisé en manipulant, non pas un simple bit, mais au minimum un groupe de huit bits qu'on nomme un octet. Chaque octet est repéré par un numéro qu'on nomme son adresse. Un dispositif, associé à cette mémoire permet :

- soit d'aller chercher en mémoire un octet d'adresse donnée ; notez bien que, dans ce cas, le contenu du mot en question n'est pas modifié (il n'y a pas vraiment prélèvement, mais plutôt recopie) ;
- soit d'aller ranger une information donnée dans un octet d'adresse donnée ; naturellement, l'ancienne information figurant à cette adresse est remplacée par la nouvelle.

En général, les différents octets de la mémoire peuvent accueillir indifféremment des instructions de programme ou des informations.

La plupart des machines actuelles offrent la possibilité de manipuler simultanément plusieurs octets consécutifs. On parle parfois de mot pour désigner un nombre donné d'octets (mais ce nombre varie d'une machine à l'autre).

Remarque

À la place du terme mémoire centrale, on emploie également souvent celui de mémoire vive, ou encore de RAM, abréviation de *Random Access Memory* (mémoire à accès aléatoire).

3.3 L'unité centrale

Elle sait exécuter, très rapidement, un certain nombre d'opérations très simples telles que :

- addition, soustraction, multiplication ou division de nombres codés dans des mots (suite d'octets) de la mémoire centrale ;
- comparaison de valeurs contenues dans deux octets ou dans deux mots ;
- communication à un périphérique d'une information élémentaire (contenue dans un octet ou un mot).

Chaque instruction de programme doit préciser :

- la nature de l'opération à réaliser ; il s'agit d'un numéro (codé en binaire, bien sûr) qu'on appelle « code opération » ;
- les adresses (ou l'adresse) des informations sur lesquelles doit porter l'opération.

L'unité centrale est conçue pour exécuter les instructions dans l'ordre naturel où elles figurent en mémoire. Cependant, pour qu'un programme puisse réaliser des choix ou des répétitions, il est nécessaire de pouvoir rompre cet ordre. C'est pourquoi il existe également des instructions particulières dites de branchement. Elles demandent à l'unité centrale de poursuivre l'exécution du programme à une adresse donnée, au lieu de poursuivre naturellement « en séquence ». Ces branchements peuvent être conditionnels ; autrement dit, ils peuvent n'avoir lieu que si une certaine condition (par exemple égalité de deux valeurs) est réalisée.

3.4 Les périphériques

Comme nous l'avons vu, ils servent à échanger de l'information avec la mémoire centrale et ils se classent en deux grandes catégories : communication et archivage.

3.4.1 Les périphériques de communication

Les plus répandus sont certainement le clavier, l'écran, l'imprimante et la souris. Il en existe cependant beaucoup d'autres tels que les tables traçantes, les écrans tactiles, les synthétiseurs de parole, les lecteurs optiques de caractères, les lecteurs de codes-barres...

3.4.2 Les périphériques d'archivage

La mémoire centrale permet des accès très rapides à l'information qu'elle contient. Mais son coût est élevé ; cela est dû à la technologie utilisée qui doit permettre d'accéder directement à un octet d'adresse quelconque. En outre, elle est généralement « volatile », c'est-à-dire que sa mise hors tension provoque la disparition de la totalité de l'information qu'elle contient.

Les périphériques d'archivage pallient ces difficultés en fournissant à la fois des mémoires permanentes et des coûts beaucoup plus faibles. En contrepartie, l'accès à l'information y est beaucoup plus lent. Deux grandes catégories de périphériques d'archivage sont en concurrence :

- Les périphériques ne permettant qu'un accès séquentiel à l'information : bandes ou cassettes magnétiques ; pour accéder à un octet quelconque, il est nécessaire de parcourir toute l'information qui le précède, à l'image de ce que vous faites avec un lecteur de cassettes audio ou vidéo.
- Les périphériques permettant l'accès direct à l'information : disques magnétiques (dits souvent « disques durs »), disquettes, CD-Rom, DVD... Ils permettent d'accéder presque directement à un octet, comme sur un lecteur de CD audio ; plus précisément, l'information y est rangée suivant des pistes concentriques et un mécanisme permet d'accéder mécaniquement à l'une quelconque de ces pistes : la lecture de cette piste est ensuite séquentielle (en toute rigueur, l'organisation des CD-Rom et des DVD est sensiblement différente de celle des disques magnétiques, mais celà n'a guère d'incidence sur leurs possibilités d'accès direct).

