Préface

Depuis les années 1940, les programmes informatiques ont pris une place de plus en plus grande dans nos vies et leur qualité est devenue un enjeu majeur. Écrire un programme de qualité, c'est-à-dire un programme qui fait ce que l'on attend de lui, qui est efficace, lisible, robuste, facile à modifier et à étendre, etc., demande une grande exigence intellectuelle. Cela demande aussi de maîtriser des méthodes de développement rigoureuses. Mais cela demande, avant tout, une connaissance approfondie du langage de programmation que l'on utilise, de sa syntaxe bien entendu, mais surtout de sa sémantique, c'est-à-dire de ce qui se passe lorsqu'un programme est exécuté.

La description de cette sémantique fait émerger un certain nombre de concepts fondamentaux, tels ceux de valeur, de référence, d'exception ou d'objet qui constituent les briques de base de la théorie des langages de programmation. C'est la maîtrise de ces concepts qui distingue le programmeur aguerri du débutant. Certains concepts – tel celui de valeur – sont partagés par tous les langages de programmation, d'autres – tel celui de fonction – se déclinent de manière différente dans un langage et dans un autre, d'autres enfin – tel celui d'objet – n'existent que dans certains langages. Les informaticiens parlent souvent de paradigme de programmation pour désigner un ensemble de concepts partagés par une famille de langages, qui induit un style de programmation : impératif, fonctionnel, à objets, logique, concurrent, etc. Mais plus que les paradigmes, ce sont ces concepts eux-mêmes qu'il importe de connaître, car plusieurs paradigmes se mêlent souvent dans un même langage.

Il existe de nombreux ouvrages d'introduction à la programmation dans tel ou tel langage. Il en existe aussi quelques-uns qui présentent les concepts fondamentaux de la sémantique des langages. Beaucoup plus rares sont ceux qui, tel le livre que vous avez entre les mains, établissent un lien entre ces concepts et leur incarnation dans les langages utilisés quotidiennement par les programmeurs, tels C, C++, Ada, Java, OCaml ou Python. De nombreux exemples dans ces langages illustrent ainsi les notions introduites et leur donnent vie. En proposant des modèles généraux, comme par

exemple celui de *kit* dans le volume 2, les auteurs proposent une vue unificatrice de différentes notions permettant ainsi au lecteur d'appréhender les constructions des langages usuels et de pouvoir ainsi mieux les comparer. Ce livre très complet permet au lecteur non seulement de comprendre ces notions, mais surtout d'appréhender comment les utiliser pour écrire des programmes de qualité et contribuer ainsi à un monde dans lequel les objets informatiques sont plus sûrs et plus fiables.

Gilles DOWEK
Directeur de recherche Inria
Professeur à l'École normale supérieure de Paris-Saclay

Catherine DUBOIS
Professeur à l'École nationale supérieure
d'informatique pour l'industrie et l'entreprise

Avant-propos

Cet ouvrage, divisé en deux volumes, est dédié à la programmation. Il a d'abord pour objectif de permettre aux lecteurs d'acquérir non seulement les bases de la programmation de style fonctionnel ou impératif, mais aussi une bonne connaissance des mécanismes des modules et des classes. Nous estimons que la programmation doit être étudiée en privilégiant le point de vue sémantique, afin de passer facilement la barrière des différences de syntaxe entre langages de programmation. Nous sommes également soucieux de ne pas négliger les aspects pratiques découlant des caractéristiques matérielles des ordinateurs ou objets connectés. Ces deux phrases décrivent l'approche utilisée dans les deux volumes de l'ouvrage, où l'on trouvera à la fois des formules mathématiques et des schémas d'états de la mémoire. Nous espérons, avec cet ouvrage, aider le lecteur à comprendre la signification des constructions décrites dans le manuel de référence du langage de programmation de son choix, à fonder ses raisonnements, à juger de la correction de ses programmes et à en effectuer une relecture critique. En résumé, nous serions heureux de faciliter la tâche de développement de logiciels sûrs.

