Introduction

Ce livre a pour objectif de proposer une approche pédagogique de l'algorithmique. Il est structuré en deux grandes parties, la conception d'algorithmes et l'étude d'algorithmes existants.

La première partie traite de l'algorithmique et de l'analyse des données. Elle montre comment créer ses propres algorithmes, et présente la gestion de structures complexes comme les listes chaînées et les tableaux d'enregistrements ou d'objets qui servent de support à de nombreux algorithmes. Elle montre comment, à partir de l'analyse d'un problème, aboutir à un algorithme utilisant des mécanismes comme les boucles. Cette partie correspond à de l'algorithmique élémentaire. Elle permet d'assimiler les méthodes usuelles dans la conception logique des programmes, utiles à la compréhension d'algorithmes plus complexes.

La seconde partie présente des algorithmes connus. Elle en analyse le fonctionnement et détaille, pour chacun d'eux, les processus logiques utilisés. On y trouve, par exemple, des algorithmes de tri et de recherche.

Les exercices permettent au lecteur de mettre en pratique les notions présentées. Les algorithmes ainsi que les programmes C, C++ et Java associés sont fournis sur le site http://compagnons.pearson.fr/algorithmique, ce qui facilite le travail d'apprentissage en épargnant la saisie.

Qu'est-ce qu'un algorithme ?

Un algorithme est une *méthode logique de résolution d'un problème donné*, en vue d'être développé dans un langage de programmation. C'est la suite logique des actions à entreprendre pour aboutir à la solution finale, c'est donc… *un programme*.

Ce qui différencie un algorithme d'un programme écrit dans un langage de programmation est l'aspect *logique*. En algorithmique, le développeur se concentre sur la logique d'action pour produire un programme qui fournit une solution, en essayant de minimiser le temps de calcul et/ou l'espace mémoire occupé, sans se soucier des contraintes techniques de l'écriture du programme. La logique et l'efficacité du programme sont privilégiées. L'algorithme utilise souvent un langage simplifié (appelé pseudo-langage) pour décrire la méthode qui aboutit à la solution, et présenter la logique d'action. L'intérêt principal du pseudo-langage est de s'affranchir des contraintes techniques liées à un langage de programmation particulier. À l'inverse, ce n'est qu'un langage théorique, inventé pour les besoins de l'écriture de l'algorithme, qui ne peut pas être compilé directement.

L'étape finale du développement d'une application consiste à réécrire l'algorithme dans un langage de programmation pour produire un programme exécutable. Le travail porte alors sur la traduction de l'algorithme dans le langage choisi. Cette étape n'est que technique, car la logique du programme est déjà décrite dans l'algorithme.

L'algorithme est donc primordial, car ce sont les solutions qu'il propose qui seront programmées. La qualité et la rapidité du programme final découlent directement de l'algorithme.

Pseudo-langage et langages C, C++ ou Java comme support de développement

Le choix du pseudo-langage ou d'un langage de programmation particulier pour écrire l'algorithme est un débat qui n'est toujours pas tranché. Les partisans du pseudo-langage mettent en avant que la priorité de l'algorithme est de présenter la méthode qui aboutit à la solution, et qu'un langage de programmation risque de noyer le lecteur dans des détails techniques, et de spécialiser la solution selon les contraintes du langage choisi. Les adeptes du langage de programmation argumentent qu'un programme écrit en pseudo-langage ne peut être ni compilé, ni exécuté, ce qui lui enlève tout intérêt puisque l'utilisateur ne peut pas l'évaluer réellement. Nous pensons que c'est un faux débat, et que les deux méthodes sont complémentaires. Il est souhaitable de présenter la logique sous-jacente à l'algorithme en pseudo-langage pour se concentrer sur l'essentiel, et indispensable d'en proposer une version finale en langage de programmation pour permettre de le tester. Avec ces deux versions, le lecteur peut différencier la logique du programme de son aspect technique, ce qui lui laisse la liberté de le réécrire dans un autre langage que ceux qui sont proposés dans cet ouvrage.

De plus, un algorithme présenté uniquement en pseudo-langage revêt souvent une connotation théorique pour la plupart des étudiants qui ne perçoivent pas toujours son utilité, et surtout qui ne savent pas le traduire dans un langage de programmation. Il est donc important de présenter l'algorithme réécrit dans un langage de programmation usuel, sous la forme d'un programme qui peut être compilé et exécuté.

Cet ouvrage utilise les langages C, C++ et Java comme langages de développement. Ces langages présentent beaucoup d'avantages. Ils sont à la fois évolués et structurés, ce qui en font d'excellents supports pour l'apprentissage de la programmation en général, mais ils sont également très utilisés dans le monde industriel, scientifique ou Internet.

