Traduction de C++ Primer 5th Edition

Auteurs: Stanley B. Lippman, Josée Lajoie et Barbara E. Moo

Traduction non-officielle: Stomab

Contents

0.1 Préface

De nombreux programmeurs ont appris le C++ avec les précédentes éditions de *C++ Primer*. Pendant ce temps, le C++ a gagné en maturité: son objectif, encouragé par la communauté de développeurs, a été de se focaliser davantage sur l'efficacité du *programmeur* plutôt que de l'efficacité de la *machine*.

En 2011, le comité des normes C++ a publié une révision majeure de la norme ISO C++. Cette révision est la dernière étape de l'évolution du C++ et continue de mettre l'accent sur l'efficacité du programmeur. Les principaux objectifs de la nouvelle norme sont les suivants:

- Rendre le langage plus uniforme, plus facile à enseigner et à apprendre ;
- Simplifier les bibliothèques standards, les rendre plus sûres et plus efficaces à utiliser ;
- Faciliter l'écriture des abstractions et des bibliothèques de manière efficace.

Dans cette édition, nous avons intégralement revu le *C++ Primer* en y incorporant les dernières normes. Vous pourrez apprécier les différents changements impactés par ces nouvelles normes en consultant la table des matières.

Certains ajouts dans la nouvelle norme, tels que *auto* pour l'inférence de type, sont omniprésents. Ces fonctionnalités rendent le code de cette édition plus facile à lire et à comprendre. Les programmes (et les programmeurs !) peuvent ignorer ces détails, pour pouvoir se concentrer sur ce que le programme est censé faire. D'autres nouvelles fonctionnalités, telles que les pointeurs intelligents et les conteneurs avec éléments déplacés, permettront d'écrire des classes plus sophistiquées sans avoir à composer avec les complexités de la gestion des ressources. En conséquence, vous pourrez apprendre à écrire vos propres classes bien plus tôt dans le livre par rapport à l'édition précédente.

Nous avons mis en évidence, dans le texte, les nouvelles fonctionnalités de la dernière norme, à l'aide d'une icône spécifique. Nous espérons que les lecteurs connaissant déjà le C++, trouveront ces remarques pertinentes pour décider où concentrer leur attention. Nous espérons également que ces remarques, encadrées par ces icônes spécifiques, aideront à expliquer les messages d'erreur des compilateurs qui pourraient ne pas encore prendre en charge toutes les nouvelles fonctionnalités. Bien que la quasi-totalité des exemples de ce livre ont été compilés avec une version récente du compilateur GNU, nous sommes conscients que certains lecteurs n'auront pas encore accès à des compilateurs récents et mis à jour. Même si de nombreuses fonctionnalités ont été ajoutées pour répondre aux nouvelles normes, le langage de base reste inchangé et constitue l'essentiel du matériel que nous couvrons. Les lecteurs peuvent utiliser ces icônes pour noter quelles fonctionnalités peuvent ne pas être encore disponibles avec leur compilateur.

C++ 11

0.1.1 Pourquoi lire cet ouvrage ?

Le C++ moderne peut se résumer en trois parties :

- Un langage de bas niveau, dont une grande partie est héritée du C ;
- Des fonctionnalités de langage plus avancées qui nous permettent de définir nos propres types et organiser des programmes et des systèmes à grande échelle ;
- La bibliothèque standard, qui utilise ces fonctionnalités avancées pour fournir des structures de données et des algorithmes.

La plupart des ouvrages présentent le C++ dans l'ordre dans lequel il a évolué. Le sous-ensemble C est d'abord enseigné, puis viennent les fonctionnalités les plus abstraites de C++ sous forme de sujets avancés à la fin du livre seulement. Il y a deux problèmes avec cette approche: les lecteurs peuvent s'enliser dans les détails inhérents à la programmation de bas niveau et abandonner par frustration. Ceux qui persévèrent prennent de mauvaises habitudes qu'ils doivent désapprendre plus tard.