La première catégorie de périphériques ne doit sa survie qu'à son coût moins élevé que la seconde. Actuellement, elle reste surtout utilisée avec les très gros calculateurs.

4 Le langage de l'ordinateur

4.1 Langage machine ou langage de notre cru

Comme nous l'avons vu, l'ordinateur ne sait exécuter qu'un nombre limité d'opérations élémentaires, dictées par des instructions de programme et codées en binaire. On traduit cela en disant que l'ordinateur ne « comprend » que le langage machine.

Mais, fort heureusement, cela ne signifie nullement que tout programme doive être réalisé dans ce langage machine. En effet, et c'est là qu'intervient la seconde idée fondamentale de l'informatique (après celle de programme enregistré), à savoir : employer l'ordinateur lui-même (ou, plus précisément, un programme) pour effectuer la traduction du langage utilisé dans celui de l'ordinateur.

Nous ne pouvons pas pour autant utiliser n'importe quel langage de notre choix. En effet, il ne suffit pas de définir un langage, il faut qu'il puisse être traduit en langage machine, ce qui lui impose nécessairement d'importantes contraintes : un langage naturel comme le français ne pourrait pas convenir . En outre, il faut que le programme de traduction existe réellement. Tout ceci explique qu'à l'heure actuelle on doive se restreindre à des langages ayant un nombre très limité de mots, avec des règles de syntaxe assez rigoureuses.

4.2 En langage assembleur

Supposons, de façon un peu simplifiée, que l'on soit en présence d'un ordinateur pour lequel l'instruction machine :

```
0101010011011010
```

signifie : additionner (code opération 0101) les valeurs situées aux adresses 010011 et 011010.

Nous pouvons choisir d'exprimer cela sous une forme un peu plus parlante, par exemple :

```
ADD A, B
```

Pour que la chose soit réalisable, il suffit de disposer d'un programme capable de convertir le symbole `ADD` en 0101 et de remplacer les symboles `A` et `B` par des adresses binaires (ici 010011 et 011010).

Sans entrer dans le détail des tâches précises que doit réaliser un tel programme, on voit bien :

- qu'il doit faire correspondre un code opération à un symbole mnémonique ;
- qu'il doit être capable de décider des adresses à attribuer à chacun des symboles tels que `A` et `B` ; notamment, à la première rencontre d'un nouveau symbole, il doit lui attribuer une adresse parmi les emplacements disponibles (qu'il lui faut gérer) ; à une rencontre ultérieure de ce même symbole, il doit retrouver l'adresse qu'il lui a attribuée.

Tous les constructeurs sont en mesure de fournir avec leur ordinateur un programme capable de traduire un langage du type de celui que nous venons d'évoquer. Un tel langage se nomme langage d'assemblage ou encore assembleur. Le programme de traduction correspondant se nomme, lui aussi, assembleur.

Bien qu'ils se ressemblent, tous les ordinateurs n'ont pas exactement le même répertoire d'instructions machine. Dans ces conditions, chaque modèle d'ordinateur possède son propre assembleur. De plus, même si un tel langage est plus facile à manipuler que le langage machine, il ne s'en distingue que par son caractère symbolique, pour ne pas dire mnémonique. Les deux langages (assembleur et langage machine) possèdent pratiquement les mêmes instructions ; seule diffère la façon de les exprimer. Dans tous les cas, l'emploi de l'assembleur nécessite une bonne connaissance du fonctionnement de l'ordinateur utilisé. On exprime souvent cela en disant que ce langage est orienté machine. Réaliser un programme dans ce langage nécessite de penser davantage à la machine qu'au problème à résoudre.

