Dans ce chapitre, vous allez entrer enfin dans le vif du sujet et écrire vos premiers programmes en C++. Le but sera d'avoir un aperçu du processus de compilation, d'afficher des messages et de faire quelques calculs simples.

Programme C++ minimal

Créer une application en C++ n'est pas très compliqué. Commençons par voir un code C++ minimal, qui permet de créer un programme qui ne fait rien. Même s'il ne fait rien, nous allons pouvoir aborder quelques notions importantes.

Tester l'environnement de compilation en ligne

Pour commencer, revoyons les étapes pour lancer un programme sur l'éditeur en ligne que vous allez utiliser dans ce cours (vous avez déjà réalisé cette procédure dans le chapitre Comment réaliser les exercices de ce cours ? pour tester les codes d'exemple). Cliquez sur le lien suivant, avec le bouton droit de la souris et choisissez “Ouvrir le lien dans un nouvel onglet” : Tester le code C++ minimal. Ce lien permet d'ouvrir un environnement de développement en ligne, qui contient un éditeur pour écrire le code C++ et un compilateur pour transformer votre code C++ en programme exécutable.

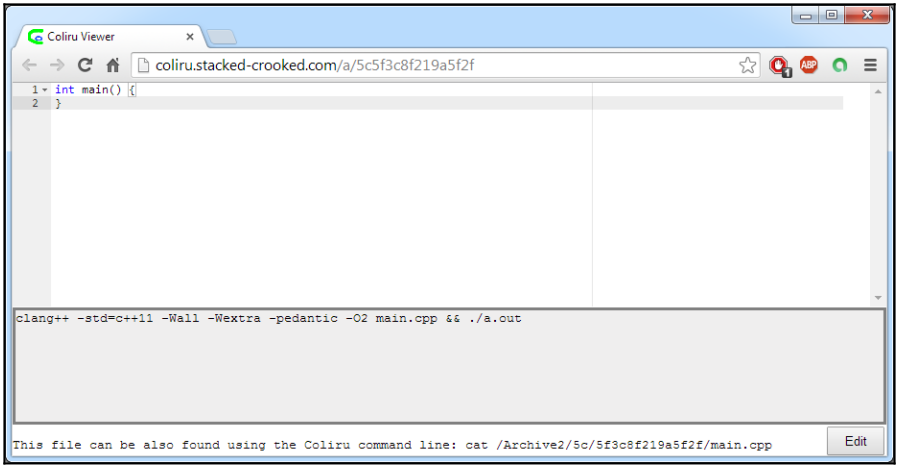
Vous pouvez également copier le code suivant dans un éditeur en ligne (Coliru, IdeOne ou autre), dans n'importe quel éditeur installé sur votre ordinateur (Qt Creator, Visual C++, Code::Block ou autre) ou dans un simple éditeur de texte.

main.cpp

```
int main() {
}
```

Dans le début de ce cours, vous pouvez utiliser un éditeur en ligne, ce qui évite l'installation, dans un premier temps, d'un environnement de développement. Mais, bien sûr, la création d'un programme C++ professionnel nécessitera d'installer de nombreux outils pour faciliter le développement et apporter des garanties à la qualité du logiciel. Cela sera détaillé dans la suite de ce cours.

La nouvelle fenêtre ressemble à l'image suivante :



Mettre à jour les images pour utiliser le C++14 :

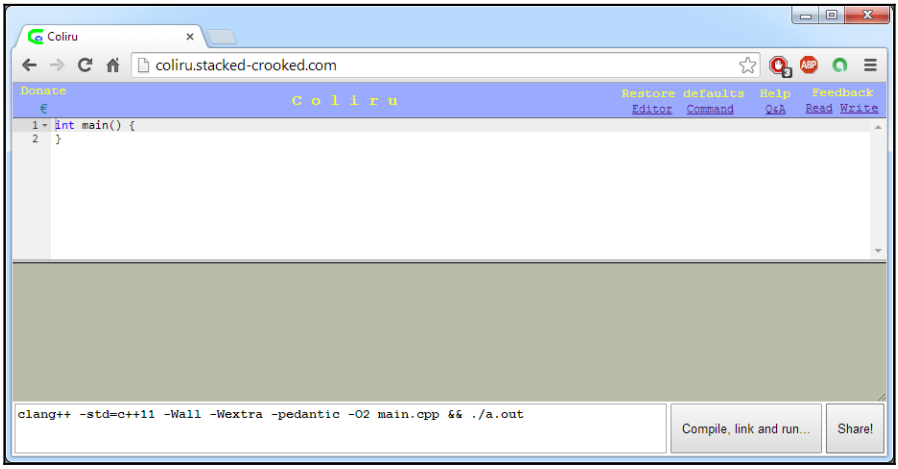
```
clang++ main.cpp -std=c++14 -Wall -Wextra -pedantic -O2 &&
./a.out
```

Cette fenêtre se décompose en deux parties. De haut en bas :

- le code du programme ;
- le résultat de la compilation et l'exécution.