Les langages C++ et Java sont des langages objet. Ils permettent d'écrire plus facilement certains algorithmes qui portent sur des données structurées possédant leurs propres méthodes de traitement, comme les files, les piles ou les arbres. Cependant, cet ouvrage d'algorithmique n'a pas vocation à être spécifique à la programmation objet, les algorithmes présentés en pseudo-langage utilisent une programmation classique pour privilégier la logique du traitement, seuls les programmes sources C++ et Java en présentent une version objet.

Enfin, le langage C a fortement inspiré les concepteurs d'autres langages comme le PHP. Les techniques utilisées pour réécrire les algorithmes en langage C peuvent facilement être adaptées aux autres langages.

Malheureusement il serait fastidieux de présenter in-extenso les programmes sources des trois langages pour chaque algorithme décrit dans cet ouvrage. Les programmes sources sont donc téléchargeables sur le site des éditions Pearson. Nous présentons l'algorithme en pseudo-code ainsi qu'un exemple de compilation et d'exécution du programme source équivalent écrit en C, C++ ou Java.

Le fond et la forme

Cet ouvrage est conçu pour donner des bases d'algorithmique avant d'aborder des algorithmes plus complexes.

Chaque chapitre est divisé en deux parties, *Théorie* et *Exercices*. La partie théorique présente les notions algorithmiques et l'analyse des données. Elle comprend des figures et des encadrés qui insistent sur les points importants, et s'appuie sur des exemples. Elle se termine par un résumé qui rappelle les notions importantes étudiées. La partie *Exercices* reprend les notions présentées dans la partie *Théorie*. Les corrigés des exercices ainsi que des exercices complémentaires sont téléchargeables sur le site http://compagnons.pearson.fr/algorithmique.

Les exercices sont très importants. Leur rôle est d'asseoir l'apprentissage de l'algorithmique sur la pratique. Chaque développeur possède sa propre logique de programmation. Faire les exercices permet à chacun de trouver *sa* solution. Les solutions commentées aident le lecteur dans cette démarche.

Les programmes sources téléchargeables sont développés en respectant les langages C, C++ ou Java normalisés. Ils sont donc exploitables dans n'importe quel environnement de programmation. Le but recherché est de proposer des programmes facilement lisibles. Par conséquent, ils n'intègrent pas certaines astuces de programmation particulières à chaque langage, qui permettent de faire des codes plus courts. Les programmeurs avertis pourront donc encore améliorer ces programmes sources.

Prérequis

Cet ouvrage est conçu pour permettre l'écriture des algorithmes présentés dans n'importe quel langage de programmation. Cependant, le choix des langages C, C++ et Java comme supports de développement du programme final suppose la maîtrise d'un de ces langages de la part du lecteur, pour comprendre les codes sources (aucune description technique de ces langages n'est présentée dans ce livre).

Conventions

L'emploi de polices différentes permet de distinguer les syntaxes écrites en pseudo-langage, ou bien de faire ressortir un nouveau terme ou un concept important.

- Les instructions en pseudo-langage sont dans une police à espacement fixe, par exemple `tab[i]`.
- Les termes nouveaux ou importants sont en italique, par exemple *modélisation des données*.
- Dans les programmes donnés en exemples ou en exercices, les syntaxes nouvelles ou importantes sont en police à espacement fixe et en gras, comme la syntaxe suivante : **`Lire(article.prix)`**.
- Les exemples d'exécution des programmes C, C++ et Java présentent la saisie en gras pour la différencier de l'affichage.

Le contenu

La première partie de l'ouvrage traite de la conception d'algorithmes. Elle englobe les chapitres 1, 2 et 3.

Chapitre 1 : Environnement algorithmique et conventions. Il détaille le rôle de l'algorithme dans le développement des applications, et les règles à respecter pour son écriture.

Chapitre 2 : Les traitements logiques. Il présente les grands mécanismes utilisés dans les algorithmes comme les tests, les boucles et les sous-programmes. Il présente également la notion de complexité algorithmique.

Chapitre 3 : La gestion des données. Ce chapitre détaille les différents types de données qui servent de support aux algorithmes, comme les tableaux ou les listes chaînées.

La seconde partie de l'ouvrage, qui présente des algorithmes importants ou des méthodes algorithmiques spécifiques, est constituée des chapitres 4, 5, 6, 7, 8 et 9.

Chapitre 4 : La récursivité. Ce chapitre détaille le fonctionnement de la programmation récursive, et en présente l'intérêt.

Chapitre 5 : Les données abstraites. Ce chapitre présente les piles, les files et les arbres qui sont utilisés dans de nombreux algorithmes.

