

Le guide du C++ moderne

de débutant à développeur

par
Benoît Vittupier
Mehdi Benharrats
avec la contribution de Luc Hermitte
et de Zeste de Savoir

Le guide du C++ moderne • de débutant à développeur
par Benoît Vittupier, Mehdi Benharrats, avec la contribution de Luc Hermitte,
et de Zeste de Savoir

ISBN (imprimé) : 978-2-8227-1176-0

Copyright © 2025 Éditions D-Booker
Tous droits réservés

Conformément au Code de la propriété intellectuelle, seules les copies ou reproductions strictement réservées à l'usage privé du copiste et non destinées à une utilisation collective ainsi que les analyses et les courtes citations dans un but d'exemple et d'illustration sont autorisées. Tout autre représentation ou reproduction, qu'elle soit intégrale ou partielle, requiert expressément le consentement de l'éditeur (art L 122-4, L 122-5 2 et 3a).

Publié par les Éditions D-Booker, 229 rue Solférino, 59000 Lille
www.d-booker.fr
contact@d-booker.fr

Les exemples (téléchargeables ou non), sauf indication contraire, sont propriété des auteurs.

Conception de la couverture : d'après une création de Marie Van Der Marlière
(www.marie-graphiste.com)

Visuel de couverture : Réalisé par putracetol (#22795181 sur VectorStock.com)

Mise en page : générée sous Calenco avec des XSLT développées par la société
NeoDoc (www.neodoc.biz)

Diffusion : AFPU-D (<https://www.afpu-diffusion.fr>)

Distribution : DILISCO (<https://dilisco-diffusion.centprod.com>)

Dépôt légal : Novembre 2025

Édition : 2

Version : 2.0

Table des matières

Bienvenue !	1
1. Codes sources des exemples	1
2. Remerciements	2
1. C'est décidé, je m'y mets !	3
1.1. Le C++, qu'est-ce que c'est ?	3
1.2. Pourquoi apprendre le C++ ?	4
1.3. Développer, un métier à part entière	5
1.4. Votre part du travail	5
1.5. Les mathématiques, indispensables ?	5
1.6. L'anglais, indispensable ?	6
1.7. La documentation	6
1.8. Récapitulons !	7
Le début du voyage	9
2. Le minimum pour commencer	11
2.1. Des outils en ligne	11
2.2. Des outils plus poussés	13
Visual Studio 2022, l'outil tout-en-un pour Windows	14
Visual Studio Code, la solution portable	18
2.3. Un mot concernant Windows	23
2.4. Récapitulons !	24
3. Rencontre avec le C++	25
3.1. Compilons notre premier programme	25
3.2. Démystifions le code	26
Utiliser des fonctionnalités déjà codées	26
Le point d'entrée	26
Voici mes instructions...	26
3.3. Les commentaires	27
3.4. Variantes pré-C++23	28
3.5. Récapitulons !	30
4. Tout ça est bien variable	31
4.1. Les littéraux	31
Les chaînes de caractères	31
Les caractères	32

Les caractères spéciaux	33
Les nombres	36
4.2. Les variables	38
Comment créer des variables en C++ ?	38
Un peu de constance, voyons !	42
Manipulation de variables	43
4.3. Simple déduction ?	46
Avec <i>constexpr</i>	47
Avec <i>std::string</i>	47
Quel intérêt ?	48
4.4. Les entrées	48
4.5. Récapitulons !	50
5. Le conditionnel conjugué en C++	51
5.1. Les booléens	51
5.2. <i>if</i> – si, et seulement si...	53
5.3. <i>else</i> – sinon...	55
5.4. <i>else if</i> – la combinaison des deux précédents	57
5.5. Conditions imbriquées	58
5.6. [Pratique] Gérer les erreurs d'entrée • Partie I	59
5.7. La logique booléenne	60
AND – tester si deux conditions sont vraies	60
OR – tester si au moins une condition est vraie	61
NOT – tester la négation	62
5.8. Tester plusieurs expressions	63
5.9. Entraînez-vous !	66
Une pseudo-horloge	66
Score	66
XOR – le OU exclusif	67
5.10. Récapitulons !	67
6. Des boucles qui se répètent, répètent, répètent...	69
6.1. <i>while</i> – tant que...	69
6.2. Entraînez-vous !	71
Une laverie	71
PGCD	72
Somme de nombres de 1 à <i>n</i>	72
6.3. <i>do while</i> – répéter ... tant que	72
6.4. [Pratique] Gérer les erreurs d'entrée • Partie II	73
6.5. <i>for</i> – une boucle condensée	74