4.3 En langage évolué

Très vite est apparu l'intérêt de définir des langages généraux utilisables sur n'importe quel ordinateur et orientés problème, autrement dit permettant aux utilisateurs de penser davantage à leur problème qu'à la machine.

C'est ainsi que sont apparus de très nombreux langages que l'on a qualifiés d'évolués. La plupart sont tombés dans l'oubli mais quelques-uns sont passés à la postérité : Fortran, Basic, Cobol, Pascal, ADA, C, Visual Basic, Delphi, C++, Java, C#, PHP, Python...

Dès maintenant, vous pouvez percevoir l'intérêt d'un langage évolué en examinant l'intruction suivante (elle se présentera textuellement sous cette forme dans la plupart des langages) :

```
Y = A * X + 2 * B + C
```

Sa signification est quasi évidente : à partir des valeurs contenues dans les emplacements nommés `A`, `X`, `B` et `C`, calculer la valeur de l'expression arithmétique `A*X + 2*B + C` (le symbole ***** représente une multiplication), puis ranger le résultat dans l'emplacement nommé `Y`.

Comme vous pouvez vous en douter, le même travail demanderait bon nombre d'opérations en langage machine (ou en assembleur), par exemple : prélever la valeur de `A`, la multiplier par celle de `X`, ranger le résultat dans un emplacement provisoire, prélever la valeur de `B`, la multiplier par 2, ajouter la valeur provisoire précédente, ajouter la valeur de `C` et, enfin, ranger le résultat final en `Y`.

Bien entendu, quel que soit le langage évolué utilisé, il est nécessaire, là encore, d'en réaliser, par programme, la traduction en langage machine. Pour cela, il existe deux techniques principales : la compilation et l'interprétation**.**

La compilation consiste à traduire globalement l'ensemble du programme en langage évolué (qu'on nomme souvent programme source) en un programme en langage mahine (qu'on nomme souvent programme objet), en utilisant un programme nommé compilateur. Si cette traduction s'est déroulée sans erreur, le programme objet peut être exécuté, en le plaçant en mémoire, autant de fois qu'on le désire, sans avoir besoin de recourir à nouveau au compilateur. De nombreux programmes sont fournis sous cette forme objet (dite aussi compilée) avec la plupart des micro-ordinateurs du commerce.

L'interprétation consiste à traduire chaque instruction du programme source, avant de l'exécuter, à l'aide d'un programme nommé interpréteur. Dans ce cas, il n'existe plus de programme objet complet et, à un instant donné, on trouve en mémoire, à la fois le programme source et le programme interpréteur.

On notera bien que le compilateur, comme l'interpréteur dépendent, non seulement du langage concerné, mais également du type d'ordinateur pour lequel on effectue la traduction.

Par ailleurs, il existe une technique intermédiaire entre compilation et interprétation qui consiste à traduire globalement un programme source (compilation) en un langage intermédiaire défini comme étant commun à un ensemble de machines, et à interpréter le résultat à l'aide d'un programme approprié. Cette technique avait été employée avec Pascal et elle l'est actuellement avec Java et C#. En toute rigueur, cette technique est très proche de la compilation, dans la mesure où tout se passe comme si le langage intermédiaire en question était en fait une sorte de langage machine unviersel. L'interprétation finale ne sert qu'à l'adapter à la machine concernée au moment de l'exécution.

5 Les concepts de base des langages évolués

Malgré leur multitude, la plupart des langages de programmation se basent sur un bon nombre de principes fondamentaux communs.

Certains découlent immédiatement de la nature même de l'ordinateur et de l'existence d'un programme de traduction. C'est, par exemple, le cas de la notion de variable que nous avons rencontrée sans la nommer : elle consiste à donner un nom à un emplacement de la mémoire destiné à contenir une information ; elle est donc liée à la fois à la notion technologique d'adresse et à l'existence d'un compilateur. Nous verrons que le besoin de traduire un programme en langage évolué nécessitera de définir la notion de type d'une variable, type qui sert à définir la manière dont doit s'opérer le codage des valeurs correspondantes.