Chapitre 6 : Les tris. Ce chapitre porte sur les algorithmes de tri, qui sont répartis en tris élémentaires et en tris avancés.

Chapitre 7 : Les recherches. Ce chapitre présente quelques algorithmes de recherche. Cela va de la recherche séquentielle aux algorithmes plus sophistiqués, comme les tables de hachage.

Chapitre 8 : Les méthodes numériques. Ce chapitre présente, la mise en œuvre d'algorithmes mathématiques, tels que la recherche des racines d'une équation ou l'interpolation polynomiale.

Chapitre 9 : Les algorithmes classiques. Ce chapitre présente quelques algorithmes incontournables.

Les suppléments

Les programmes des exercices et des exemples au format source « .c », « .cpp » ou « .java » sont disponibles au téléchargement sur le site : http://compagnons.pearson.fr/algorithmique. Le lecteur pourra les utiliser pour son apprentissage, mais aussi les compléter et se constituer ainsi sa propre base d'exemples. Le site contient également les corrigés des exercices du livre et des exercices supplémentaires entièrement corrigés. Pour y accéder, le mot de passe est : **tdqKvB569Exn**

Remerciements

Je tiens à remercier Frédéric Jacquenod pour sa relecture avisée. Une attention toute particulière pour Sylvie, Guillaume et Julien qui ont fait preuve de beaucoup de patience.

1

Environnement algorithmique et conventions

Un algorithme est un maillon de la chaîne de développement d'une application. Il est le lien indispensable entre l'analyse et le développement final. Ce chapitre présente le rôle d'un algorithme dans cette chaîne de production, et les conventions syntaxiques utilisées pour son écriture. Il aborde également les notions de performances et de génie logiciel, qui servent souvent de guides dans la phase de conception algorithmique.

1. Les étapes de développement d'une application

L'écriture d'un programme dans un langage de programmation n'est que l'étape finale d'un développement qui se déroule en trois phases : *l'analyse*, *l'algorithme* et *la programmation*.

1.1. L'analyse

Le but de tout développement est de fournir une solution informatique à un problème donné. La première étape est *l'analyse du problème*. Elle consiste à explorer le domaine d'application pour en déduire les contraintes et les limites de l'étude, à définir les principales fonctionnalités qui devront être couvertes par le logiciel, à mettre en évidence les problèmes connexes, à faire l'inventaire des acteurs (utilisateurs, gestionnaires, etc.) et à en déduire l'interface d'exploitation du produit, etc. L'analyse tend à cerner complètement le sujet et à apporter des solutions aux problèmes dévoilés par l'étude.

L'analyse se termine généralement sur des considérations plus techniques qui font le lien avec l'algorithme. Il s'agit de l'*analyse des données* qui aboutit à leur *modélisation*, de la définition des *modules de traitement*, ainsi que leurs interactions

réciproques qui débouchent sur la création des sous-programmes. L'analyse conditionne le développement futur, car ce sont les solutions proposées à cette étape qui seront programmées. Une mauvaise analyse produira un logiciel médiocre ou qui ne répondra pas aux attentes initiales, quelle que soit la qualité de l'algorithme ou de la programmation.

Voici deux exemples d'analyses partielles. Le premier montre l'incidence de l'étude du domaine d'application sur le résultat de l'analyse. Le second propose une analyse des données et montre comment les structurer en fonction des traitements à effectuer.

L'analyse du premier exemple va évoluer au fur et à mesure que les bonnes questions seront soulevées. Elle produira au final un programme très différent dans sa complexité. Prenons le problème de l'interprétation du numéro de Sécurité sociale (sans la clé finale). La signification des champs de ce numéro est indiquée à la figure 1.1.

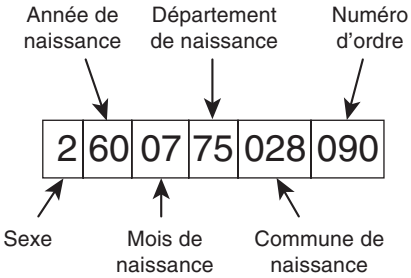

Figure 1.1 • L'interprétation d'un numéro de Sécurité sociale.

Une première analyse de ce numéro montre qu'il s'agit d'une *femme*, née en *1960*, au mois de *juillet*, à *Paris*. Or, l'interprétation du quatrième élément de cet exemple est fausse ! Cette personne est née dans le département de la Seine qui, en 1960, regroupait Paris et les départements limitrophes (Val-de-Marne, Val d'Oise, etc.). Cette remarque soulève d'autres questions qui vont compliquer l'analyse, en intégrant un volet historique :

- Comment le découpage des départements a-t-il évolué (Île-de-France, Corse, Outre-Mer, etc.) ?
- Qu'en est-il des personnes nées dans les colonies françaises avant leur indépendance ?
- Comment le problème des centenaires est-il réglé ?