Pour le moment, vous ne pouvez pas modifier ce code, cette fenêtre est en fait une simple archive du code qui a été créé pour ce cours. Pour pouvoir modifier et tester ce code, vous devez cliquer sur le bouton “Edit” en bas à droite de la fenêtre.

L'aspect de le fenêtre change pour ressembler à l'image suivante :

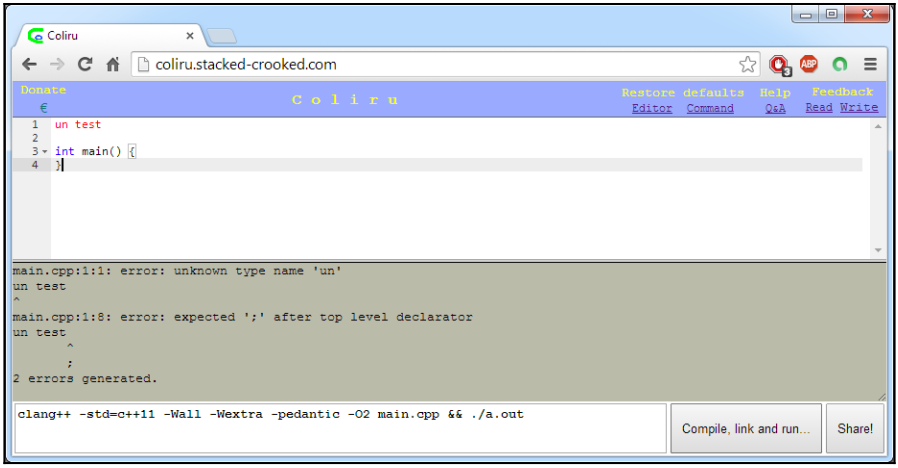


Cette nouvelle fenêtre contient maintenant trois parties. De haut en bas :

- Dans la partie du haut, l'éditeur de code, qui contient le code C++ de votre programme. Contrairement à l'étape précédente, vous pouvez maintenant modifier ce code en cliquant dessus et en tapant le code sur le clavier.
- La partie centrale grise, le résultat de la compilation et de l'exécution du programme. Pour le moment, comme vous n'avez pas lancé le programme, cette partie est vide.
- Dans la dernière partie, les instructions pour la compilation et l'exécution du programme.

Pour lancer la compilation et l'exécution, il vous suffit de cliquer sur le bouton “Compile, link and run...” en bas à droite. Si vous le faites, vous verrez que l'éditeur change de couleur quelques instants, mais rien ne change dans la partie centrale grise.

La raison est simple : le code minimal donné ne fait rien et donc rien ne s'affiche, ce qui donne l'impression que rien ne se passe. Mais faites une petite modification du code (n'importe quoi, c'est pour tester) et relancez la compilation, vous verrez des messages s'afficher dans la partie grise :



Même sans comprendre ce qui se passe, vous pouvez voir un mot dans la partie grise qui n'est pas compliqué à interpréter : “error”. Le compilateur a analysé le code C++ fourni et a déterminé qu'il n'est pas valide. Il indique donc ce qu'il n'a pas compris (vous verrez par la suite, le compilateur est un ami qui va beaucoup vous aider en indiquant les erreurs que vous faites. Il faudra juste apprendre à le comprendre).

Ce résultat est normal, puisque l'on a écrit n'importe quoi dans le code ! Vous allez apprendre à écrire du code C++ correct dans la suite de ce cours, mais avant cela, vous devez comprendre les bases de la compilation d'un code C++ et de l'exécution d'un programme.

La compilation d'un code C++

Fondamentalement, une application est une suite d'instructions donnée au processeur de votre ordinateur, qui lui indique les tâches qu'il doit accomplir. Un processeur ne connait qu'un seul langage, appelé langage machine, spécifique à chaque type de processeur. Un langage machine est difficilement compréhensible pour les humains et il est donc très peu utilisé. C'est pour cela que les programmes sont écrits dans des langages plus compréhensibles pour les humains, appelés *langages de programmation* (le C++ est un exemple d'un tel langage).