6.6. Boucles imbriquées	75
6.7. Convention de nommage	76
6.8. Contrôler plus finement l'exécution de la boucle	76
<i>break</i> – je m'arrête là	76
<i>continue</i> – saute ton tour !	78
6.9. Récapitulons !	78
7. Au tableau !	79
7.1. Un tableau c'est quoi ?	79
7.2. <i>std::vector</i> – mais quel dynamisme !	79
Déclarer un <i>std::vector</i>	80
Manipuler les éléments d'un <i>std::vector</i>	81
7.3. Entraînez-vous !	88
Calcul de moyenne	88
Minimum et maximum	89
Séparer les pairs des impairs	89
Compter les occurrences d'une valeur	89
7.4. <i>std::array</i> – le tableau statique	89
Déclarer un <i>std::array</i>	89
Accéder aux éléments	90
Connaître la taille	91
Vérifier si le tableau est vide	92
7.5. <i>std::string</i> – un type qui cache bien son jeu	92
Accéder à un caractère	92
Premier et dernier caractère	93
Vérifier qu'une chaîne est vide	93
Ajouter ou supprimer un caractère à la fin	94
Supprimer tous les caractères	94
Boucler sur une chaîne	95
7.6. Récapitulons !	95
8. Découpons le code – les fonctions	97
8.1. Les éléments de base d'une fonction	97
Une fonction, c'est quoi ?	97
Une fonction incontournable	99
Les composants d'une fonction	99
8.2. Entraînez-vous !	102
Afficher un rectangle	103
Distributeur d'argent	103
Parenthésage	103

8.3. Quelles sont vos références ?	104
La problématique	104
Les références	105
Paramètres de fonctions et références	106
Valeur de retour et références	109
Un mot sur la déduction de type	110
Tableaux et références	111
8.4. Nos fonctions sont surchargées !	112
8.5. [Pratique] Gérer les erreurs d'entrée • Partie III	114
8.6. Dessine-moi une fonction	115
8.7. Récursivité : L'art de se rappeler... soi-même !	116
PGCD récursif	117
La suite de Fibonacci	118
8.8. Récapitulons !	118
9. Déployons la toute puissance des conteneurs	119
9.1. La pièce maîtresse : les itérateurs	119
Déclaration d'un itérateur	120
Début et fin d'un conteneur	121
Accéder à l'élément pointé	121
Se déplacer dans une collection	122
const et les itérateurs	125
Itérer depuis la fin	127
Utilisation conjointe avec les conteneurs	128
9.2. Des algorithmes à gogo !	129
std::find – trouver un certain élément	129
std::count – compter les occurrences d'une valeur	130
std::sort – trier une collection	131
std::reverse – inverser l'ordre des éléments d'une collection	133
std::remove – suppression d'éléments	133
std::search – rechercher un sous-ensemble dans un en-	
semble	134
std::equal – vérifier l'égalité de deux ensembles	136
Et tant d'autres	137
9.3. Personnalisation à la demande	137
Le prédicat, pivot de la personnalisation des algorithmes	137
Trier un ensemble par ordre décroissant	137
Somme et produit	138
Et d'autres encore	139
9.4. Entraînez-vous !	139

Palindrome	139
Un <code>std::set</code> fait maison	140
Couper une chaîne	140
9.5. Récapitulons !	140
10. Des flux dans tous les sens	143
10.1. Avant toute chose	143
Prendrez-vous une extension ?	143
Vois sur ton chemin...	144
Pas d'interprétation, on est des brutes	145
10.2. <code>std::ofstream</code> — écrire dans un fichier	146
Ouvrir le fichier	146
Écrire	146
Ouvrir sans effacer	147
10.3. <code>std::ifstream</code> — lire dans un fichier	148
Ouvrir le fichier	148
Lire	149
Tout lire	151
10.4. Encore plus de flux !	151
Un flux c'est quoi ?	152
Des flux... même avec des chaînes !	152
Un <i>buffer</i> dites-vous ?	154
10.5. Aller plus loin avec les fichiers	156
10.6. Entraînez-vous !	158
Statistiques sur des fichiers	158
Lire un fichier CSV simple	158
Combiner des fichiers	159
10.7. Récapitulons !	159
On passe la deuxième !	161
11. Erreur, erreur, erreur...	163
11.1. L'utilisateur est un idiot	163
11.2. À une condition... ou plusieurs	164
Contrats assurés par le compilateur	165
Vérifions nous aussi	165
11.3. Le développeur est un idiot	165
Préconditions et assertions	168
11.4. Les tests unitaires à la rescousse	169
Pourquoi tester ?	169

