1

Bien débuter en C++

Au sommaire de ce chapitre

- Pourquoi choisir le langage C++ ?
- Les étapes du cycle de développement d'un programme
- Écrire, compiler et lancer votre premier programme C++

Introduction

Bienvenue dans *Le Langage C++* de la collection Le Programmeur ! Cet ouvrage a pour objectif de vous initier efficacement à la programmation en langage C++.

Un peu d'histoire...

Les langages de programmation ont considérablement évolué depuis les premiers calculateurs qui avaient été conçus pour les calculs de trajectoires d'artillerie durant la seconde guerre mondiale. À cette époque, les programmeurs travaillaient avec le langage informatique le plus primitif qui soit – le langage machine – ce qui les obligeait à gérer de longues

MCours.com

chaînes de 1 et de 0. Puis, les premiers assembleurs apparurent, afin de rendre les instructions machine plus compréhensibles et plus faciles à utiliser puisqu'elles étaient désormais représentées par des instructions mnémoniques comme `MOV` et `ADD`.

Dans les années 60 apparurent des langages plus évolués comme BASIC et COBOL, qui permettaient aux programmeurs d'utiliser une syntaxe proche de la langue anglaise (le code source), avec des instructions et des mots comme `Let I = 100`. Les lignes d'instructions étaient ensuite traduites en langage machine par des interpréteurs ou des compilateurs.

Un *interpréteur* traduit et exécute une à une les instructions du programme (ou *code source*) et les transforme directement en actions.

Un *compilateur* passe par une étape intermédiaire (la compilation) qui produit d'abord un fichier objet. Le compilateur fait ensuite appel à un éditeur de liens (ou *linker*) qui transforme ce fichier objet en programme exécutable.

Les interpréteurs lisent les instructions ligne à ligne et exécutent le code immédiatement, ce qui en simplifie l'utilisation. Aujourd'hui, les programmes interprétés sont généralement appelés *scripts*.

Certains langages, comme Visual Basic 6, désignent l'interpréteur sous le nom de *bibliothèque d'exécution*. D'autres langages, comme Visual Basic .NET et Java, disposent d'un autre composant, appelé "machine virtuelle" (VM, *Virtual Machine*) ou exécuteur. Bien que la VM soit aussi un interpréteur, il ne s'agit plus d'un interpréteur de code source qui se contente de traduire un langage lisible en code objet : la VM interprète et exécute un "langage machine indépendant de l'ordinateur" compilé, parfois appelé "langage intermédiaire".

Les compilateurs ajoutent les étapes supplémentaires de compilation du code source (compréhensible par l'homme) en code objet (lisible par la machine). Ceci pourrait apparaître comme un inconvénient mais, en réalité, cette étape permet de créer un programme dont la vitesse d'exécution est optimisée puisque la traduction du fichier source en langage machine a déjà été réalisé une fois pour toutes, au moment de la compilation. Il n'est donc plus nécessaire de retraduire le programme à chaque fois qu'on l'exécute.

L'autre avantage des langages compilés comme C++ tient à la diffusion des programmes puisque l'on peut distribuer un fichier exécutable à des personnes qui ne disposent pas du compilateur. Avec un langage interprété, par contre, l'utilisateur doit nécessairement posséder l'interpréteur pour pouvoir exécuter le programme.

C++ est généralement un langage compilé, même s'il existe quelques interpréteurs C++. À l'instar de nombreux langages compilés, il a la réputation de produire des programmes rapides et performants.

En fait, pendant longtemps, la principale préoccupation des programmeurs était de concevoir des applications très courtes pouvant s'exécuter rapidement, car la mémoire et le temps de calcul coûtaient cher. Avec la miniaturisation des ordinateurs, l'augmentation de

leurs performances et la chute des prix, les priorités ont changé. Désormais, le coût de développement dépasse largement celui d'un ordinateur de type PC ou mini. L'important est d'écrire des programmes performants, bien construits et faciles à mettre à jour (c'est-à-dire sans surcoût excessif).

Info

Le mot programme a deux sens. Il désigne les instructions (ou code source) écrites par un développeur, mais également l'ensemble d'un logiciel exécutable. Cette homonymie peut être une source de confusion et il est important de faire la distinction entre le fichier source et le programme exécutable.