L'analyse simpliste du début aurait conduit à écrire un programme ne traitant qu'une partie de la problématique. Le problème est en réalité beaucoup plus complexe qu'il n'y paraît. L'erreur qu'il ne faut pas faire pendant la phase d'analyse est d'apporter des solutions basées sur ses certitudes, au lieu de se poser d'abord des questions.

Le second exemple montre comment, à partir de l'analyse des données, on peut aboutir à leur modélisation. L'exemple étudié est celui d'une course hippique. La première question à se poser est « qu'est-ce qu'un cheval ? ». La réponse n'est pas liée à la nature

de l'animal, mais aux actions à entreprendre pour gérer une course. On peut les résumer à :

- indiquer la liste des chevaux au départ de la course ;
- affecter le temps de chaque cheval à la fin de la course ;
- afficher l'ordre d'arrivée.

On peut déduire qu'une donnée représentant un cheval doit contenir :

- le nom du cheval ;
- le numéro du dossard ;
- le temps d'arrivée ;
- le classement.

On peut également ajouter le nom du jockey, celui du propriétaire, et le montant des gains si on désire ajouter un volet financier à la gestion de la course.

La figure 1.2 présente une modélisation des données de la course hippique. Les chevaux sont gérés comme une liste, où chaque élément est une structure contenant le nom du cheval, le numéro du dossard, le temps effectué durant la course et le classement final. La liste peut être représentée sous la forme d'un tableau, où chaque case contient toute l'information d'un cheval, ou bien sous la forme d'éléments distincts liés entre eux par la connaissance de l'élément précédent et suivant (liste chaînée de structures ou d'objets). La modélisation des données est ensuite suivie par la description de leurs traitements, ce qui est le but de l'algorithme.

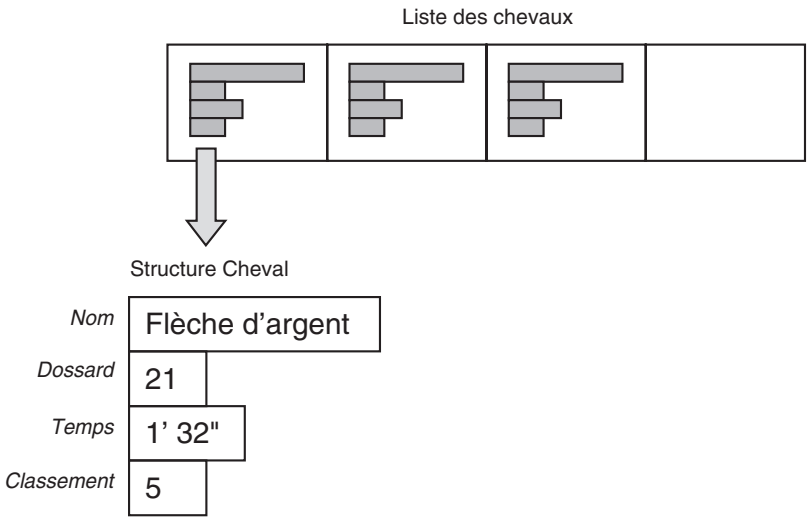

Figure 1.2 • Modélisation des données d'une course hippique.

1.2. L'algorithme

Le rôle de l'algorithme est avant tout d'organiser et de structurer l'ensemble des solutions apportées par l'analyse pour aboutir à la solution globale. Il doit surtout présenter la logique aboutissant au résultat final et être dissocié de l'aspect technique des langages informatiques. Il est généralement présenté en pseudo-langage (langage simplifié), ce qui facilite sa traduction dans un langage de programmation. Voici l'algorithme interprétant le numéro de Sécurité sociale, correspondant à une analyse ne traitant pas l'aspect historique.