Un test unitaire, qu'est-ce que c'est ?	169
Écrire des tests unitaires	170
Les tests unitaires et les postconditions	172
11.5. [Pratique] Gérer les erreurs d'entrée • Partie IV	172
11.6. L'exception à la règle	173
La problématique	173
C'est quoi une exception ?	174
Lancement des exceptions dans 3, 2, 1...	175
Attends que je t'attrape !	176
Attrapez-les tous !	178
Un type pour les gouverner tous et dans le catch les lier	179
11.7. [Pratique] Gérer les erreurs d'entrée • Partie V	180
11.8. Récapitulons !	181
12. Des fonctions somme toute lambdas	183
12.1. Une lambda, c'est quoi ?	183
12.2. Pourquoi a-t-on besoin des lambdas ?	183
12.3. Syntaxe	184
Quelques exemples simples	184
12.4. Entraînez-vous !	187
Vérifier si un nombre est négatif	187
Tri par valeur absolue	187
<i>string_trim</i> – supprimer les espaces au début et en fin de chaîne	188
12.5. On va stocker ça où ?	188
12.6. Paramètres génériques	189
12.7. [Pratique] Gérer les erreurs d'entrée • Partie VI	190
12.8. [Pratique] Gérer les erreurs d'entrée • Partie VII	190
12.9. Capture en cours...	191
Capture par valeur	191
Capture par référence	192
12.10. Récapitulons !	194
13. Formez les rang(e)s !	195
13.1. Un exemple concret	195
13.2. Qu'est-ce qu'un <i>range</i> et une <i>view</i> ?	197
13.3. Réécrivons le passé	197
<i>std::ranges::find</i> – trouver un certain élément	197
<i>std::ranges::count</i> – compter les occurrences d'une valeur	198
<i>std::ranges::sort</i> – trier une collection	199

<code>std::ranges::reverse</code> – inverser l'ordre des éléments d'une collection	199
<code>std::ranges::equal</code> – vérifier l'égalité de deux ensembles	200
13.4. Pour quelques exemples de plus	200
13.5. Récapitulons !	203
14. Envoyez le générique !	205
14.1. Les fonctions, c'est <i>auto</i> -matique	205
14.2. Quel beau modèle !	206
La bibliothèque standard : une fourmilière de modèles	209
Un point de vocabulaire	209
14.3. Instanciation explicite	210
14.4. [Pratique] Gérer les erreurs d'entrée • Partie VIII	211
14.5. Ce type-là, c'est un cas !	211
14.6. Quand les lambdas s'en mêlent	213
14.7. Récapitulons !	214
15. De nouvelles structures de données	215
15.1. <code>struct</code> – un agrégat de données	215
Stockage d'informations personnelles	215
Analyse de la méthode	216
Introduction aux structures	216
Et maintenant, structurons !	218
15.2. <code>std::tuple</code> – une collection hétérogène	219
Déclaration	219
Accès aux éléments	219
Renvoyer plusieurs valeurs	221
Un outil pour stocker sans étiqueter	223
15.3. <code>std::unordered_map</code> – une table associative	224
Un dictionnaire de langue Zestienne	225
Toujours plus de clés	227
Cette clé est unique !	228
Cherchez un élément	230
Un exemple concret	231
15.4. Un peu d'ordre, voyons !	232
15.5. Une longue énumération	234
15.6. Récapitulons !	235
16. Reprendrez-vous un peu de sucre syntaxique ?	237
16.1. Ce type est trop long !	237
16.2. En pleine décomposition	238

16.3. La surcharge d'opérateurs	240
Le problème... ..	240
...et la solution	242
Mise en pratique	242
Et la bibliothèque standard ?	253
Sur le bon usage de la surcharge	254
16.4. Récapitulons !	254
17. [Pratique] Un gestionnaire de discographie	255
17.1. Cahier des charges	255
Ajout de morceaux	255
Affichage de la discographie	256
Enregistrement et chargement d'une discographie	258
Fermeture du programme	258
Gestion des erreurs	258
Dernières remarques	258
17.2. Guide de résolution	259
Analyse des besoins	259
Attaquons le code !	260
17.3. Code complet	264
17.4. Conclusion et pistes d'améliorations	275
17.5. Récapitulons !	276
18. Découpons du code — les fichiers	277
18.1. C++, un langage très modulable	277
Créer son propre module	278
Des modules bien rangés	283
Diviser pour mieux régner	285
18.2. Les <code>#include</code> , à l'époque où tout était simple... ou pas !	287
Le fichier d'en-tête	287
Le fichier source	288
Créons un lien	289
La sécurité, c'est important	290
18.3. À projet moderne, solution moderne	292
18.4. [Pratique] Découpons notre programme !	293
18.5. Les avantages du découpage en fichiers	296
18.6. Le cas des templates	297
18.7. Finis les conflits, vive les namespaces	300
18.8. Récapitulons !	302