Nous adoptons la démarche inverse: dès le début, nous utilisons les fonctionnalités qui permettent aux programmeurs d'ignorer les détails inhérents à la programmation de bas niveau. Par exemple, nous introduisons et utilisons les bibliothèques *string* et *vector* avec les opérateurs arithmétiques et les tableaux. Les programmes qui utilisent ces types de bibliothèques sont plus faciles à écrire, plus facile à comprendre, et moins sujets aux erreurs.

Trop souvent, la bibliothèque est enseignée comme un sujet avancé. Au lieu d'utiliser la bibliothèque, de nombreux livres utilisent des techniques de programmation de bas niveau basées sur des pointeurs vers des tableaux de caractères et une gestion dynamique de la mémoire. Obtenir des programmes qui utilisent ces techniques de bas niveau pour fonctionner correctement est beaucoup plus difficile que d'écrire le code C++ correspondant à l'aide de la bibliothèque.

Tout au long de *C++ Primer*, nous mettons l'accent sur le bon style à adopter: nous voulons vous aider, le lecteur, à acquérir immédiatement de bonnes habitudes et à éviter d'avoir à désapprendre de

mauvaises habitudes à mesure que vous apprenez de nouvelles choses de plus en plus complexes. Nous soulignons les questions particulièrement délicates et mettons en garde contre les idées fausses et les pièges courants.

Nous expliquons également la raison d'être des règles, sans nous contenter de les décrire, mais en expliquant également le pourquoi de leur existence. Nous croyons qu'en comprenant pourquoi les choses fonctionnent ainsi, les lecteurs peuvent faciliter plus rapidement leur compréhension du langage.

Même si vous n'avez pas spécialement besoin de connaître le C pour comprendre ce livre, nous supposons que vous en savez suffisamment sur la programmation pour écrire, compiler et exécuter un programme dans un langage moderne structuré en blocs. En particulier, nous partons du principe que vous savez utiliser des variables, écrire et appeler des fonctions et que vous avez déjà utilisé un compilateur.

0.1.2 Ce qui change dans cette nouvelle édition

La nouveauté de cette édition de *C++ Primer* est l'ajout d'icônes dans les marges pour guider le lecteur. C++ est un langage puissant qui offre des fonctionnalités adaptées à des raisonnements particuliers de programmation. Certaines de ces fonctionnalités sont très importantes pour les grandes équipes de projet, mais peuvent ne pas être nécessaires pour de plus petits projets. Par conséquent, tous les programmeurs ne sont pas dans l'obligation de connaître chaque détail de chaque fonctionnalité. Nous avons ajouté ces icônes accessoires pour guider le lecteur et lui indiquer les parties qui peuvent être approfondies ultérieurement ou au contraire, les sujets qui sont plus essentiels.

Certains passages du livre qui abordent les principes fondamentaux du langage, seront signalés par une image représentant une personne plongée dans un livre. Les sujets traités dans les sections marquées de cette manière forment la partie centrale du langage. Tout le monde devrait lire et comprendre ces passages du livre.

Nous avons également indiqué les passages qui couvrent des sujets avancés ou à usage spécifique. Ces sections peuvent être sautées ou survolées lors d'une première lecture. Nous indiquons les passages en question à l'aide d'une icône représentant une pile de livres. Cela signifie que vous pouvez poser le livre en toute sécurité à ce stade de la lecture. Parcourir ces différentes sections peut s'avérer utile pour se tenir informé de ce qu'il est possible de faire. Cependant, il n'y a aucune raison de passer du temps à étudier ces sujets jusqu'à ce que vous ayez réellement besoin d'utiliser la fonctionnalité dans vos propres programmes.

Pour aider les lecteurs à retenir leur attention sur un passage en particulier, nous avons mis en évidence les concepts particulièrement délicats avec une icône en forme de loupe. Nous espérons que les lecteurs prendront le temps de bien comprendre le sujet traité dans les passages marqués ainsi. Dans au moins certaines de ces sections, l'importance du sujet peut ne pas être évidente ; mais nous misons sur votre capacité à comprendre l'importance des sujets abordés et qui s'avèrent essentiels à la compréhension du langage.