« Pseudo-code »

```
Programme  Num_Sec_Soc
Déclarations
   Variables  Num_Sec, Libellé, Nom_Mois  en Chaînes_de_Caractères
   Variables  Sexe, Année, Mois, Département, Commune, Num_Ordre  en Entier
Début
  Écrire ("Entrez votre numéro de Sécurité sociale :")
  Lire(Num_Sec)
  Sexe ← Conversion_En_Entier(sous_chaîne(Num_Sec,1,1)
  Si (Sexe = 1) alors
     Libellé ← "Monsieur"
  Sinon
     Libellé ← "Madame"
  Finsi
  Année ← Conversion_En_Entier(sous_chaîne(Num_Sec,2,2))
  Année ← Année + 1900
  Mois ← Conversion_En_Entier(sous_chaîne(Num_Sec,4,2))
  Selon  Mois  Faire
    Cas 1 : Nom_Mois ← "Janvier"
    Cas 2 : Nom_Mois ← "Février"
    Cas 3 : Nom_Mois ← "Mars"
    Cas 4 : Nom_Mois ← "Avril"
    Cas 5 : Nom_Mois ← "Mai"
    Cas 6 : Nom_Mois ← "Juin"
    Cas 7 : Nom_Mois ← "Juillet"
    Cas 8 : Nom_Mois ← "Août"
    Cas 9 : Nom_Mois ← "Septembre"
    Cas 10 : Nom_Mois ← "Octobre"
    Cas 11 : Nom_Mois ← "Novembre"
    Cas 12 : Nom_Mois ← "Décembre"
  FinSelon
  Département ← Conversion_En_Entier(sous_chaîne(Num_Sec,6,2))
  Commune ← Conversion_En_Entier(sous_chaîne(Num_Sec,6,5))
  Num-Ordre ← Conversion_En_Entier(sous_chaîne(Num_Sec,11,3))
  Écrire ("Bonjour : ", Libellé)
  Écrire ("Vous êtes né en : ", Année)
  Écrire ("Au mois de : ", Nom_Mois)
  Écrire ("Dans le département : ", Département)
  Écrire ("Dans la commune : ", Commune)
  Écrire ("Avec le numéro d'ordre : ", Num_Ordre)
Fin
```

La syntaxe du pseudo-langage est librement choisie par le développeur. Cependant, elle respecte certaines règles et conventions usuelles qui sont décrites dans ce chapitre.

1.3. L'écriture du programme dans un langage de programmation

Une fois l'algorithme écrit, il faut le traduire en un *programme source*. Cela consiste à écrire un fichier texte contenant la traduction de l'algorithme dans un langage de programmation. Le programme source est ensuite traduit en langage binaire du processeur grâce au compilateur. Le fichier résultat de cette opération est le *programme exécutable*.

Dans l'absolu, le développeur doit choisir le langage de programmation le plus adapté au domaine d'application et aux contraintes liées à l'algorithme. En effet, les langages de programmation ont été créés pour répondre à des besoins spécifiques. Par exemple, le langage Fortran est dédié au calcul scientifique, le langage C est polyvalent et il excelle dans les applications systèmes et réseaux, le langage C++ est très présent dans les applications temps réel, le langage Java est conçu pour les développements liés au Web, etc. En réalité, le développeur choisit le langage qu'il connaît le mieux ou, de manière plus pragmatique, pour lequel il possède un compilateur. Ce n'est donc pas toujours le langage le mieux adapté aux spécificités du développement qui est utilisé.

Afin de permettre au lecteur de tester les programmes sources C, C++ et Java qui sont téléchargeables sur le site de Pearson, nous présentons la réécriture des algorithmes de ce chapitre dans les trois langages, les syntaxes de compilation de chaque programme source ainsi qu'un exemple d'exécution. Par la suite, dans les autres chapitres, seuls les syntaxes de compilation et un exemple d'exécution seront proposés pour ne pas alourdir cet ouvrage par de nombreux codes sources. Cependant, quand la présentation du code source est nécessaire pour une parfaite compréhension de l'algorithme et de sa réécriture dans un langage de programmation, le programme C est alors présenté, en plus de l'algorithme en pseudo-langage.

Voici le programme source `numsecsoc.c` écrit en langage C qui correspond à l'algorithme du numéro de Sécurité sociale. La syntaxe du langage C n'admet aucun caractère accentué dans le nom des variables. La variable `Libellé` de l'algorithmique est ainsi écrite `Libelle` dans le programme C.