De même, tout langage possède :

- des instructions dites d'affectation : analogues à celle présentée dans le paragraphe 4.3, page 9, elles permettent de calculer la valeur d'une expression et de la ranger dans une variable ;
- des instructions permettant d'échanger des informations entre la mémoire et des périphériques (qu'ils soient de communication ou d'archivage) ; on parle d'instructions :
  - de lecture, lorsque l'échange a lieu du périphérique vers la mémoire ;
  - d'écriture, lorsque l'échange a lieu de la mémoire vers le périphérique.

D'autres concepts, plus théoriques, ont été inventés par l'homme pour faciliter l'activité de programmation. C'est notamment le cas de ce que l'on nomme les structures de contrôle, les structures de données, les fonctions (ou procédures) et, plus récemment, les objets.

Les structures de contrôle servent à préciser comment doivent s'enchaîner les instructions d'un programme. En particulier, elles permettent d'exprimer les répétitions et les choix que nous avons déjà mentionnés : on parle alors de structure de choix ou de structure de répétition. Bien entendu, au bout du compte, après traduction du programme, ces structures se ramènent à des instructions machine et elles font finalement intervenir des instructions de branchement.

Les structures de données (attention, ici, le mot donnée est employé au sens général d'information) servent à mieux représenter les informations qui doivent être manipulées par un programme. C'est le cas de la notion de tableau dans laquelle un seul nom permet de désigner une liste ordonnée de valeurs, chaque valeur étant repérée par un numéro nommé indice. Bien entendu, là encore, au bout du compte, à chaque valeur correspondra un emplacement défini par son adresse.

La fonction (ou procédure) permet de donner un nom à un ensemble d'instructions qu'il devient possible d'utiliser à volonté, sans avoir à les écrire plusieurs fois. Comme dans le cas d'une fonction mathématique, ces instructions peuvent être paramétrées, de façon à pouvoir être utilisées à différentes reprises avec des variables différentes, nommées paramètres. Le bon usage des fonctions permet de structurer un programme en le décomposant en différentes unités relativement indépendantes.

Les notions d'objet et de classe sont les piliers de la programmation orientée objet. Un même objet regroupe, à la fois des données et des fonctions (nommées alors méthodes) ; seules ces méthodes sont habilitées à accéder aux données de l'objet concerné. La classe généralise aux objets la notion de type des variables. Elle définit les caractéristiques d'objets disposant de la même structure de données et des mêmes méthodes. Cette notion de classe offre une nouvelle possibilité de décomposition et de structuration des programmes. Elle sera complétée par les notions :

- d'héritage : possibilité d'exploiter une classe existante en lui ajoutant de nouvelles fonctionnalités) ;
- de polymorphisme : possibilité de s'adresser à un objet sans en connaître exactement la nature, en le laissant adapter son comportement à sa nature véritable.

On parle généralement de langage procédural pour qualifier un langage disposant de la notion de procédure (fonction), ce qui est le cas de tous les langages actuels. On parle souvent de langage objet pour qualifier un langage qui, en plus de l'aspect procédural, dispose de possibilités orientées objets. En toute rigueur, certains langages objet ne disposent pas de la fonction « usuelle », les seules fonctions existantes étant les méthodes des objets. De tels langages sont souvent qualifiés de totalement orientés objets. Ils sont cependant assez rares et, de toute façon, en dehors de cette différence, ils utilisent les mêmes concepts fondamentaux que les autres langages. Les autres langages objet permettent de faire cohabiter la décomposition procédurale avec la décomposition objet.

Dans la suite de l'ouvrage, nous étudierons d'abord les notions communes aux langages procéduraux, avant d'aborder les concepts objet. Cette démarche nous semble justifiée par le fait que la programmation orientée objet s'appuie sur les concepts procéduraux (même la notion de méthode reste très proche de celle de fonction).

Remarque

Initialement, les langages ne comportaient pas de structures de contrôle, mais seulement des instructions de branchement conditionnel ou non (nommées souvent `goto`). Puis sont apparues les structures de contrôle et l'on a alors parlé de programmation structurée. Par la suite, on a utilisé comme synonymes les termes programmation procédurale et programmation structurée, alors que, en toute rigueur, ils ne correspondaient pas au même concept. Quoi qu'il en soit, il n'existe plus de langages disposant de la notion de procédure et recourant encore de façon systématique aux instructions de branchement, de sorte que la distinction n'a dorénavant plus d'importance.