Ce qui reste inchangé, c'est que *C++ Primer* est un guide didactique clair, correct et complet sur C++. Nous enseignons le langage en présentant une série d'exemples de plus en plus sophistiqués, qui expliquent les fonctionnalités du langage et montrent comment tirer le meilleur parti de C++.

0.1.3 Structure de ce livre

Nous commençons par couvrir ensemble les bases du langage et de la bibliothèque dans les parties I et II. Ces parties couvrent suffisamment de contenu pour vous permettre d'écrire des programmes. La plupart des programmeurs en C++ ont besoin de connaître essentiellement tout ce qui sera abordé dans cette partie du livre.

En plus d'enseigner les bases du C++, le contenu des parties I et II vise un autre objectif important: en utilisant les fonctions définies par la bibliothèque, vous deviendrez plus à l'aide avec l'utilisation de techniques de programmation de haut niveau. Les fonctionnalités de la bibliothèque sont elles-mêmes des types de données abstraites qui sont généralement écrites en C++. La bibliothèque peut être définie en utilisant les mêmes caractéristiques de construction de classe que ...

[A CONTINUER - INCOMPLET]

Chapter 1

COMMENCER

Ce chapitre introduit la plupart des éléments de base de C++: types, variables, expressions, déclarations, et les fonctions. En cours de route, nous expliquerons brièvement comment compiler et exécuter un programme.

Après avoir lu ce chapitre et réalisé les différents exercices, vous devriez être capable d'écrire, de compiler et d'exécuter des programmes simples. Les chapitres suivants partiront du principe que vous pouvez utiliser les fonctionnalités présentées dans ce chapitre et expliqueront ces fonctionnalités plus en détail.

Le secret de la réussite dans l'apprentissage d'un nouveau langage de programmation est d'écrire des programmes. Dans ce chapitre, nous allons écrire un programme pour résoudre un problème simple pour une librairie. Notre magasin conserve un fichier de transactions ; chacune d'elles enregistrant la vente d'un ou plusieurs exemplaires d'un même livre. Chaque transaction contient trois éléments de données:

```
0-201-70353-X 4 24.99
```

Le premier élément est un numéro ISBN (International Standard Book NUmber, l'identifiant unique d'un livre), le second est le nombre de copies qui ont été vendues, et le dernier est le prix auquel chacun de ces exemplaires a été vendu. De temps en temps, le propriétaire de la librairie lit ce fichier et pour chaque livre calcule le nombre d'exemplaires vendus, le revenu total de ce livre et le prix de vente moyen.

Pour pouvoir réaliser ce programme, nous devons parcourir quelques fonctionnalités de base du C++. De plus, nous allons avoir besoin de savoir comment compiler et exécuter un programme.

Bien que nous n'ayons pas encore conçu notre programme, il est facile de deviner que l'on va devoir :

- Définir des variables ;
- Faire l'entrée et la sortie ;
- Utiliser une structure de données qui contiendra les données ;
- Tester si deux enregistrements ont le même ISBN ;
- Créer une boucle qui traitera chaque enregistrement du ficher de transaction.

Nous allons commencer à nous intéresser à la résolution de ces sous-problèmes en C++, puis nous écrirons notre programme de gestion de librairie.

1.1 Écrire un programme simple en C++

Chaque programme en C++ contient une ou plusieurs **fonctions**, dont l'une d'entre elles doit être appelée **main**. Le système d'exploitation exécute un programme C++ en appelant *main*. Voici la syntaxe de base d'une fonction *main* qui pour l'instant ne fait rien, mais renvoie une valeur au système d'exploitation:

```
int main()
{
        return 0;
}
```

Une définition de fonction comporte quatre éléments: un **type de valeur renvoyée**, le **nom de la fonction**, une **liste de paramètres** (éventuellement vide) entre parenthèses, et le **corps de la fonction**. Bien que *main* est un peu particulière à certains égards, nous la définissons de la même manière que toute autre fonction.

Dans cet exemple, *main* ne dispose que d'une liste vide de paramètres (indiquée par les parenthèses () qui ne contiennent rien à l'intérieur). Dans la partie 6.2.5 de livre, nous verrons les autres types de paramètres que nous pouvons définir pour la fonction *main*.