« C »

```
/* numsecsoc.c      */
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
main()
{/* --- déclarations des variables --- */
 char Num_Sec[14], Libelle[9], Nom_Mois[10], Dept[3], Com[6] ;
 int Sexe, Annee, Mois, Num_Ordre, Departement, Commune ;
 /* --- instructions --- */
```

```
/* saisie du numéro sous la forme d'une chaîne */
printf("Entrez votre numéro de Sécurité sociale : ") ;
scanf("%s",Num_Sec) ;
/* récupération du numéro 1 ou 2 du Sexe */
Sexe = Num_Sec[0]-48 ;
/* Affectation du Libellé */
if (Sexe == 1)
  strcpy(Libelle,"Monsieur") ;
else
  strcpy(Libelle,"Madame") ;
/* récupération de l'année de naissance */
Annee = ((Num_Sec[1]-48)*10) + (Num_Sec[2]-48) ;
Annee = Annee + 1900 ;
/* récupération du mois de naissance */
Mois = ((Num_Sec[3]-48)*10) + (Num_Sec[4]-48) ;
switch (Mois)
{
  case 1  : strcpy(Nom_Mois,"Janvier") ;
            break ;
  case 2  : strcpy(Nom_Mois,"Février") ;
            break ;
  case 3  : strcpy(Nom_Mois,"Mars") ;
            break ;
  case 4  : strcpy(Nom_Mois,"Avril") ;
            break ;
  case 5  : strcpy(Nom_Mois,"Mai") ;
            break ;
  case 6  : strcpy(Nom_Mois,"Juin") ;
            break ;
  case 7  : strcpy(Nom_Mois,"Juillet") ;
            break ;
  case 8  : strcpy(Nom_Mois,"Août") ;
            break ;
  case 9  : strcpy(Nom_Mois,"Septembre") ;
            break ;
  case 10 : strcpy(Nom_Mois,"Octobre") ;
            break ;
  case 11 : strcpy(Nom_Mois,"Novembre") ;
            break ;
  case 12 : strcpy(Nom_Mois,"Décembre") ;
            break ;
}
/* récupération du département  */
/* qui reste une chaîne         */
Dept[0]=Num_Sec[5] ;
Dept[1]=Num_Sec[6] ;
Dept[2]='\0'       ;
Departement=atoi(Dept) ;
/* récupération de la Commune */
Com[0]=Num_Sec[5] ;
Com[1]=Num_Sec[6] ;
Com[2]=Num_Sec[7] ;
Com[3]=Num_Sec[8] ;
```

```
 Com[4]=Num_Sec[9] ;
 Com[5]='\0'       ;
 Commune=atoi(Com) ;
 /* récupération du numéro d'ordre */
 Num_Ordre=((Num_Sec[10]-48)*100)+((Num_Sec[11]-48)*10)+(Num_Sec[12]-48);
 /* Affichage des résultats */
 printf("Bonjour                %s\n",Libelle)      ;
 printf("Vous êtes né en        : %d\n",Annee)      ;
 printf("Au mois de             : %s\n",Nom_Mois)   ;
 printf("Dans le département    : %d\n",Departement);
 printf("Dans la commune        : %d\n",Commune)    ;
 printf("Avec le numéro d'ordre : %d\n",Num_Ordre)  ;
}
```

La compilation de ce programme est obtenue par la commande suivante :

```
$ make numsecsoc
```

Voici l'exécution du logiciel (fichier exécutable) `numsecsoc` produit par la compilation précédente (le caractère `$` représente le prompt du système UNIX) :

```
$ numsecsoc
Entrez votre numéro de Sécurité sociale : 1650475110019
Bonjour                Monsieur
Vous êtes né en        : 1965
Au mois de             : Avril
Dans le département    : 75
Dans la commune        : 75110
Avec le numéro d'ordre : 19
```

Voici le programme source `numsecsoc.cpp` écrit en langage C++ qui correspond à l'algorithme du numéro de Sécurité sociale. Aucun caractère accentué n'est utilisé dans le nom des variables.