Comment résoudre les problèmes

Les problèmes auxquels sont confrontés les programmeurs sont totalement différents de ceux qu'ils devaient résoudre il y a une vingtaine d'années. Dans les années 80, les programmes traitaient des volumes importants de données brutes. Le programmeur et l'utilisateur final étaient tous deux des spécialistes de l'informatique. De nos jours, les utilisateurs sont bien plus nombreux et peu connaissent tous les détails des ordinateurs et du fonctionnement des programmes. L'utilisateur final actuel recherche des solutions prêtes à l'emploi et capables d'effectuer des opérations de gestion courantes ou ponctuelles.

L'informatique est devenue plus conviviale, mais ce processus a également conduit à la mise en œuvre de programmes de plus en plus complexes. Dans les années 70, les utilisateurs étaient contraints de saisir des commandes énigmatiques pour voir défiler à l'écran des volumes impressionnants de données brutes. Cette époque est révolue ! Une application se compose désormais de fenêtres, de menus et de boîtes de dialogue intégrés à une interface conviviale.

Avec le développement du Web, les ordinateurs ont abordé une ère nouvelle de pénétration du marché ; les utilisateurs d'ordinateurs sont plus nombreux que jamais, et ils sont très exigeants.

Au cours des dernières années, les applications se sont également étendues à d'autres périphériques : l'ordinateur de bureau n'est plus la seule cible des applications. Les téléphones portables, les assistants personnels (PDA), les PC de poche et autres périphériques constituent des cibles toutes trouvées pour les applications modernes.

Depuis la première édition de ce livre, les programmeurs ont répondu aux demandes des utilisateurs et les programmes sont devenus plus volumineux et plus complexes. La nécessité de développer des techniques de programmation permettant de gérer cette complexité est devenue évidente.

Les besoins changeant, les techniques et les langages évoluent également pour aider les programmeurs à gérer la complexité des demandes. Dans cet ouvrage, nous nous

concentrerons uniquement sur une partie essentielle de cette évolution : le passage de la programmation procédurale à la programmation orientée objet.

La programmation procédurale, structurée et orientée objet

Il y a quelques années encore, les programmes étaient conçus comme des suites de procédures destinées à traiter les données. Une *procédure* – également appelée *fonction* ou *méthode* – est un ensemble d'instructions s'exécutant l'une après l'autre. Les données et les procédures étaient totalement dissociées et le travail du programmeur consistait à connaître les fonctions appelées par d'autres fonctions et les données qui étaient modifiées. Pour faire face à ce niveau de complexité, on a donc inventé la *programmation structurée*.

Le principe général de la programmation structurée consiste à diviser pour mieux régner. Un programme peut alors être considéré comme un ensemble de tâches. Toute opération trop complexe pour être décrite simplement est décomposée en un ensemble d'opérations plus simples, jusqu'à n'obtenir que des tâches suffisamment triviales pour être aisément compréhensibles.

Le calcul du salaire moyen de chaque employé d'une entreprise est, par exemple, une tâche assez complexe. Toutefois, il est possible de diviser le traitement en plusieurs tâches secondaires :

1. Compter le nombre d'employés.
2. Déterminer le revenu de chaque employé.
3. Faire le total de tous les salaires.
4. Diviser cette valeur par le nombre d'employés.

La troisième étape (total des salaires) peut également se diviser en tâches plus simples :

1. Lire l'enregistrement de chaque salarié.
2. Extraire le salaire de chaque employé.
3. Ajouter cette valeur au total général.
4. Accéder à l'enregistrement suivant.

La lecture de chaque enregistrement peut, de la même façon, être décomposée en opérations plus élémentaires :

1. Ouvrir le fichier des employés.
2. Rechercher le bon enregistrement.
3. Lire les données.

La programmation structurée résout des problèmes complexes de manière très fiable. Toutefois, à la fin des années 80, cette méthode a montré ses limites.

D'une part, il est naturel d'associer les données (les enregistrements des employés, par exemple) et leur traitement (tri, modification, etc.). Malheureusement, la programmation structurée sépare les données des fonctions qui les manipulent et ne propose pas de moyen naturel de les regrouper. La programmation structurée est donc souvent désignée par le terme de *programmation procédurale*, car elle met l'accent sur les procédures (plutôt que sur les *objets*).

D'autre part, les programmeurs devaient souvent réutiliser des fonctions. Or, certaines fonctions qui convenaient à un type de données ne pouvaient pas toujours être réutilisées avec d'autres, ce qui limitait leurs avantages.

La programmation orientée objet (POO)

La programmation orientée objet (POO) répond à ces besoins. Elle fournit les techniques permettant de traiter des applications très complexes, exploite des composants logiciels réutilisables et associe les données aux tâches qui les manipulent.