Le problème est que le processeur est incapable de comprendre autre chose que le langage machine. Il faut donc passer par une étape qui convertit un code écrit dans un langage de programmation en programme en langage machine.

La compilation est le processus qui transforme le code C++ en programme exécutable par un ordinateur.

Cette étape est donc indispensable pour tester vos programmes écrits en C++. La compilation est en fait constituée de plusieurs étapes (qui seront détaillées dans un prochain chapitre), chaque étape nécessitant l'utilisation d'un programme dédié. Vous verrez également dans un prochain chapitre différents outils de compilation, comment les installer et les utiliser. C'est pour éviter les problèmes liés à l'installation et l'utilisation de ces outils que vous utilisez dans un premier temps dans ce cours un environnement de compilation en ligne, qui possède déjà tous ces outils.

Classifications historiques des langages

Historiquement, les langages étaient classifiés selon plusieurs critères. Par exemple, on distinguait les langages compilés et les langages interprétés. Ou les langages de haut niveau et bas niveau.

Avec un **langage compilé**, les étapes de compilation et d'exécution sont séparées dans le temps (comme vous venez de le voir pour le C++) alors que pour un **langage interprété**, la conversion en langage machine a lieu en même temps que l'exécution.

Un langage est dit de **bas-niveau** lorsqu'il est proche du langage machine, un langage est **haut-niveau** lorsqu'il est proche du langage humain. (les définitions de "haut" et "bas-niveau" ne sont pas formalisées et dépendent des auteurs. Peu importe ici, lire la suite).

De nos jours, cette distinction n'est plus forcément pertinente, beaucoup de langages peuvent être utilisés via compilation et interprétation, voire même un mélange des deux.

Revenons sur la compilation de notre code C++ minimal. Dans l'éditeur

utilisé, la compilation est réalisée en suivant les instructions données dans la partie du bas de la fenêtre. Cette partie (que vous pouvez également modifier) contient les instructions suivantes :

```
clang++ -std=c++11 -Wall -Wextra -pedantic -O2 main.cpp &&
./a.out
```

Ces instructions sont en fait des commandes Linux (le serveur utilisé par l'éditeur tourne sous Linux - mais les commandes sur Windows ou Mac OS X sont très proches). Si vous ne savez pas utiliser Linux en ligne de commande, ce n'est pas très grave (pour le moment... un développeur doit quand même avoir quelques bases sur l'utilisation des lignes de commandes).

Vous pourrez utiliser par la suite des outils qui se chargeront d'appeler ces instructions pour vous, de façon transparente. Mais gardez quand même en mémoire que quel que soit l'outil que vous utiliserez, celui-ci ne fera rien d'autre que d'appeler ces instructions, comme vous pourriez le faire vous-même (certains préfèrent d'ailleurs compiler manuellement leurs programmes en utilisant directement des lignes de commande).

Les instructions suivantes réalisent en fait deux tâches : lancer la compilation puis lancer l'exécution. Chaque étape est séparée par l'opérateur `&&` ou par un retour à la ligne. Ainsi, le code précédant peut également s'écrire :

```
clang++ main.cpp -std=c++14 -Wall -Wextra -pedantic -O2
./a.out
```

Remarque : dans l'éditeur, pour ajouter un retour à la ligne dans la partie du bas, il faut appuyer sur les touches `Shift` + `Entrée` du clavier. Appuyer simplement sur la touche `Entrée` lance la compilation et l'exécution.

La première ligne, qui nous intéresse pour le moment, permet de lancer la compilation. Cette ligne se décompose de la façon suivante :

```
<compilateur>            = clang++
<fichier à compiler>     = main.cpp
```

```
<options de compilation> = -std=c++14 -Wall -Wextra
-pedantic -O2
```

Voyons ces éléments en détail. La commande `clang++` permet de lancer un programme de compilation (le compilateur - qui convertit le code C++ en langage machine) appelé Clang. Ce compilateur est gratuit et l'un des plus à jour pour le support du C++. L'éditeur Coliru permet d'utiliser un autre compilateur appelé GCC, que vous pouvez utiliser en remplaçant `clang++` par `g++`.

La valeur `main.cpp` est le nom du fichier à compiler. Par défaut, le code dans l'éditeur Coliru est enregistré dans ce fichier, il faut donc indiquer à Clang qu'il doit compiler ce fichier.