Interlude : être développeur	303
19. Un coup d'œil dans le rétro	305
19.1. Au-delà du code	305
19.2. Tout est une question de paradigme	306
Le paradigme impératif	306
Le paradigme fonctionnel	307
Le paradigme générique	307
La programmation par contrat	308
19.3. L'art de la conception	308
19.4. De la persévérance avant tout	309
19.5. Récapitulons !	310
20. Mais où est la doc ?	311
20.1. Lire une page de doc	312
vector — retour sur le plus célèbre des conteneurs	314
Les algorithmes	316
Les chaînes de caractères au grand complet	317
20.2. Entraînez-vous	318
Remplacer une chaîne de caractères par une autre	318
Norme d'un vecteur	319
Nombres complexes	319
Transformations	320
20.3. Documenter son code avec Doxygen	320
Installation d'un outil dédié	320
Écrire la documentation	324
20.4. Quelques bonnes pratiques	329
Ce qu'il faut documenter	330
Définition ou implémentation ?	330
Rien ne vaut un exemple	331
Commentaire vs documentation	332
20.5. Récapitulons !	334
21. Compilation en cours...	335
21.1. Le préprocesseur	335
Inclure des fichiers	335
Conditions	336
Debug ou release ?	337
21.2. Les modules, pour une compilation améliorée	339

21.3. La compilation	340
Les templates	340
<i>constexpr</i>	341
La compilation à proprement parler	344
Une étape intermédiaire cachée	345
Influer sur la compilation	345
21.4. Le <i>linker</i>	351
Une question de symboles	352
21.5. Schéma récapitulatif	354
21.6. Récapitulons !	355
22. Chasse aux bugs !	357
22.1. Le principe	357
22.2. Un code d'exemple	358
22.3. Avec Visual Studio	358
Interface	359
Pas-à-pas	361
Point d'arrêt conditionnel	362
22.4. En ligne de commande avec gdb	364
Poser un point d'arrêt	364
Supprimer des points d'arrêt	365
Désactiver des points d'arrêt	366
Afficher l'état d'une variable	366
Pas-à-pas	366
Conditions	367
S'y retrouver dans le code	368
22.5. Avec Visual Studio Code	368
Interface	368
Pas-à-pas	370
Point d'arrêt conditionnel	371
22.6. Récapitulons !	372
23. Une foule de bibliothèques	373
23.1. Quelles bibliothèques choisir ?	373
23.2. Généralités	374
Statique ou dynamique ?	374
<i>Debug</i> ou <i>release</i> ?	375
32 ou 64 bits ?	375
Bibliothèques <i>header-only</i>	375
23.3. Installer Boost	376

GNU/Linux	376
Visual Studio	379
23.4. Installer SFML	382
GNU/Linux	383
Visual Studio	383
23.5. Vos bibliothèques en un clic (ou presque)	387
23.6. Récapitulons !	387
24. Construisons mieux avec CMake	389
24.1. CMake, l'ingénieur en chef de vos compilations	389
Visual Studio	390
Visual Studio Code	390
24.2. Notre premier projet avec CMake	391
24.3. Compilation multi-fichiers	398
24.4. Inclure des bibliothèques externes	400
Bibliothèques installées localement	400
Que ferait-on sans Internet ?	403
24.5. Aller plus loin avec CMake	406
24.6. Récapitulons !	407
25. Pour une poignée d'outils	409
25.1. git – sauvegarder et versionner son code	409
Installer git	410
Initialiser git	410
Créer un dépôt	410
Connaître l'état de ses fichiers	411
Ajouter un fichier	411
Valider les modifications	412
Voir l'historique	412
Bloquer certains fichiers	413
Aller plus loin	415
25.2. GitLab – partager son code	415
Création d'un projet	416
Synchroniser les modifications	417
Aller plus loin	418
25.3. Encore quelques outils	418
GitLab CI – de l'intégration continue	418
De vrais tests unitaires	418
CppCheck – vérification du code	419
StackOverflow – la réponse à quasi toutes les questions	420

25.4. Récapitulons !	420
----------------------------	-----

La programmation orientée objet 421

26. Premiers pas avec la POO 423

26.1. Le principe : des objets bien serviables	423
Penser en termes de données... ..	423
...ou de services ?	424
Exemple concret	424
26.2. Un peu de vocabulaire	425
L'objet	425
Le type	425
L'interface d'une classe	425
26.3. En C++, ça donne quoi ?	426
Penser services	426
Penser tests	426
Fonctions membres	427
Fonctions libres	428
Les attributs	428
26.4. Désolé, cet objet n'est pas modifiable	429
26.5. On ne fait pas d'exception	432
26.6. Découpage en fichiers	434
26.7. Entraînez-vous !	436
Cercle	436
Informations personnelles	437
26.8. Récapitulons !	437