La dernière partie d'une définition de fonction, le corps de la fonction, est un **bloc d'instructions** qui commence par une accolade ouvrante et se termine par une accolade fermante:

```
{
        return 0;
}
```

La seule instruction de ce bloc est *return*, qui est donc l'instruction qui termine la fonction. Comme c'est le cas ici, un *return* peut également renvoyer une valeur à l'appelant de la fonction. Lorsqu'un *return* inclut une valeur, cette valeur renvoyée doit avoir un type compatible avec le type déclaré dans la fonction. Dans cette exemple, le type de retour de *main* est *int* et la valeur retournée est 0, qui donc est un *int*.

Notez le point-virgule à la fin de l'instruction *return*. Les points-virgules marquent la fin de la plupart des instructions en C++. On peut facilement les oublier ce qui conduit à de mystérieux messages d'erreur du compilateur.

Sur la plupart des systèmes, la valeur renvoyée par *main* est un indicateur d'état. Un 0 retourné indique un succès. Un retour différent de zéro a une signification qui est définie par le système. Habituellement, un retour différent de zéro indique le type d'erreur qui s'est produit.

NOTION CLÉ: LES TYPES

Les types sont l'un des concepts les plus fondamentaux de la programmation et un concept sur lequel nous reviendrons encore et encore dans ce livre. Un type définit à la fois le contenu d'une donnée et les opérations possibles sur cette donnée.

Les données manipulées par nos programmes sont stockées dans des variables et chacune a un type. Lorsque le type d'une variable nommée *v* est *t*, on dit souvent que "v a le type de t" ou, indifféremment, que "v est un t".

1.1.1 Compilation et Exécution de Notre Programme

Après avoir écrit le programme, nous devons le compiler. Comment compiler un programme dépend de votre système d'exploitation et de votre compilateur. Pour plus de détails sur la manière dont votre compilateur fonctionne, consultez le manuel de référence ou demandez à un collègue compétent.

De nombreux compilateurs basés sur PC sont exécutés à partir d'un environnement de développement intégré (IDE). Il s'agit d'un package qui contient : le compilateur et des outils de construction et d'analyse. Ces IDE peuvent être un atout majeur dans le développement de programmes volumineux, mais nécessite un peu de temps pour apprendre à les utiliser efficacement. Apprendre à utiliser de tels environnements dépasse largement le cadre de ce livre.

La plupart des compilateurs, y compris ceux fournis avec un IDE, fournissent une interface en ligne de commande. À moins que ne connaissiez déjà l'IDE, vous trouverez peut-être plus facile de démarrer avec l'interface de ligne de commande. Cela vous permettra de vous concentrer sur l'apprentissage C++ d'abord. De plus, une fois que vous comprenez le langage, l'IDE est susceptible d'être plus facile à apprendre.

Convention pour les noms de fichier source du programme

Que vous utilisiez une interface en ligne de commande ou un IDE, la plupart des compilateurs s'attendent à ce que le code source du programme soit stocké dans un ou plusieurs fichiers. Ces fichiers sont appelés: **les fichiers sources**. Sur la plupart des systèmes, le nom d'un fichier source se termine par une extension, c'est à dire un point suivi de plusieurs caractères. L'extension indique au système que le fichier est un programme C++. Les compilateurs ont adopté certaines conventions au niveau de l'extension. Ainsi, on retrouve habituellement les extensions *.cc*, *.cxx*, *.cpp*, *.cp* et *.C*.

Utiliser le compilateur en ligne de commande

Si nous utilisons une interface en ligne de commande, nous compilerons généralement un programme dans une fenêtre de console (tel qu'un terminal shell sur un système UNIX ou l'invite de commande pour Windows). Supposons que notre programme principal soit écrit dans un fichier nommé *prog1.cc*, nous pourrions le compiler en utilisant une commande telle que :

```
$ CC prog1.cc
```

où CC nomme le compilateur et $ est le prompt du terminal. Le compilateur génère un fichier exécutable. Sur Windows, ce fichier exécutable est nommé *prog1.exe*. Les compilateurs UNIX ont tendance à placer leurs exécutables dans des fichiers nommés *a.out*.