Pour terminer, les options de compilation permettent de spécifier à Clang comment il doit compiler le code. Il existe plusieurs centaines d'options de compilation, il ne sera pas possible de toutes les détailler dans ce cours. Pour résumer les options utilisées ici, sachez que l'option `-std=c++14` permet d'activer le support de la norme la plus récente du C++, les options `-Wall -Wextra -pedantic` permettent de vérifier de nombreuses erreurs de programmation dans vos codes et l'option `-O2` permet d'optimiser le programme généré.

Il est important de connaître le processus et les options de compilation, cela sera détaillé dans la suite du cours. Cependant, pour les codes d'exemple et les exercices donnés dans ce cours, les instructions de compilation seront données, vous n'aurez pas besoin de les modifier.

Pour en savoir plus sur ces compilateurs, vous pouvez consulter les pages correspondantes de Wikipédia : Clang et GCC. Vous pouvez également consulter les sites officiels : Clang et GCC, en particulier les documentations pour connaître les options de compilation utilisables.

Notez aussi qu'il existe un dernier compilateur très utilisé, celui fourni par Microsoft : Visual Studio. Ce n'est pas le compilateur qui fournit le meilleur support du C++14 - bien qu'ils aient fait de grands progrès sur ce point - mais il reste incontournable sur Windows.

L'exécution d'un programme C++

Par défaut, le programme généré par la compilation s'appelle `a.out`. L'instruction `./a.out` permet donc de lancer son exécution. (Pour rappel, le code actuel ne fait rien, l'exécution ne produit aucun message dans la fenêtre de sortie de Coliru pour le moment).

Le schéma suivant résume l'ensemble des étapes de compilation et d'exécution :

Il est possible de changer le nom du programme généré en utilisation l'option `-o` (pour *output*, qui signifie *sortie*), par exemple :

```
clang++ main.cpp -o mon_programme -std=c++14 -Wall -Wextra
-pedantic -O2
./mon_programme
```

La fonction main

Revenons maintenant sur le code C++ de notre programme. Ce code minimal définit une fonction appelée `main` (qui signifie “principal” en anglais), qui ne fait rien :

```
int main() {
}
```

Pour le moment, vous ne savez pas créer une fonction en C++ et ce code peut vous paraître obscur. Savoir créer des fonctions est un point important de votre apprentissage du C++, mais il n'est pas possible de voir cela en détail dans un seul chapitre. Cela sera détaillé par la suite.

Mais pas de panique. Maîtriser les fonctions n'est pas nécessaire pour

créer une fonction `main`. En effet, la façon d'écrire une fonction `main` (on parle de sa *signature*) est définie par la norme C++. Donc, en pratique, vous devrez simplement copier à l'identique cette fonction pour créer un programme C++ de base.

Fondamentalement, un programme est simplement une suite d'instructions que l'on donne à l'ordinateur, pour réaliser une tâche. Tout comme il est souvent possible de décomposer une tâche en plusieurs sous-tâches plus simples, un programme sera décomposé en fonctions, qui sont des suites d'instructions destinées à réaliser une sous-tâche particulière.

Une fonction pourra appeler d'autres fonctions, qui pourront à leur tour appeler d'autres fonctions, jusqu'à ce que l'ensemble des fonctions appelées réalisent le travail demandé.

La fonction `main` est un peu particulière puisqu'elle est appelée par le système d'exploitation. Elle est obligatoire dans un programme. Si vous ne créez pas de fonction `main` ou si vous en créez plusieurs, le système d'exploitation ne saura pas comment lancer votre application et produira une erreur. Par contre, vous ne pouvez pas vous-même appeler cette fonction, seul le système peut l'appeler.

Vous apprendrez par la suite à créer des fonctions et leurs syntaxes en détail, mais pour comprendre la fonction `main`, voici quelques explications. Comme indiqué ci-dessus, une fonction est une suite d'instructions. Elle peut être appelée par une autre fonction. L'appel d'une fonction se déroule en trois étapes :

- le code appelant appelle la fonction à partir de son nom, en lui transmettant des informations si besoin ;
- la suite d'instructions correspondant à la fonction appelée est exécutée ;
- la fonction se termine en transmettant éventuellement une réponse à la fonction appelante.