27. Et qui va construire tout ça ? 439

27.1. Encapsulation et invariants	439
Cette classe, c'est open bar	439
Tout est question d'invariant	440
Encapsulons tout ça	441
Modifier sa visibilité	441
27.2. On cherche un constructeur	442
Le principe	442
La syntaxe	444
27.3. Constructeur par défaut	446
27.4. Soyons explicites	449
27.5. En toute amitié	452
Attributs publics	455

Accesseurs	455
La troisième, c'est la bonne	456
27.6. Entraînez-vous !	457
Un autre constructeur pour <i>Fraction</i>	457
Une pile	458
27.7. Récapitulons !	459
28. Une classe de grande valeur	461
28.1. Une histoire de sémantique	461
28.2. La sémantique de valeur, c'est quoi ?	463
28.3. Égalité	463
28.4. Le retour des opérateurs	464
Les opérateurs d'affectation intégrés	464
Du deux en un	466
Les opérateurs de manipulation des flux	467
Quelques bonnes pratiques	467
Les autres opérateurs	468
28.5. Copier des objets	468
Constructeur de copie	468
Opérateur d'affectation par copie	469
28.6. Récapitulons !	471
29. [Pratique] Entrons dans la matrice	473
29.1. Qu'est-ce qu'une matrice ?	473
29.2. Cahier des charges	475
Les services	475
Les détails d'implémentation	475
Les tests	476
29.3. Réalisation	480
Les services	480
Les détails d'implémentation	482
29.4. Code complet	489
29.5. Entraînez-vous !	496
Entiers infinis	496
Des polynômes	497
29.6. Récapitulons !	498
30. Classes à sémantique d'entité	499
30.1. Des objets uniques !	499
Conséquences sur l'interface	499
Un exemple	500

30.2. Des sous-types...	502
Exemple	502
Héritage et constructeurs	503
Une question de visibilité	506
30.3. ...substituables	508
Une histoire de manchots	510
Le principe de substitution de Liskov	511
Substitution dans la bibliothèque standard	511
Substitution et programmation par contrat	512
Héritage public et LSP	514
30.4. Un monde virtuel	516
Un problème... ..	516
...une solution	519
En interne	522
Faisons le point	525
30.5. Interfaces et classes abstraites	525
Des fonctions virtuelles vraiment pures	525
Classe abstraite et implémentation	527
30.6. La copie, une vraie source de problème	528
30.7. Récapitulons !	530
31. Ressources, indirections et handles	531
31.1. Retour sur la mémoire	531
Du stockage pour tous !	531
C'est à quelle adresse ?	534
31.2. Les indirections	535
31.3. Les ressources et les handles	537
31.4. Restitution garantie 100% satisfaction	538
Le travail manuel ne paie pas toujours... ..	538
Resource Acquisition Is Initialization (RAII)	540
L'exemple corrigé	541
31.5. <code>std::unique_ptr</code> – pour surveiller la mémoire	543
Mais quel rôle ce pointeur joue-t-il ?	543
Un seul propriétaire... ..	544
...et plusieurs observateurs	545
Liens entre propriétaire et observateurs	547
Transfert de responsabilité	547
Plusieurs propriétaires, plusieurs observateurs	549
31.6. Les entités et les handles	549
31.7. Un problème de destruction	553

31.8. Les pointeurs nus, on oublie !	555
std::function – un remplaçant pour manipuler des fonctions	555
std::optional – un remplaçant pour valeur optionnelle	556
31.9. De l'importance de prendre ses responsabilités	557
31.10. Récapitulons !	558
32. La sémantique de déplacement	561
32.1. T'as saisi la référence ?	561
Le besoin... ..	561
...la cause... ..	562
...et la solution	563
std::move, la mal-nommée	564
32.2. ♪ I like to move it, move it ♪	565
32.3. En interne	568
32.4. Comme les cinq doigts de la main	571
32.5. Récapitulons !	571
33. Oh, le bel héritage	573
33.1. NVI – Un contrat est un contrat	573
Des problèmes d'héritage... ..	573
Non Virtual Interface	573
Changements dans les contrats	576
33.2. <i>protected</i> – la portée intermédiaire	578
33.3. Mettons un point final à cet héritage	581
33.4. L'héritage multiple	583
33.5. L'héritage privé	586
33.6. L'héritage, solution miracle ?	589
33.7. Récapitulons !	590
34. Les classes templates	593
34.1. Les classes génériques	593
Définir une classe générique	593
Référencer une classe générique	594
Arguments types et non-types	595
Valeur par défaut	596
34.2. [Pratique] Un wrapper pour les pointeurs nus	596
34.3. Les traits	597
Exemple simple	598
Traits et programmation par contrat	598
En interne	601
34.4. Ces modèles sont bien trop génériques... ..	602