Pour lancer cet exécutable sur Windows, il suffit d'écrire le nom de celui sans être obligé de préciser l'extension *.exe* :

```
$ prog1
```

Sur certains systèmes, vous devez spécifier explicitement l'emplacement du fichier, même si le fichier est dans le répertoire ou le dossier en cours. Dans ce cas, on écrirait :

```
$ .\prog1
```

Le point "." qui précède l'antislash indique que le fichier se trouve dans le répertoire courant.

Pour lancer un exécutable sous UNIX, nous utilisons le nom complet du fichier, y compris l'extension :

```
$ ./a.out
```

La valeur renvoyée par *main* est accessible de manière dépendante du système. Sur systèmes UNIX et Windows, après avoir exécuté le programme, vous devez émettre une commande *echo* appropriée.

Sur les systèmes UNIX, on obtient le statut en écrivant :

```
$ echo $?
```

Pour obtenir le statut sur Windows, on écrit :

```
$ echo %ERRORLEVEL%
```

LANCER LE COMPILATEUR GNU OU MICROSOFT

La commande utilisée pour lancer le compilateur C++ varie selon les compilateurs et les systèmes d'exploitation. Les compilateurs les plus courants sont le compilateur GNU et le compilateur Microsoft Visual Studio.Par défaut, la commande pour exécuter le compilateur GNU est **g++**:

```
$ g++ -o prog1 prog1.cc
```

Le symbole $ est le prompt. Le **-o prog1** est un argument pour le compilateur et il nomme le fichier dans lequel sera mis le fichier exécutable. La commande génère un fichier exécutable nommé **prog1** ou **prog1.exe**, en fonction du système d'exploitation. Sur UNIX, les fichiers exécutables n'ont pas d'extension ; sur Windows, l'extension est **.exe**. Si l'on n'avait pas mis le **-o prog1**, le compilateur aurait généré un exécutable appelé **a.out** sur un système UNIX et **a.exe** sur Windows. (À noter : selon la version du compilateur GNU que vous utilisez, vous devrez peut-être spécifier **-std=c++0x** pour activer la prise en charge de C++ 11).

La commande pour lancer le compilateur Microsoft Visual Studio 2010 est **cl**:

```
$ C:\Users\me\Programs> cl /EHsc prog1.cpp
```

Ici : **C:/Users/me/Programs>** est le prompt du système et **/Users/me/Programs** est le nom du répertoire courant (alias: le dossier courant). La commande **cl** invoque le compilateur, et **/EHsc** est l'option du compilateur qui active la gestion standard des exceptions. Le compilateur Microsoft génère automatiquement un exécutable avec un nom qui correspond au premier nom du fichier source. L'exécutable a l'extension **.exe** et le même nom que le ficheir source. Dans notre cas, l'exécutable est nommé **prog1.exe**.

Les compilateurs incluent généralement des options pour générer des avertissements sur constructions problématiques. C'est généralement une bonne idée d'utiliser ces options. Notre préférence est d'utiliser **-Wall** avec le compilateur GNU, et utiliser **/W4** avec les compilateurs Microsoft.

Pour plus d'informations, consultez le guide utilisateur de votre compilateur.

EXERCICES SECTION 1.1.1

Exercice 1.1: Passez en revue la documentation de votre compilateur et déterminez quelle convention de nommage de fichier il utilise.

Exercice 1.2: Modifiez le programme pour retournez **-1**. Une valeur de retour -1 est souvent traitée comme un indicateur que le programme a échoué. Recompilez et exécutez votre programme pour voir comment votre système traite une indication de panne du *main*.

1.2 Un premier regard sur les Entrées/Sorties

Le langage C++ ne définit aucune instruction pour effectuer une entrée ou une sortie (Input/Output).Au lieu de cela, le C++ inclut une bibliothèque standard étendue qui fournit des IO (et de nombreuses autres fonctionnalités).