Ces trois étapes apparaissent dans la déclaration d'une fonction, avec la syntaxe suivante :

```
InformationsRetournées NomDeLaFonction(InformationsEnvoyées)
{
    Suite d'instructions
}
```

La déclaration d'une fonction commence par définir les informations retournées par la fonction (encore appelé *paramètre de retour* de la fonction) lorsqu'elle se termine. Il ne peut y avoir qu'un seul paramètre de retour de fonction (mais ce point n'est pas limitant, puisqu'un paramètre peut contenir plusieurs informations). Lorsqu'une fonction ne retourne aucune information, le paramètre de retour est `void`. Dans le cas de la fonction `main`, celle-ci retourne toujours une information de type `int`, qui signifie un nombre entier (*integer* en anglais).

Vient ensuite le nom de la fonction. Le nommage des fonctions (et plus généralement de tous les éléments que vous allez pouvoir définir dans un programme C++) suit des règles spécifiques, qui seront détaillées plus tard. Pour faire simple, un nom est une suite de caractères alphanumériques (n'importe quelle lettre de l'alphabet sans accentuation et n'importe quel chiffre, sauf en première position), avec des majuscules et/ou des minuscules. Certains “mots” sont réservés par la norme C++, il vous est interdit de les utiliser. Il est également habituel d'écrire ses programmes en anglais, je vous conseille de faire de même.

On voit ici que l'écriture d'un programme C++ ne suit pas simplement des règles imposées par le langage (la norme), mais également les habitudes et bonnes pratiques mises en place avec le temps. Suivre ces règles permet :

- de fournir un cadre commun à tous les développeurs C++, pour faciliter la communication ;
- de simplifier la création du code, en proposant une approche “classique” de faire les choses ;
- d'apporter des garanties sur la qualité du code, pour qu'il fasse exactement ce que l'on attend de lui.

Il est classique de voir des débutants ne pas suivre ces bonnes pratiques durant leur phase d'apprentissage et justifier cela en disant que lorsqu'ils travailleront sur de vrais projets, ils suivront ces règles. C'est une mauvaise idée de faire comme cela : une pratique (bonne ou mauvaise) acquise durant sa phase d'apprentissage sera très dure à faire évoluer. Il est important de pratiquer correctement dès le début (même si cela peut sembler faire perdre du temps au début, le temps d'acquérir des habitudes de programmation sera largement rattrapé).

Les informations d'entrée (ou *paramètres d'entrée*) sont définies à la suite du nom de la fonction, entre parenthèses. Lorsqu'il n'y a pas de paramètres d'entrée, on met simplement les parenthèses sans rien dedans (les parenthèses doivent toujours être présentes). Remarquez qu'il ne peut y avoir qu'un seul paramètre de retour de fonction, mais que l'on peut avoir plusieurs paramètres d'entrée.

Pour terminer, la partie la plus importante : la suite d'instructions, dans un bloc de code définit par des accolades `{` et `}`. Chaque instruction se termine par un point-virgule. Dans le code d'exemple de ce chapitre, le bloc d'instructions est vide, il n'y a que les accolades, le programme ne fait rien (mais vous verrez dès le prochain chapitre les bases pour écrire des instructions).

En fait, il existe plusieurs *signatures* pour la fonction `main` (i.e. plusieurs façons différentes d'écrire cette fonction). Cela permet en particulier au système d'envoyer des informations lors du lancement du programme, que vous pourrez utiliser dans votre code. Pour le moment, vous ne savez pas encore comment traiter ces informations, donc il n'est pas nécessaire de détailler ce point.

Mise en forme du code

Vous avez maintenant les informations de base pour comprendre ce premier programme C++ (qui ne fait rien). Pour terminer ce chapitre, deux points importants, sur la présentation du code.

Dans un code C++, les espaces et les retours à la ligne ne sont pas pris en compte dans la compilation (sauf bien sûr si vous accolez deux termes ensembles, si vous écrivez `intmain` sans espace, votre compilateur vous insultera copieusement). Vous pouvez donc ajouter des espaces et des retours à la ligne de façon à rendre votre code le plus lisible possible, sans que cela ne change votre programme.

Un point important à ne pas oublier : un code sera plus souvent lu qu'il n'est écrit ou modifié. Il faut donc privilégier la qualité de lecture d'un code, plutôt que d'essayer de gagner du temps à l'écriture (même pour un simple code de test ou d'apprentissage). Présenter correctement un code permet de gagner du temps sur le long terme.
Ainsi, le programme d'exemple peut s'écrire selon les façons suivantes :

```
int main(){}
```

```
int
main()
{
}
```

```
int     main     (     )     {     }
```

Il faut trouver un compromis entre la concision (écrire un code qui sera le plus compact possible) et avoir un code aéré. Il est habituel de définir des règles d'écriture du code, pour faciliter la lecture et permettre à plusieurs personnes de comprendre le code des autres développeurs. Il existe plusieurs conventions pour ces règles, à vous de choisir celles qui vous conviennent.