Le volume 1 débute par une présentation de l'ordinateur, faite dans le chapitre 1, d'abord au niveau matériel en tant qu'assemblage de composants puis au niveau logiciel en tant qu'outil d'exécution de programmes. Le chapitre 2 est une introduction intuitive, petits pas par petits pas, à la sémantique des langages, de manière à familiariser le lecteur avec cette approche de la programmation. Le chapitre 3 rentre dans le vif du sujet en présentant de manière formelle les sémantiques d'exécution des traits fonctionnels, suivi en cela par le chapitre 4 qui porte sur les sémantiques d'exécution des traits impératifs. Ces deux chapitres se placent résolument dans un cadre mathématique permettant d'étayer les propos. Pour aider à l'acquisition de ces concepts sémantiques, toutes les notions introduites ont été programmées à la fois dans les langages Python et OCaml et de nombreux exercices accompagnés de corrections détaillées ont été inclus dans ces chapitres. Le chapitre 5, dédié au typage, porte d'abord

sur les règles de typage permettant la vérification des programmes, puis expose l'algorithme d'inférence de types polymorphes et toutes les notions mathématiques afférentes, elles aussi programmées dans les deux langages. Il s'achève avec l'extension du typage aux traits impératifs. Dans le chapitre 6, sont présentés les principaux types de données ainsi que le filtrage sur ces données. Il contient de nombreux exemples exprimés dans des langages de programmation variés. Le chapitre 7 est consacré aux traits de bas niveau de la programmation : boutisme, pointeurs, allocation et libération de la mémoire, principalement illustrés avec les langages C et C++. Le volume 1 s'achève avec le traitement des erreurs à l'aide d'exceptions dans le chapitre 8, leur sémantique est étudiée avec le langage OCaml puis les mécanismes de gestion d'exceptions de Python, Java et C++ sont présentés.

Le volume 1 propose donc un vaste panorama des traits fonctionnels et impératifs de la programmation, depuis des notions ayant pu être mathématiquement modélisées jusqu'à des notions liées à la configuration matérielle des ordinateurs. Le volume 2 est dédié à la programmation modulaire et objet. Il s'appuie fortement sur le volume 1 puisque modules et objets ne font qu'organiser des constructions fonctionnelles ou impératives. Ce volume 2 présente d'abord un modèle sémantique original, appelé *kit*, présentant de manière conjointe l'ensemble des traits des modules et des objets. La sémantique de ces *kits* est définie de manière assez informelle, les recherches en ce domaine n'ayant pas encore abouti à un modèle mathématique de cet ensemble de traits tout en restant relativement simple. De ce modèle, est dégagé un ensemble de questions dont l'objectif est de guider l'acquisition d'un langage. Cette approche est ensuite exemplifiée par l'étude des systèmes de modules de Ada puis de OCaml et de C. Il est repris ensuite pour en déduire un modèle sémantique des traits objet permettant de présenter les classes de Java, C++, OCaml et Python d'un point de vue unifié.

Cet ouvrage s'adresse aussi bien à des développeurs chevronnés qui y trouveront des compléments d'information sur la sémantique des langages qu'à des programmeurs débutants, ayant simplement déjà écrit quelques courts programmes. Nous conseillons à ces débutants de manipuler les concepts sémantiques du volume 1 à l'aide des implantations fournies en OCaml ou Python afin de mieux les assimiler. La lecture du manuel de référence d'un langage de programmation peut être un bon exercice pour tous, si on confronte les présentations des constructions dans le manuel et dans l'ouvrage tout en se laissant guider par les questions du volume 2.

Il est à noter que l'aspect « algorithmique du traitement des données » n'est pas abordé dans cet ouvrage. Cependant, choisir la représentation des données et l'algorithme bien adaptés aux exigences de la spécification est une étape clé dans la création d'un logiciel. De nombreux et excellents ouvrages d'algorithmique sont disponibles et le lecteur est vivement encouragé à les consulter. Nous recommandons également l'utilisation des bibliothèques offertes par le langage de programmation choisi. Elles offrent des implantations pour bon nombre d'algorithmes, beaucoup testées car beaucoup utilisées, ce qui donne une bonne assurance de leur correction.