En voilà un bon concept !	603
Créer ses propres concepts	604
34.5. Un peu de politique	607
À la base, une classe très hospitalière	608
Points de variation	608
Lier l'implémentation et les points de variation	610
34.6. Récapitulons !	612
35. Ça, c'est du SOLID !	613
35.1. S – Single responsibility principle	613
Le principe	613
Un exemple problématique... ..	614
...et sa solution	615
35.2. O – Open-closed principle	615
Le principe	615
Un exemple problématique... ..	616
...et sa solution	617
35.3. L – Liskov substitution principle	619
Le principe	619
Un exemple problématique... ..	620
...et sa solution	620
35.4. I – Interface segregation principle	621
Le principe	621
Un exemple problématique... ..	621
...et sa solution	623
35.5. D – Dependency inversion principle	624
Le principe	624
Un exemple problématique... ..	625
...et sa solution	626
35.6. Pour quelques principes de plus	627
35.7. Récapitulons !	628
36. Le voyage ne fait que commencer	629
36.1. Davantage sur C++	629
36.2. Davantage sur le développement	630
[Pratique] Gérer les erreurs d'entrée • Solutions	631
Solutions des exercices	641
1. Chapitre <i>Tout ça est bien variable</i>	641
2. Chapitre <i>Le conditionnel conjugué en C++</i>	642

3. Chapitre <i>Des boucles qui se répètent, répètent, répètent...</i>	645
4. Chapitre <i>Au tableau !</i>	648
5. Chapitre <i>Découpons le code – les fonctions</i>	651
6. Chapitre <i>Déployons la toute puissance des conteneurs</i>	655
7. Chapitre <i>Des flux dans tous les sens</i>	658
8. Chapitre <i>Erreur, erreur, erreur...</i>	661
9. Chapitre <i>Des fonctions somme toute lambdas</i>	662
10. Chapitre <i>De nouvelles structures de données</i>	664
11. Chapitre <i>[Pratique] Un gestionnaire de discographie</i>	667
section <i>Représentation de la discographie</i>	667
section <i>Fonctions de base</i>	667
section <i>Stockage dans un fichier</i>	672
section <i>Affichage de la discographie</i>	673
section <i>Système de commandes</i>	676
section <i>Finalisation</i>	679
12. Chapitre <i>Découpons du code – les fichiers</i>	681
13. Chapitre <i>Mais où est la doc ?</i>	692
14. Chapitre <i>Premiers pas avec la POO</i>	696
15. Chapitre <i>Et qui va construire tout ça ?</i>	697
Index	701
À propos des auteurs	711

1

C'est décidé, je m'y mets !

Voilà, vous êtes décidé, vous voulez apprendre à programmer en C++. Mais quelques questions continuent à vous trotter dans la tête : Concrètement, le langage C++, c'est quoi ? Et puis pourquoi choisir le C++ ? Que vais-je savoir à l'issue de ce livre ? Y a-t-il des conditions pour pouvoir suivre ce livre ?

1.1. Le C++, qu'est-ce que c'est ?

Faisons un plongeon dans l'histoire et revenons dans les années 1970. À cette époque, Dennis Ritchie, programmeur aux laboratoires AT&T aux États-Unis, invente le langage C, conçu pour programmer le système d'exploitation UNIX. Ce langage devint très populaire à tel point qu'il est encore beaucoup utilisé aujourd'hui, notamment pour l'écriture de Linux. Un peu plus tard, au début des années 1980, Bjarne Stroustrup, lui aussi développeur aux laboratoires AT&T, décide de prendre ce langage C comme base et de lui ajouter des fonctionnalités issues d'un autre langage appelé Simula. Cette première bouture est appelée le C *with classes*. Finalement, en 1983, son créateur, estimant que le nom de son langage est trop réducteur au vu de tous les ajouts faits par rapport au C, décide de le renommer C++. Mais l'histoire ne s'arrête pas là. Au contraire, le C++ continue d'évoluer à tel point qu'on décide au début des années 1990 de le *normaliser*, c'est-à-dire d'en établir les règles officielles. Ce travail de longue haleine s'acheva en 1998 ; cette version est ainsi souvent nommée C++98. Ensuite, en 2003, des corrections ont été apportées et l'on obtint C++03.