La caractéristique essentielle de la programmation *orientée objet* est de modéliser des "objets" (c'est-à-dire des concepts) plutôt que des "données". Ces objets peuvent être des éléments graphiques affichables, comme des boutons ou des zones de liste, ou des objets réels, comme des clients, des bicyclettes, des avions, des chats ou de l'eau.

Les objets possèdent des caractéristiques, également appelées propriétés ou attributs, comme âge, rapidité, volume, noir, humide. Ils ont aussi des fonctionnalités, appelées opérations ou fonctions, comme accélérer, voler, miauler ou couler. Le rôle de la programmation orientée objet est de représenter ces objets dans le langage de programmation.

C++ et la programmation orientée objet

Le langage C++ permet d'utiliser toutes les possibilités de la programmation orientée objet, notamment ses trois piliers que sont l'encapsulation, l'héritage et le polymorphisme.

Encapsulation

Un technicien ne fabrique pas les composants qu'il assemble. Il les choisit selon leurs spécifications sans se préoccuper de leur fonctionnement interne.

L'autonomie d'un objet est une propriété résultant d'un processus appelé *encapsulation*. Celle-ci permet de masquer les données internes d'un objet et de l'utiliser sans connaître les détails de son fonctionnement, exactement comme vous utilisez votre réfrigérateur

sans comprendre le principe des compresseurs. Il reste possible de modifier ce fonctionnement interne sans affecter celui du programme, à condition toutefois que les spécifications soient respectées (le compresseur du réfrigérateur peut être remplacé par un autre de conception similaire).

Lorsque notre technicien veut utiliser un composant électronique, il n'a pas besoin d'en connaître les rouages internes. Toutes les propriétés de ce composant sont encapsulées dans l'objet composant, elles ne sont pas communiquées au circuit monté. Il n'est pas nécessaire de connaître le fonctionnement du composant pour l'utiliser efficacement. Ce fonctionnement est masqué par son boîtier.

Le langage C++ gère l'encapsulation à l'aide de types définis par l'utilisateur : les *classes*. Pour en savoir plus sur la conception d'une classe, reportez-vous au Chapitre 6. Si elle est correctement définie, une classe agit comme une entité encapsulée – elle fonctionne comme un composant autonome. Comme pour un objet du monde réel, son fonctionnement interne peut être masqué. Les utilisateurs d'une classe bien conçue n'ont pas besoin de savoir comment elle fonctionne, mais uniquement comment l'utiliser.

Héritage et réutilisabilité

Lorsqu'ils souhaitent concevoir une nouvelle voiture, les ingénieurs de Superauto ont le choix entre monter un projet à partir de zéro et modifier un modèle existant. Le modèle Pipo est peut-être parfait pour la ville, mais cette voiture manque de nervosité sur autoroute. Les ingénieurs ont donc décidé de lui adjoindre un turbo-compresseur et une boîte à six vitesses. Le responsable du projet préfère donc partir d'un modèle existant, le perfectionner, le tester, et l'appeler Star. La Star sera donc une sorte de Pipo, mais il s'agira d'une version spéciale, disposant de nouvelles fonctionnalités.

Le langage C++ implémente la notion d'héritage. Grâce à l'héritage, vous pouvez déclarer un nouveau type à partir d'un type existant. On dit que la sous-classe obtenue dérive du type existant ; on la nomme quelquefois "type dérivé". Si la Star est dérivée de la Pipo et hérite donc de toutes ses qualités, certaines propriétés peuvent lui être ajoutées, d'autres modifiées. Pour en savoir plus sur l'héritage et son application en C++, reportez-vous aux Chapitres 12 et 16.

Polymorphisme

La Star ne répondra pas nécessairement de la même façon que la Pipo à un appui sur l'accélérateur. En effet, la première peut utiliser son injection et son turbo alors que la seconde devra se contenter de sa carburation classique. Quoi qu'il en soit, il suffit au conducteur de démarrer son véhicule et de se déplacer où bon lui semble. Il n'est pas obligé de connaître les détails du moteur.

En C++, des objets différents peuvent avoir des comportements adéquats différents en réponse à la même action grâce au polymorphisme de fonction et de classe. *Poly* signifie plusieurs et *morphe* signifie forme. Le polymorphisme se traduit donc par un nom unique pouvant prendre plusieurs formes, et il est traité aux Chapitres 10 et 14.

Évolution de C++

Lorsque les qualités de la programmation, de la conception et de l'analyse orientées objet commencèrent à être reconnues, Bjarne Stroustrup créa C++ à partir du langage le plus utilisé pour le développement des logiciels professionnels, le langage C. Il lui ajouta tous les éléments nécessaires à la programmation orientée objet.