« C++ »

```
// ----------------
// numsecsoc.cpp
// ----------------
#include <cstdio>
#include <cstring>
#include <cstdlib>
int main()
{// --- déclarations des variables ---
 char Num_Sec[14], Libelle[9], Nom_Mois[10], Dept[3], Com[6];
 // --- instructions ---
 // saisie du numéro sous la forme d'une chaîne
 printf("Entrez votre numéro de Sécurité sociale : ") ;
 scanf("%s",Num_Sec) ;
 // récupération du numéro 1 ou 2 du Sexe
 int Sexe = Num_Sec[0]-48 ;
 // Affectation du Libellé
 if (Sexe == 1)
```

```
  strcpy(Libelle,"Monsieur") ;
else
  strcpy(Libelle,"Madame") ;
// récupération de l'année de naissance
int Annee = ((Num_Sec[1]-48)*10) + (Num_Sec[2]-48) ;
Annee = Annee + 1900 ;
// récupération du mois de naissance
int Mois = ((Num_Sec[3]-48)*10) + (Num_Sec[4]-48) ;
switch (Mois)
{
  case 1  : strcpy(Nom_Mois,"Janvier") ;
            break ;
  case 2  : strcpy(Nom_Mois,"Février") ;
            break ;
  case 3  : strcpy(Nom_Mois,"Mars") ;
            break ;
  case 4  : strcpy(Nom_Mois,"Avril") ;
            break ;
  case 5  : strcpy(Nom_Mois,"Mai") ;
            break ;
  case 6  : strcpy(Nom_Mois,"Juin") ;
            break ;
  case 7  : strcpy(Nom_Mois,"Juillet") ;
            break ;
  case 8  : strcpy(Nom_Mois,"Août") ;
            break ;
  case 9  : strcpy(Nom_Mois,"Septembre") ;
            break ;
  case 10 : strcpy(Nom_Mois,"Octobre") ;
            break ;
  case 11 : strcpy(Nom_Mois,"Novembre") ;
            break ;
  case 12 : strcpy(Nom_Mois,"Décembre") ;
            break ;
}
// récupération du département
// qui reste une chaîne
Dept[0]=Num_Sec[5] ;
Dept[1]=Num_Sec[6] ;
Dept[2]='\0'       ;
int Departement=atoi(Dept);
// récupération de la Commune
Com[0]=Num_Sec[5] ;
Com[1]=Num_Sec[6] ;
Com[2]=Num_Sec[7] ;
Com[3]=Num_Sec[8] ;
Com[4]=Num_Sec[9] ;
Com[5]='\0'       ;
int Commune=atoi(Com);
// récupération du numéro d'ordre
int Num_Ordre=((Num_Sec[10]-48)*100)+((Num_Sec[11]-48)*10)+(Num_Sec[12]-48);
// Affichage des résultats
printf("Bonjour                %s\n",Libelle)      ;
```

```
 printf("Vous êtes né en        : %d\n",Annee)      ;
 printf("Au mois de             : %s\n",Nom_Mois)   ;
 printf("Dans le département    : %d\n",Departement);
 printf("Dans la commune        : %d\n",Commune)    ;
 printf("Avec le numéro d'ordre : %d\n",Num_Ordre)  ;
}
```

La compilation de ce programme est obtenue par la commande suivante (le caractère `$` représente le prompt du système UNIX) :

```
$ make numsecsoc
```

Voici l'exécution du fichier exécutable `numsecsoc` produit par la compilation précédente :

```
$ numsecsoc
Entrez votre numéro de Sécurité sociale : 2621133028088
Bonjour                Madame
Vous êtes né en        : 1962
Au mois de             : Novembre
Dans le département    : 33
Dans la commune        : 33028
Avec le numéro d'ordre : 88
```

Voici le programme source `numsecsoc.java` écrit en langage Java qui correspond à l'algorithme du numéro de Sécurité sociale. Aucun caractère accentué n'est utilisé dans le nom des variables.

« Java »

```
// numsecsoc.java
import java.io.*;
class Num_Sec_Soc {
  private String ch ;

  public Num_Sec_Soc() {
   ch=Saisie.lire_String("Entrez votre numéro de Sécurité sociale : ") ;
  }
  public void interprete_sexe() {
   int sexe=Integer.parseInt(ch.substring(0,1));
   if (sexe == 1) System.out.println("Bonjour  Monsieur");
   else System.out.println("Bonjour  Madame");
  }
  public void interprete_annee() {
   int annee=Integer.parseInt(ch.substring(1,3))+1900;
   System.out.println("Vous êtes né en        : "+annee);
  }
  public void interprete_mois() {
   int mois=Integer.parseInt(ch.substring(3,5));
   String Nom_mois=new String("");
   switch (mois)
   { case 1  : Nom_mois="Janvier" ; break ;
     case 2  : Nom_mois="Février" ; break ;
     case 3  : Nom_mois="Mars" ; break ;
```

```
      case 4  : Nom_mois="Avril"; break ;
      case 5  : Nom_mois="Mai" ; break ;
      case 6  : Nom_mois="Juin" ; break ;
      case 7  : Nom_mois="Juillet" ; break ;
      case 8  : Nom_mois="Août" ; break ;
      case 9  : Nom_mois="Septembre" ; break ;
      case 10 : Nom_mois="Octobre" ; break ;
      case 11 : Nom_mois="Novembre" ; break ;
      case 12 : Nom_mois="Décembre" ; break ;
    }
    System.out.println("Au mois de             : "+Nom_mois) ;
   }
   public void interprete_departement() {
    int departement=Integer.parseInt(ch.substring(5,7));
    System.out.println("Dans le département    : "+departement) ;
   }
   public void interprete_commune() {
    int commune=Integer.parseInt(ch.substring(5,10));
    System.out.println("Dans la commune        : "+commune)    ;
   }
   public void interprete_ordre() {
    int ordre=Integer.parseInt(ch.substring(10));
    System.out.println("Avec le numéro d'ordre : "+ordre)  ;
   }
}
// programme principal
public class numsecsoc {
  public static void main(String[] args) {
   Num_Sec_Soc Numero = new Num_Sec_Soc() ;
   Numero.interprete_sexe();
   Numero.interprete_annee();
   Numero.interprete_mois();
   Numero.interprete_departement();
   Numero.interprete_commune();
   Numero.interprete_ordre();
  }
}
```