Puis de nouveau un chantier titanesque est mis en place pour améliorer encore plus le C++, ce qui aboutit huit ans plus tard, en 2011, à la sortie de C++11, jugée par beaucoup de développeurs comme la renaissance du C++. À partir de ce moment est établi un rythme de trois ans entre chaque nouvelle mise à jour de C++, ce qui a donné C++14, C++17, C++20 et C++23. À l'heure où j'écris, C++26 est sur le point d'être finalisé.

Pour ceux qui lisent l'anglais, vous trouverez plus d'informations sur l'histoire du langage, ses choix de conception, les améliorations apportées par les différentes normes, etc. dans la publication de Bjarne Stroustrup *Thriving in a Crowded and Changing World: C++ 2006–2020* [<https://d-booker.io.my/cpp-hist>].

C++ HISTORIQUE ET C++ MODERNE

Beaucoup de programmeurs utilisent le terme C++ *historique* pour désigner les normes C++98 et C++03 et le terme C++ *moderne* pour parler de C++11 et au-delà. Cette distinction n'est pas fausse, mais la différence entre le C++ historique et le C++ moderne tient moins à une question de norme qu'à des bonnes pratiques, de meilleures façons de faire, etc.

Par exemple, on peut utiliser la dernière norme mais continuer à écrire son code comme on le faisait à l'époque du commencement de C++, dans un style proche du langage C. À l'inverse, on peut être bloqué pour une raison quelconque avec un ancien compilateur et donc être forcé d'utiliser C++98 ou C++03 et pourtant écrire du code moderne, en accord avec les bonnes pratiques issues du C++ moderne.

1.2. Pourquoi apprendre le C++ ?

- Sa *popularité* : le C++ est utilisé dans de nombreux projets importants (citons LibreOffice, 7-zip ou encore KDE). Il est au programme de beaucoup de formations informatiques. Il possède une communauté très vaste, beaucoup de documentation et d'aide, surtout sur l'internet anglophone.
- Sa *rapidité* : il offre un grand contrôle sur la rapidité des programmes. C'est cette caractéristique qui fait de lui un des langages de choix pour les programmes scientifiques, par exemple. Il s'associe notamment très bien avec des langages comme Python, pour écrire des blocs de code qui doivent s'exécuter très rapidement.
- Son *ancienneté* : c'est un langage ancien d'un point de vue informatique (30 ans, c'est énorme), ce qui donne une certaine garantie de maturité, de stabilité et de pérennité (il ne disparaîtra pas dans quelques années).
- Son *évolution* : C++11 est un véritable renouveau du C++ historique, qui le rend plus facile à utiliser et plus puissant dans les fonctionnalités qu'il offre aux développeurs. C++14, C++17 et C++20 améliorent encore la chose.
- Il est *multiparadigme* : il n'impose pas une façon unique de concevoir et découper ses programmes mais laisse le développeur libre de ses choix.

Bien entendu, tout n'est pas parfait et le C++ a aussi ses défauts.

- Son *héritage du C* : le C++ est un descendant du langage C, inventé dans les années 1970. Certains choix de conception, adaptés pour l'époque, sont plus problématiques aujourd'hui, et le C++ les traîne avec lui.

- Sa *complexité* : il ne faut pas se le cacher, atteindre une certaine maîtrise du C++ est très long et demandera des années d'expérience, notamment parce que certaines de ses fonctionnalités les plus puissantes requièrent de très bien connaître les bases.
- Sa *bibliothèque standard* : bien qu'elle permette de faire beaucoup de choses (d'ailleurs, nous ne ferons que l'aborder dans ce livre), elle n'offre pas de mécanisme natif pour manipuler des bases de données, faire des programmes en fenêtres, jouer avec le réseau, etc. Par rapport à d'autres langages comme Python, C++ peut paraître plus limité.

1.3. Développer, un métier à part entière

Eh oui ! Créer des logiciels, c'est un métier. Il y a des écoles et des études spécialisées pour ça. C'est que développer, ce n'est pas simplement écrire du code. Il y a aussi des phases de réflexion, de conception, d'écriture, de validation, de tests, de réécriture d'anciennes portions, etc. Et puis, même si ce n'est pas directement lié à la programmation, celle-ci requiert des connaissances en gestion de base de données, en usage du réseau, en management, en gestion de projet, etc. En bref, *être développeur c'est beaucoup de compétences différentes dans des domaines variés.*

La cruelle vérité qui se cache derrière tout ça, c'est que ce livre ne fera pas de vous des experts, ni des développeurs professionnels. Par contre, une fois fini, vous aurez des bases solides pour continuer votre apprentissage. La route du savoir est infinie.