6 La programmation

L'activité de programmation consiste, au bout du compte, à réaliser un programme (ou une partie de programme, par exemple une fonction) résolvant un problème donné ou satisfaisant

à un besoin donné. Compte tenu de la multiplicité des langages existants, il existe différentes façons d'aborder cette activité.

Une première démarche consiste à étudier la syntaxe précise d'un langage donné puis à utiliser ce langage pour écrire le programme voulu. Cela laisse supposer alors qu'il existe autant de façon de résoudre le problème qu'il existe de langages différents.

Une autre démarche consiste à exploiter le fait que la plupart des langages se fondent sur des principes communs tels que ceux que nous venons d'évoquer et que l'on peut alors utiliser pour résoudre le problème donné. Encore faut-il disposer d'un moyen d'exprimer ces concepts. C'est précisément ce que nous vous proposons dans la suite de l'ouvrage, par le biais de ce que nous nommerons un **pseudo-langage** (certains parlent de langage algorithmique), lequel nous permettra d'utiliser les concepts fondamentaux pour rédiger (sur papier) de véritables programmes qu'il vous sera ensuite facile de transposer dans la plupart des langages actuels. Nous vous montrerons d'ailleurs comment s'expriment ces concepts fondamentaux dans des langages répandus (C, C++, C#, Java, PHP) et nous fournirons quelques exemples de programmes.

Remarque

Lorsqu'il s'agit de développer de gros programmes, il peut s'avérer nécessaire de recourir à des méthodes d'analyse plus abstraites, en s'éloignant des concepts fondamentaux des langages. Il n'en reste pas moins que l'emploi de telles méthodes sera plus efficace si l'on maîtrise les concepts de base de programmation.

7 Notion de système d'exploitation et d'environnement de programmation

Pour utiliser un programme, quel qu'il soit, il doit être présent en mémoire centrale. Mais, a priori, les programmes, comme les données, sont conservés sur un périphérique d'archivage tel le disque dur (rappelons que la mémoire centrale est de taille limitée et, surtout, volatile).

Pour amener un programme en mémoire centrale, on fait appel à ce que l'on nomme le système d'exploitation. Ce dernier n'est rien d'autre qu'un ensemble de programmes, stockés sur le disque, dont une partie (dite souvent résidante) est chargée automatiquement en mémoire au démarrage de votre ordinateur. Cette partie résidante vous permet de dialoguer avec votre ordinateur, à l'aide du clavier et de la souris, en gérant convenablement vos demandes. Notamment, elle vous permet d'assurer la bonne gestion de vos données (gestion des répertoires, suppression de fichiers, déplacement ou copie de fichiers...). Et, bien entendu, elle vous permet également de lancer un programme, qu'il s'agisse d'une connection Internet, d'une lecture d'un DVD, d'un compilateur ou... de votre propre programme traduit en langage machine.

On notera que lorsque vous réalisez un programme dans un langage donné, vous utilisez un certain type d'ordinateur, un certain système d'exploitation (il peut en exister plusieurs pour un même modèle d'ordinateur), un compilateur donné (il peut en exister plusieurs pour un même langage utilisé sur un même ordinateur, avec un même système). Il est également fréquent que vous recouriez à ce que l'on nomme un environnement de développement intégré, c'est-à-dire un logiciel qui vous facilite l'écriture et la mise au point d'un programme à l'aide d'outils plus ou moins sophistiqués : éditeur syntaxique qui vous permet de saisir le texte de votre programme en mettant en évidence sa structure ; débogueur qui vous permet de suivre pas à pas le déroulement de votre programme en visualisant des valeurs de variables...

Il nous arrivera de parler d'environnement de programmation pour désigner cet ensemble formé de l'ordinateur, du système, du langage et du compilateur utilisés (ou de l'environnement de développement intégré choisi). Comme nous le verrons, cet environnement de programmation pourra avoir un légère incidence sur le fonctionnement d'un programme, notamment dans les situations d'erreur.