Claude Delannoy

Programmer en langage C++

8e édition

Avec une intro aux design patterns
et une annexe sur la norme C++11

© Groupe Eyrolles, 2014, pour la nouvelle présentation, ISBN : 978-2-212-14008-8.

EYROLLES

Avant-propos

1 Historique de C++

Très tôt, les concepts de la programmation orientée objet (en abrégé P.O.O.) ont donné naissance à de nouveaux langages dits « orientés objets » tels que Smalltalk, Simula, Eiffel ou, plus récemment, Java, PHP ou Python. Le langage C++, quant à lui, a été conçu suivant une démarche hybride. En effet, Bjarne Stroustrup, son créateur, a cherché à adjoindre à un langage structuré existant (le C), un certain nombre de spécificités lui permettant d'appliquer les concepts de P.O.O. Dans une certaine mesure, il a permis à des programmeurs C d'effectuer une transition en douceur de la programmation structurée vers la P.O.O. De sa conception jusqu'à sa normalisation, le langage C++ a quelque peu évolué. Initialement, un certain nombre de publications de AT&T ont servi de référence du langage. Les dernières en date sont : la version 2.0 en 1989, les versions 2.1 et 3 en 1991. C'est cette dernière qui a servi de base au travail du comité ANSI qui, sans la remettre en cause, l'a enrichie de quelques extensions et surtout de composants standard originaux se présentant sous forme de fonctions et de classes génériques qu'on désigne souvent par le sigle S.T.L (*Standard Template Library*). Une première norme de C++ a été publiée par l'ANSI en juillet 1998. Quelques modifications mineures ont été apportées en 2003, mais elles ne concernent pas l'utilisateur. En revanche, un nouveau standard, dit C++11 a été publié en 2011 (l'ancien standard étant noté indifféremment C++98 ou C++03).

2 Objectifs et structure de l'ouvrage

Cet ouvrage est destiné à tous ceux qui souhaitent maîtriser la programmation en C++. Il s'adresse à la fois aux étudiants, aux développeurs et aux enseignants en informatique. Il ne requiert aucune connaissance en P.O.O, ni en langage C[1] ; en revanche, il suppose que le lecteur possède déjà une expérience de la programmation structurée, c'est-à-dire qu'il est habitué aux notions de variables, de types, d'affectation, de structures de contrôle, de fonctions, etc., qui sont communes à la plupart des langages en usage aujourd'hui (Cobol, Pascal, C/C++, Visual Basic, Delphi, Perl, Python, JavaScript, Java, PHP...).

L'ouvrage est conçu sous la forme d'un cours progressif. Généralement, chaque notion fondamentale est illustrée d'un programme simple mais complet (et assorti d'un exemple d'exécution) montrant comment la mettre en œuvre dans un contexte réel. Cet exemple peut également servir à une prise de connaissance intuitive ou à une révision rapide de la notion en question, à une expérimentation directe dans votre propre environnement de travail ou encore de point de départ à une expérimentation personnelle.

Nous y étudions l'ensemble des possibilités de programmation structurée du C++, avant d'aborder les concepts orientés objet. Cette démarche se justifie pour les raisons suivantes :

- La P.O.O. s'appuie sur la plupart des concepts de programmation structurée : variables, types, affectation, structure de contrôle, etc. Seul le concept de fonction se trouve légèrement adapté dans celui de « méthode ».
- Le C++ permet de définir des « fonctions ordinaires » au même titre qu'un langage non orienté objet, ce qui n'est théoriquement pas le cas d'un pur langage objet où il n'existe que des méthodes s'appliquant obligatoirement à des objets[2]. Ces fonctions ordinaires (indépendantes d'un objet) sont même indispensables en C++ dans certaines circonstances telles que la surdéfinition d'opérateurs. Ne pas utiliser de telles fonctions, sous prétexte qu'elles ne correspondent pas à une « pure programmation objet », reviendrait à se priver de certaines possibilités du langage.
- Sur un plan pédagogique, il est plus facile de présenter la notion de classe quand sont assimilées les autres notions fondamentales sur lesquelles elle s'appuie.

Les aspects orientés objet sont ensuite abordés de façon progressive, mais sans pour autant nuire à l'exhaustivité de l'ouvrage. Nous y traitons, non seulement les purs concepts de P.O.O. (classe, constructeur, destructeur, héritage, redéfinition, polymorphisme, programmation générique), mais aussi les aspects très spécifiques au langage (surdéfinition d'opérateurs, fonctions amies, flots, gestion d'exceptions). Nous pensons ainsi permettre au lecteur de devenir parfaitement opérationnel dans la conception, le développement et la mise au point

1. Un autre ouvrage du même auteur, *C++ pour les programmeurs C*, s'adresse spécifiquement aux connaisseurs du langage C.

2. En fait, certains langages, dont Java, permettent de définir des « méthodes de classe », indépendantes d'un quelconque objet, jouant finalement pratiquement le même rôle que ces fonctions ordinaires.

de ses propres classes. C'est ainsi, par exemple, que nous avons largement insisté sur le rôle du constructeur de recopie, ainsi que sur la redéfinition de l'opérateur d'affectation, éléments qui conduisent à la notion de « classe canonique ». Toujours dans le même esprit, nous avons pris soin de bien développer les notions avancées mais indispensables que sont la ligature dynamique et les classes abstraites, lesquelles débouchent sur la notion la plus puissante du langage (et de la P.O.O.) qu'est le polymorphisme. De même, la S.T.L. a été étudiée en détail, après avoir pris soin d'exposer préalablement d'une part les notions de classes et de fonctions génériques, d'autre part celles de conteneur, d'itérateur et d'algorithmes qui conditionnent la bonne utilisation de la plupart de ses composants.

3 L'ouvrage, C, C++ et Java

L'ouvrage est entièrement fondé sur la norme ANSI/ISO du langage C++. Compte tenu de la popularité du langage Java, nous avons introduit de nombreuses remarques titrées « En Java ». Elles mettent l'accent sur les différences majeures existant entre Java et C++. Elles seront utiles non seulement au programmeur Java qui apprend ici le C++, mais également au lecteur qui, après la maîtrise du C++, souhaitera aborder l'étude de Java. En outre, quelques remarques titrées « En C » viennent signaler les différences les plus importantes existant entre C et C++. Elles serviront surtout au programmeur C++ souhaitant réutiliser du code écrit en C.

Enfin, la présente édition a été dotée d'un nouveau chapitre d'introduction aux design patterns, ainsi que d'une annexe présentant les nouveautés de la norme C++11.