

Table des matières

À propos des auteurs	xx
Bienvenue !	1
1. Codes sources des exemples	1
2. Remerciements	1
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	8
2. Le minimum pour commencer	9
2.1. Des outils en ligne	9
2.2. Des outils plus poussés	11
Visual Studio 2022, l'outil tout-en-un pour Windows	12
Visual Studio Code, la solution portable	16
2.3. Un mot concernant Windows	21
2.4. Récapitulons !	22
3. Rencontre avec le C++	23
3.1. Compilons notre premier programme	23
3.2. Démystifions le code	24
Utiliser des fonctionnalités déjà codées	24
Le point d'entrée	24
Voici mes instructions...	24
3.3. Les commentaires	25
3.4. Variantes pré-C++23	26
3.5. Récapitulons !	28
4. Tout ça est bien variable	29
4.1. Les littéraux	29

Les chaînes de caractères	29
Les caractères	30
Les caractères spéciaux	31
Les nombres	34
4.2. Les variables	36
Comment créer des variables en C++ ?	36
Un peu de constance, voyons !	40
Manipulation de variables	41
4.3. Simple déduction ?	44
Avec <code>constexpr</code>	45
Avec <code>std::string</code>	45
Quel intérêt ?	46
4.4. Les entrées	46
4.5. Récapitulons !	48
5. Le conditionnel conjugué en C++	50
5.1. Les booléens	50
5.2. <i>if</i> – <i>si</i> , et seulement si...	52
5.3. <i>else</i> – sinon...	54
5.4. <i>else if</i> – la combinaison des deux précédents	56
5.5. Conditions imbriquées	57
5.6. [Pratique] Gérer les erreurs d'entrée • Partie I	58
5.7. La logique booléenne	59
AND – tester si deux conditions sont vraies	59
OR – tester si au moins une condition est vraie	60
NOT – tester la négation	61
5.8. Tester plusieurs expressions	62
5.9. Entraînez-vous !	65
Une pseudo-horloge	65
Score	65
XOR – le OU exclusif	66
5.10. Récapitulons !	66
6. Des boucles qui se répètent, répètent, répètent...	73
6.1. <i>while</i> – tant que...	73
6.2. Entraînez-vous !	75
Une laverie	75
PGCD	76
Somme de nombres de 1 à <i>n</i>	76
6.3. <i>do while</i> – répéter ... tant que	76

6.4. [Pratique] Gérer les erreurs d'entrée • Partie II	77
6.5. <i>for</i> – une boucle condensée	78
6.6. Boucles imbriquées	79
6.7. Convention de nommage	80
6.8. Contrôler plus finement l'exécution de la boucle	80
<i>break</i> – je m'arrête là	80
<i>continue</i> – saute ton tour !	82
6.9. Récapitulons !	82
7. Au tableau !	89
7.1. Un tableau c'est quoi ?	89
7.2. <i>std::vector</i> – mais quel dynamisme !	89
Déclarer un <i>std::vector</i>	90
Manipuler les éléments d'un <i>std::vector</i>	91
7.3. Entraînez-vous !	98
Calcul de moyenne	98
Minimum et maximum	99
Séparer les pairs des impairs	99
Compter les occurrences d'une valeur	99
7.4. <i>std::array</i> – le tableau statique	99
Déclarer un <i>std::array</i>	99
Accéder aux éléments	100
Connaître la taille	101
Vérifier si le tableau est vide	102
7.5. <i>std::string</i> – un type qui cache bien son jeu	102
Accéder à un caractère	102
Premier et dernier caractère	103
Vérifier qu'une chaîne est vide	103
Ajouter ou supprimer un caractère à la fin	104
Supprimer tous les caractères	104
Boucler sur une chaîne	105
7.6. Récapitulons !	105
8. Découpons le code – les fonctions	110
8.1. Les éléments de base d'une fonction	110
Une fonction, c'est quoi ?	110
Une fonction incontournable	112
Les composants d'une fonction	112
8.2. Entraînez-vous !	115
Afficher un rectangle	116