Peu importe les règles de codage que vous choisissez, le plus important est surtout d'avoir des règles et de les respecter.

En particulier, un point important est le respect de l'indentation. L'indentation correspond aux espaces placés en début d'une ligne. Le début des lignes doit être aligné selon son niveau hiérarchique. Si on

écrit la hiérarchie suivante :

```
niveau 1
niveau 2
niveau 3
niveau 3
niveau 2
niveau 3
niveau 1
```

Il est assez difficile de visualiser facilement à quel niveau hiérarchique correspond chaque ligne. Si on utilise une indentation pour distinguer chaque niveau, le code est beaucoup plus lisible :

```
niveau 1
    niveau 2
        niveau 3
        niveau 3
    niveau 2
        niveau 3
niveau 1
```

Vous trouverez des exemples de styles d'indentation du code dans la page de Wikipédia correspondante. Dans ce cours, j'utiliserais le style K&R, avec une indentation de quatre espaces.
Il existe des outils permettant de mettre en forme le code et vérifier que la présentation du code respecte les règles que vous avez fixé. Ces outils seront présentés dans la suite de ce cours.

Commentaires du code

Pour comprendre un code, les premières informations que le lecteur verra sont les noms que l'on donne. Appeler par exemple ses fonctions `f1`, `f2` et `f3` n'aide pas du tout à comprendre à quoi servent ces fonctions. Par contre, appeler ces fonctions `add` (addition), `save` (enregistrer) ou `reset` (remettre à zéro) permet aux lecteurs d'avoir une idée de leur rôle.

Il est important de prendre le temps de nommer correctement les choses, ce n'est pas une perte de temps.

Cependant, il n'est pas toujours possible de trouver des noms significatifs (tout au moins, sans faire des noms de 100 caractères). Dans ce cas, il est possible d'ajouter des commentaires dans le code, qui seront ignorés par le compilateur (et donc ne changeront par le comportement du programme généré) et seront destinés uniquement aux développeurs.

Il existe deux formes de commentaire. Les commentaires sur une ligne et les commentaires sur plusieurs lignes. Les commentaires peuvent être placés n'importe où dans votre code. Pour écrire un commentaire sur une ligne, vous devez utiliser deux barres obliques `/□/` suivies du commentaire. Pour un commentaire sur plusieurs lignes, il faut commencer le commentaire par une barre oblique puis un astérisque `/*` et terminer le commentaire par un astérisque puis une barre oblique `*/`.

```
// un commentaire sur une ligne

int main() { // un commentaire en fin d'une ligne
    /* un
    commentaire
    sur
    plusieurs
    lignes */
}
```

Exercices

Compilation et exécution

Dans les commandes de compilation et d'exécution, il est possible d'utiliser la commande `echo` suivie d'un texte entre guillemets pour afficher un message.

1. Modifier les instructions de compilation pour afficher un message avant la compilation et un autre après l'exécution :

```
**** Compilation du programme C++ ****
```

```
...
**** Exécution du programme C++ ****
...
```

2. Modifier le code de compilation pour compiler deux fois le programme, avec les options d'avertissement et sans les options d'avertissement :

```
**** Compilation du programme C++ avec les avertissements ****
...
**** Compilation du programme C++ sans les avertissements ****
...
**** Exécution du programme C++ ****
...
```

3. Modifier le code de compilation et d'exécution pour nommer le programme “mon_programme” au lieu de “a.out”.

La fonction main

En fait, en plus des deux syntaxes possibles pour la fonction `main`, il existe une troisième syntaxe.

1. Trouvez dans la documentation du langage C++ la page correspondant à la fonction `main`.

2. Trouvez ensuite la troisième syntaxe possible pour la fonction `main`.

3. Il est courant d'écrire au début de chaque code la licence d'utilisation du code, les coordonnées de l'auteur, la date de modification et d'autres informations utiles. Modifier le code de la fonction `main` pour ajouter ces informations sous forme de commentaires.

Chapitre précédent	Sommaire principal	Chapitre suivant

Cours, C++