On a coutume de dire que C++ est un surensemble de langage C et que, par conséquent, tout programme C est virtuellement un programme C++. Pourtant, ces deux langages sont très différents. Pendant longtemps, C++ a attiré les programmeurs C car ces derniers trouvaient sa syntaxe familière. Toutefois, pour tirer le meilleur profit des fonctionnalités de C++, de nombreux développeurs ont compris qu'ils devaient laisser de côté une partie de leurs acquis en C et aborder les problèmes différemment.

Est-il nécessaire d'apprendre d'abord le langage C ?

Nombreux parmi vous sont ceux qui vont se poser cette question, puisque C++ est un *surensemble* du langage C. La réponse de son créateur et de la plupart des programmeurs C++ est la suivante : il est inutile d'apprendre le langage C, voire préférable de commencer directement par le langage C++.

La programmation C est fondée sur les concepts de programmation structurée, alors que la programmation C++ repose sur ceux de la programmation orientée objet. Si vous apprenez d'abord le langage C, vous devrez alors vous défaire des habitudes néfastes liées à ce langage.

Ce livre ne s'adresse pas obligatoirement à un public ayant une expérience préalable de la programmation. Si vous êtes programmeur C, vous pouvez vous contenter de survoler les premiers chapitres du livre. Le développement orienté objet n'est réellement abordé qu'à partir du Chapitre 6.

C++, Java et C#

C++ est l'un des principaux langages pour le développement de logiciels professionnels. Ces dernières années, Java a eu un temps la faveur des programmeurs. Toutefois, nombre

d'entre eux, qui avaient abandonné C++ au profit de Java, ont depuis fait marche arrière. En tout état de cause, les deux langages sont si semblables qu'apprendre l'un revient à connaître 90 % de l'autre.

C# (prononcez *C sharp*) est un langage plus récent développé par Microsoft pour la plate-forme .NET. Il utilise la même syntaxe que C++ et, bien que ces deux langages diffèrent en quelques points importants, l'apprentissage de C++ apporte la majorité des connaissances nécessaires à l'utilisation de C#. Si vous décidiez par la suite d'apprendre C#, l'investissement réalisé dans l'apprentissage de C++ vous sera très bénéfique.

Extensions gérées de Microsoft pour C++

Avec l'arrivée de .NET, Microsoft a introduit les extensions gérées pour C++ (*Managed C++*). Il s'agit d'une extension du langage C++ lui permettant d'utiliser la nouvelle plate-forme Microsoft et ses bibliothèques. Managed C++ permet surtout à un programmeur C++ de profiter des fonctionnalités avancées de l'environnement .NET. Au cas où vous décideriez de créer des applications spécifiquement conçues pour la plate-forme .NET, vous devrez étendre votre connaissance du C++ standard pour y inclure ces extensions.

Norme ANSI

Le comité d'accréditation des standard, qui dépend de l'ANSI (*American National Standards Institute*), a élaboré un standard international pour C++.

Le standard C++ est également appelé norme ISO (*International Organization for Standardization*), norme NCITS (*National Committee for Information Technology Standards*), norme X3 (ancienne appellation de NCITS), ou norme ANSI/ISO. Dans ce livre, nous continuerons à faire référence à la norme ANSI car c'est le terme le plus utilisé.

L'objectif du standard ANSI est de garantir la portabilité du C++ afin que le code que vous allez écrire pour le compilateur de Microsoft, par exemple, ne génère pas d'erreur avec un autre compilateur. Le code présenté dans ce livre étant compatible ANSI, il pourra être compilé sans erreur sur des plates-formes Macintosh, Windows ou Unix.

Pour la plupart des utilisateurs C++, le standard ANSI est transparent. La version la plus récente de cette norme est l'ISO/IEC 14882-2003. La version précédente, l'ISO 14882-1998, a bénéficié d'une stabilité durable et tous les fournisseurs de renom la prennent en charge. Nous nous sommes efforcés d'assurer une totale compatibilité ANSI du code dans cette édition.

Toutefois, n'oubliez pas que les compilateurs ne sont pas tous totalement compatibles avec la norme. En outre, certaines parties de la norme ont été laissées au choix du concepteur

du compilateur : il n'est donc pas garanti que ces parties se compilent ou fonctionnent de la même manière avec des compilateurs différents.

Les extensions gérées pour C++ ne s'appliquant qu'à la plate-forme .NET et, ne faisant pas partie de la norme ANSI, elles ne sont pas traitées dans cet ouvrage.