Distributeur d'argent	116
Parenthésage	116
8.3. Quelles sont vos références ?	117
La problématique	117
Les références	118
Paramètres de fonctions et références	119
Valeur de retour et références	122
Un mot sur la déduction de type	123
Tableaux et références	124
8.4. Nos fonctions sont surchargées !	125
8.5. [Pratique] Gérer les erreurs d'entrée • Partie III	127
8.6. Dessine-moi une fonction	128
8.7. Récursivité : L'art de se rappeler... soi-même !	129
PGCD récursif	131
La suite de Fibonacci	131
8.8. Récapitulons !	131
9. Déployons la toute puissance des conteneurs	138
9.1. La pièce maîtresse : les itérateurs	138
Déclaration d'un itérateur	139
Début et fin d'un conteneur	140
Accéder à l'élément pointé	140
Se déplacer dans une collection	141
const et les itérateurs	144
Itérer depuis la fin	146
Utilisation conjointe avec les conteneurs	147
9.2. Des algorithmes à gogo !	148
std::find – trouver un certain élément	148
std::count – compter les occurrences d'une valeur	149
std::sort – trier une collection	150
std::reverse – inverser l'ordre des éléments d'une collection	152
std::remove – suppression d'éléments	152
std::search – rechercher un sous-ensemble dans un en- semble	153
std::equal – vérifier l'égalité de deux ensembles	155
Et tant d'autres	156
9.3. Personnalisation à la demande	156
Le prédicat, pivot de la personnalisation des algorithmes	156
Trier un ensemble par ordre décroissant	156
Somme et produit	157

Et d'autres encore	158
9.4. Entraînez-vous !	158
Palindrome	158
Un <code>std::set</code> fait maison	159
Couper une chaîne	159
9.5. Récapitulons !	159
10. Des flux dans tous les sens	165
10.1. Avant toute chose	165
Prendrez-vous une extension ?	165
Vois sur ton chemin...	166
Pas d'interprétation, on est des brutes	167
10.2. <code>std::ofstream</code> – écrire dans un fichier	168
Ouvrir le fichier	168
Écrire	168
Ouvrir sans effacer	169
10.3. <code>std::ifstream</code> – lire dans un fichier	170
Ouvrir le fichier	170
Lire	171
Tout lire	173
10.4. Encore plus de flux !	173
Un flux c'est quoi ?	174
Des flux... même avec des chaînes !	174
Un <i>buffer</i> dites-vous ?	176
10.5. Aller plus loin avec les fichiers	178
10.6. Entraînez-vous !	180
Statistiques sur des fichiers	180
Lire un fichier CSV simple	180
Combiner des fichiers	181
10.7. Récapitulons !	181

On passe la deuxième ! 185

11. Erreur, erreur, erreur...	186
11.1. L'utilisateur est un idiot	186
11.2. À une condition... ou plusieurs	187
Contrats assurés par le compilateur	188
Vérifions nous aussi	188
11.3. Le développeur est un idiot	188
Préconditions et assertions	191

11.4. Les tests unitaires à la rescousse	192
Pourquoi tester ?	192
Un test unitaire, qu'est-ce que c'est ?	192
Écrire des tests unitaires	193
Les tests unitaires et les postconditions	195
11.5. [Pratique] Gérer les erreurs d'entrée • Partie IV	195
11.6. L'exception à la règle	196
La problématique	196
C'est quoi une exception ?	197
Lancement des exceptions dans 3, 2, 1...	198
Attends que je t'attrape !	199
Attrapez-les tous !	201
Un type pour les gouverner tous et dans le catch les lier	202
11.7. [Pratique] Gérer les erreurs d'entrée • Partie V	203
11.8. Récapitulons !	204
12. Des fonctions somme toute lambdas	208
12.1. Une lambda, c'est quoi ?	208
12.2. Pourquoi a-t-on besoin des lambdas ?	208
12.3. Syntaxe	209
Quelques exemples simples	209
12.4. Entraînez-vous !	212
Vérifier si un nombre est négatif	212
Tri par valeur absolue	212
<i>string_trim</i> – supprimer les espaces au début et en fin de chaîne	213
12.5. On va stocker ça où ?	213
12.6. Paramètres génériques	214
12.7. [Pratique] Gérer les erreurs d'entrée • Partie VI	215
12.8. [Pratique] Gérer les erreurs d'entrée • Partie VII	215
12.9. Capture en cours...	216
Capture par valeur	216
Capture par référence	217
12.10. Récapitulons !	219
13. Formez les rang(e)s !	223
13.1. Un exemple concret	223
13.2. Qu'est-ce qu'un <i>range</i> et une <i>view</i> ?	225
13.3. Réécrivons le passé	225
<i>std::ranges::find</i> – trouver un certain élément	225