Ce programme utilise la méthode `lire_String` de la classe `Saisie` pour lire le numéro de Sécurité Sociale sous la forme d'une chaîne de caractères. En langage Java, la gestion des flux d'entrée/sortie est un peu complexe à écrire. Nous utilisons ici la classe `Saisie` qui fournit plusieurs méthodes pour lire au clavier différents types de données. Cette classe est proposée dans le livre sur le langage Java, intitulé *Java 7*, de Robert Chevallier dans la collection Synthex. Le fichier `Saisie.java` doit se trouver dans le même répertoire que le fichier source qui l'utilise. Il est téléchargeable sur le site de l'éditeur. En voici le détail :

« Java »

```
//---Saisie.java
import java.io.*;
```

```
class Saisie
{public static String lire_String()
    {String ligne_lue=null;
     try {InputStreamReader isr = new InputStreamReader(System.in);
          BufferedReader br = new BufferedReader(isr);
          ligne_lue=br.readLine();
                }
     catch(IOException e) {System.err.println(e);}
     return ligne_lue;
        }
 public static String lire_String(String question)
  {System.out.print(question);
   return(lire_String());}
 public static int lire_int()
    {return Integer.parseInt(lire_String());}
 public static int lire_int(String question)
    {System.out.print(question);
         return Integer.parseInt(lire_String());}
 public static double lire_double()
    {return Double.parseDouble(lire_String());}
 public static double lire_double(String question)
    {System.out.print(question);
         return Double.parseDouble(lire_String());}
 public static float lire_float()
    {return Float.parseFloat(lire_String());}
 public static float lire_float(String question)
    {System.out.print(question);
         return Float.parseFloat(lire_String());}
 public static char lire_char()
    {String reponse=lire_String();
         return reponse.charAt(0);}
 public static char lire_char(String question)
    {System.out.print(question);
         String reponse=lire_String();
         return reponse.charAt(0);}
}
```

La compilation de ce programme est obtenue par la commande suivante (le caractère `$` représente le prompt du système UNIX) :

```
$ javac numsecsoc.java
```

Les fichiers `.class` suivants sont produits :

```
$ ls *.class
numsecsoc.class  Num_Sec_Soc.class  Saisie.class
```

Voici l'interprétation du binaire `numsecsoc` produit par la compilation précédente :

```
$ java numsecsoc
Entrez votre numéro de Sécurité sociale : 1920275112851
Bonjour  Monsieur
Vous êtes né en        : 1992
Au mois de             : Février
```

```
Dans le département    : 75
Dans la commune        : 75112
Avec le numéro d'ordre : 851
```

2. Le pseudo-langage

Le pseudo-langage n'est pas un langage normalisé. Chaque développeur peut définir son propre langage et ses conventions syntaxiques. Il est important de préciser son rôle et les conventions habituellement rencontrées.

2.1. Son rôle

Le but de ce langage est de simplifier l'écriture du programme en s'affranchissant des contraintes syntaxiques liées aux langages de programmation, et d'en souligner sa logique. Si le pseudo-langage, par sa simplicité, aide à présenter l'algorithme, il sert aussi à formaliser plus clairement la structure du programme. Le pseudo-langage doit refléter, le plus possible, les instructions couramment rencontrées dans les différents langages de programmation. Sans être l'image d'un langage spécifique, il doit néanmoins être facilement traduisible dans différents langages, ce qui suppose l'utilisation d'une syntaxe proche de ce que les langages de programmation proposent généralement.

2.2. Conventions syntaxiques

Même si chaque développeur définit son propre pseudo-langage, certaines conventions sont couramment employées. Nous en détaillons quelques-unes, à partir de l'exemple présenté précédemment. Les autres conventions utilisées dans cet ouvrage seront décrites au fur et à mesure de leur emploi.

La structure du programme

Un programme est structuré en une partie *déclaration* et une partie *instruction*. Même si tous les langages ne proposent pas une structure rigoureuse, il est préférable de la présenter dans l'algorithme. Les mots clés sont « `Programme` » pour le début du programme, « `Déclarations` » pour la déclaration des variables, et « `Début` » puis « `Fin` » pour la partie instruction.

```
Programme  Num_Sec_Soc
Déclarations
   Variables  Num_Sec, Dépt, Libellé, Nom_Mois  en Chaînes_de_Caractères
   Variables  Sexe, Année, Mois, Commune, Num_Ordre  en Entier
Début
  Écrire ("Entrez votre numéro de Sécurité sociale :")
  ...
Fin
```