1.4. Votre part du travail

Ce livre est écrit de façon à être le plus clair possible, sans vous noyer sous un flot de détails et d'explications, mais il arrivera parfois que vous ne compreniez pas un morceau de code ou une explication. C'est tout à fait normal, ça fait partie de l'apprentissage. Reprenez le passage à tête reposée, aidez-vous de schémas ou de dessins, demandez de l'aide sur les forums, et vous ne resterez jamais bloqué longtemps.

Par contre, vous devez être prêt à fournir des efforts de votre côté. Cela signifie ne pas vous ruer sur les forums à la première difficulté rencontrée, sans chercher à essayer de la résoudre par vous-même.

1.5. Les mathématiques, indispensables ?

Une des grandes appréhensions, qui revient régulièrement dans la bouche des débutants, est de savoir si les mathématiques sont un prérequis à l'apprentissage de la pro-

grammation. La réponse est non. Il est tout à fait possible d'apprendre à programmer tout en ayant un faible niveau en mathématiques. Ce livre ne requiert aucune connaissance en mathématiques, si ce n'est celle des opérations de base et, pour quelques exemples, du sinus et du cosinus.

Bien sûr, certains aspects de la programmation, comme la sécurité, la cryptographie ou les applications scientifiques demandent un bagage mathématique solide. Mais cela n'entre pas dans le cadre de ce livre.

Par contre, avoir l'esprit logique reste un plus indéniable.

1.6. L'anglais, indispensable ?

À vrai dire, pour suivre ce livre, pas besoin de savoir parler anglais. Même si nous nous reporterons régulièrement à de la documentation écrite dans la langue de Shakespeare, nous ne le ferons jamais sans l'accompagner de quelques explications et éclaircissements. Et quand bien même il resterait quelque chose sur lequel vous butiez, vous pourriez vous servir d'un traducteur automatique.

D'un point de vue plus général, si vous souhaitez continuer dans l'informatique et la programmation, il sera très difficile d'échapper à l'anglais. Beaucoup de cours, de documents, de forums sont en anglais ou en font un usage massif. L'anglais est tout simplement *la langue de l'informatique*. Sachez cependant que l'anglais informatique est simple à comprendre, car souvent écrit par des gens dont ce n'est pas la langue maternelle. Ainsi, inutile d'être bilingue, un bon dictionnaire ou un traducteur suffiront.

1.7. La documentation

En programmation, il y a un réflexe à adopter le plus rapidement possible : si on ne sait pas comment utiliser un outil, il faut aller consulter sa documentation, et ce avant de demander de l'aide sur un forum par exemple. Voici un lien vers une excellente documentation C++ : <https://en.cppreference.com/w/cpp>". Elle est en anglais, mais pas de souci, nous sommes là avec vous. Au fur et à mesure, nous vous donnerons des liens, nous vous expliquerons comment comprendre et exploiter les informations fournies pour que, par la suite, vous puissiez le faire par vous-même.

Il y a aussi un autre outil, très utile pour rechercher une information et que vous connaissez déjà : les moteurs de recherche (Google, Bing, DuckDuckGo, Qwant, etc.). Sachez aussi que les forums sont une mine d'informations. Vous pouvez par exemple utiliser ceux de Zeste de Savoir, en n'oubliant bien évidemment pas d'utiliser le tag [c++] lors de la création d'un sujet.

Enfin, sachez qu'il existe une référence ultime appelée *la norme*, produite par un organisme de validation international appelé l'*International Organization for Standardization* ou ISO, qui explique tous les détails et les règles du C++ mais qui est un document complexe et très largement hors de portée quand on débute. La dernière version de ce document est sortie en 2023 et explique les règles de fonctionnement du C++ dans sa version de 2023. Nous le mentionnons simplement pour que vous soyez au courant de son existence et que vous ne soyez pas surpris si, lors de vos recherches sur Internet, des réponses s'y rapportent ou citent la norme.

1.8. Récapitulons !

- Le C++ a une longue histoire, mais le C++ moderne a véritablement vu le jour en 2011, avec des améliorations et corrections apportées en 2014, 2017, 2020, 2023 et bientôt 2026.
- Le C++ possède des atouts qui en font un des langages les plus utilisés sur la planète.
- En contrepartie, le C++ est un langage assez complexe qui demande de nombreuses années avant d'être maîtrisé.
- La programmation est une activité parfois complexe, mais très enrichissante et accessible.
- Ce livre se veut le plus accessible possible, mais vous devez jouer le jeu et faire des efforts de votre côté.
- Les mathématiques et l'anglais ne sont pas requis pour suivre ce livre, toutefois un minimum de logique et un niveau acceptable en anglais seront extrêmement utiles par la suite.
- La documentation nous aide à mieux comprendre le langage, son fonctionnement, etc. Elle est en anglais, mais est illustrée par des exemples.