Préparation à la programmation

Plus que tout autre langage, C++ oblige le développeur à concevoir soigneusement une application avant de l'écrire. Les programmes figurant dans les premiers chapitres de cet ouvrage ne nécessitent pas d'analyse car ils sont triviaux. En revanche, les problèmes complexes rencontrés en programmation professionnelle ne peuvent être résolus qu'avec cette approche. La conception et l'analyse permettent de mieux cerner les différents aspects du problème. Un programme bien construit ne contient pas d'erreurs et peut aisément être mis à jour. D'après des études récentes, on estime que le coût d'un programme repose à 90 % sur la mise au point et la maintenance. La phase de conception permet de réduire les coûts et, par là même, le prix de revient du logiciel.

Avant de passer à la conception d'un programme, vous devez connaître parfaitement le problème à résoudre. Les programmes les plus simples comme les plus complexes s'articulent autour d'un déroulement clair et logique.

Il convient également de déterminer si le problème peut être résolu à l'aide d'un programme existant qui sera modifié, ou à l'aide d'un logiciel du commerce. Qu'il choisisse l'une ou l'autre solution, le programmeur ne manquera de toutes façons pas de travail : trouver des solutions moins coûteuses à des problèmes actuels produira toujours de nouvelles opportunités un peu plus tard.

En supposant que le problème à résoudre ait été bien compris et qu'il faille écrire un nouveau programme, vous êtes prêt à commencer votre conception.

Le processus d'appréhension totale du problème (*analyse*) et d'élaboration d'un plan pour une solution (*conception*) est indispensable à l'obtention d'une application professionnelle de carrure internationale.

Votre environnement de développement

Pour utiliser ce livre, nous supposons que vous disposez d'un compilateur permettant de saisir des données directement sur une "console" (par exemple une fenêtre de commande MS-DOS ou une fenêtre shell), c'est-à-dire sans vous préoccuper d'un environnement

graphique comme Windows ou Macintosh. Recherchez une option comme `console` ou `easy window` ou consultez la documentation de votre compilateur.

Votre compilateur peut faire partie d'un environnement de développement intégré (IDE) ou posséder son propre éditeur de texte pour saisir le code source des programmes. Vous pouvez également utiliser un éditeur de texte séparé ou un logiciel de traitement de texte du moment que vous produisez des fichiers texte sans formatage ni style particuliers. Le Bloc-notes de Windows, l'éditeur Edit de DOS, les éditeurs Brief, Epsilon, Emacs et vi sont particulièrement bien adaptés à la saisie des codes sources. Si vous disposez d'un traitement de texte tel que WordPerfect ou Word (ou autre), utilisez la commande d'enregistrement au format texte simple.

Les documents créés à partir d'un éditeur de texte s'appellent des fichiers sources. En C++, ils portent traditionnellement l'extension `.cpp`, `.cp` ou `.c`. Dans cet ouvrage, nous avons choisi l'extension `.cpp`. Vérifiez que votre compilateur la prend en charge.

La plupart des compilateurs C++ acceptent toutes les extensions et affectent par défaut l'extension `.cpp` aux fichiers sources. Toutefois, soyez prudents, car certains compilateurs traitent les fichiers `.c` comme du code C et les fichiers `.cpp` comme du code C++. Vérifiez la documentation du compilateur. Dans tous les cas, une utilisation cohérente des `.cpp` pour les fichiers du code source C++ facilitera la tâche des programmeurs qui devront comprendre votre code.

Faire	***Ne pas faire***
• Écrire le fichier source à l'aide de l'éditeur de texte fourni avec le compilateur ou d'un éditeur externe. • Enregistrer votre fichier source avec l'extension `.c`, `.cp` ou `.cpp`. • Consulter la documentation du compilateur et de l'éditeur de liens pour connaître les différentes étapes de la création du programme.	• Utiliser les fonctions de mise en forme d'un traitement de texte. Si vous utilisez un traitement de texte, sauvegardez le fichier source au format texte ASCII. • Utiliser une extension `.c` si votre compilateur considère ces fichiers comme du code C et non du code C++.

Création du programme

La première étape de la création d'un programme consiste à écrire les commandes adaptées (instructions) dans un fichier source. Même si les instructions du fichier source semblent quelque peu mystérieuses pour ceux qui ne connaissent pas C++, il s'agit quand même d'un format lisible. Le fichier source n'est pas un programme : il vous sera impossible de le lancer ou de l'exécuter comme vous le feriez avec un programme exécutable.

MCours.com