<code>std::ranges::count</code> – compter les occurrences d’une valeur	226
<code>std::ranges::sort</code> – trier une collection	227
<code>std::ranges::reverse</code> – inverser l’ordre des éléments d’une collection	227
<code>std::ranges::equal</code> – vérifier l’égalité de deux ensembles	228
13.4. Pour quelques exemples de plus	228
13.5. Récapitulons !	231
14. Envoyez le générique !	232
14.1. Les fonctions, c’est <i>auto</i> -matique	232
14.2. Quel beau modèle !	233
La bibliothèque standard : une fourmilière de modèles	236
Un point de vocabulaire	236
14.3. Instanciation explicite	237
14.4. [Pratique] Gérer les erreurs d’entrée • Partie VIII	238
14.5. Ce type-là, c’est un cas !	238
14.6. Quand les lambdas s’en mêlent	240
14.7. Récapitulons !	241
15. De nouvelles structures de données	242
15.1. <code>struct</code> – un agrégat de données	242
Stockage d’informations personnelles	242
Analyse de la méthode	243
Introduction aux structures	243
Et maintenant, structurons !	245
15.2. <code>std::tuple</code> – une collection hétérogène	246
Déclaration	246
Accès aux éléments	246
Renvoyer plusieurs valeurs	248
Un outil pour stocker sans étiqueter	250
15.3. <code>std::unordered_map</code> – une table associative	251
Un dictionnaire de langue Zestienne	252
Toujours plus de clés	254
Cette clé est unique !	255
Cherchez un élément	257
Un exemple concret	258
15.4. Un peu d’ordre, voyons !	259
15.5. Une longue énumération	261
15.6. Récapitulons !	263
16. Reprendrez-vous un peu de sucre syntaxique ?	267

16.1. Ce type est trop long !	267
16.2. En pleine décomposition	268
16.3. La surcharge d'opérateurs	270
Le problème... ..	270
...et la solution	272
Mise en pratique	272
Et la bibliothèque standard ?	283
Sur le bon usage de la surcharge	284
16.4. Récapitulons !	284
17. [Pratique] Un gestionnaire de discographie	285
17.1. Cahier des charges	285
Ajout de morceaux	285
Affichage de la discographie	286
Enregistrement et chargement d'une discographie	288
Fermeture du programme	288
Gestion des erreurs	288
Dernières remarques	288
17.2. Guide de résolution	289
Analyse des besoins	289
Attaquons le code !	290
17.3. Code complet	294
17.4. Conclusion et pistes d'améliorations	305
17.5. Récapitulons !	306
18. Découpons du code – les fichiers	322
18.1. C++, un langage très modulable	322
Créer son propre module	323
Des modules bien rangés	328
Diviser pour mieux régner	330
18.2. Les <code>#include</code> , à l'époque où tout était simple... ou pas !	332
Le fichier d'en-tête	332
Le fichier source	333
Créons un lien	334
La sécurité, c'est important	335
18.3. À projet moderne, solution moderne	337
18.4. [Pratique] Découpons notre programme !	338
18.5. Les avantages du découpage en fichiers	341
18.6. Le cas des templates	342
18.7. Finis les conflits, vive les namespaces	345

18.8. Récapitulons !	347
Interlude : être développeur	361
19. Un coup d'œil dans le rétro	362
19.1. Au-delà du code	362
19.2. Tout est une question de paradigme	363
Le paradigme impératif	363
Le paradigme fonctionnel	364
Le paradigme générique	364
La programmation par contrat	365
19.3. L'art de la conception	365
19.4. De la persévérance avant tout	366
19.5. Récapitulons !	367
20. Mais où est la doc ?	368
20.1. Lire une page de doc	369
vector – retour sur le plus célèbre des conteneurs	371
Les algorithmes	373
Les chaînes de caractères au grand complet	374
20.2. Entraînez-vous	375
Remplacer une chaîne de caractères par une autre	375
Norme d'un vecteur	376
Nombres complexes	376
Transformations	377
20.3. Documenter son code avec Doxygen	377
Installation d'un outil dédié	377
Écrire la documentation	381
20.4. Quelques bonnes pratiques	386
Ce qu'il faut documenter	387
Définition ou implémentation ?	387
Rien ne vaut un exemple	388
Commentaire vs documentation	389
20.5. Récapitulons !	391
21. Compilation en cours...	398
21.1. Le préprocesseur	398
Inclure des fichiers	398
Conditions	399
Debug ou release ?	400

21.2. Les modules, pour une compilation améliorée	402
21.3. La compilation	403
Les templates	403
<i>constexpr</i>	404
La compilation à proprement parler	407
Une étape intermédiaire cachée	408
Influer sur la compilation	408
21.4. Le <i>linker</i>	414
Une question de symboles	415
21.5. Schéma récapitulatif	417
21.6. Récapitulons !	418
22. Chasse aux bugs !	419
22.1. Le principe	419
22.2. Un code d'exemple	420
22.3. Avec Visual Studio	420
Interface	421
Pas-à-pas	423
Point d'arrêt conditionnel	424
22.4. En ligne de commande avec gdb	426
Poser un point d'arrêt	426
Supprimer des points d'arrêt	427
Désactiver des points d'arrêt	428
Afficher l'état d'une variable	428
Pas-à-pas	428
Conditions	429
S'y retrouver dans le code	430
22.5. Avec Visual Studio Code	430
Interface	430
Pas-à-pas	432
Point d'arrêt conditionnel	433
22.6. Récapitulons !	434
23. Une foule de bibliothèques	435
23.1. Quelles bibliothèques choisir ?	435
23.2. Généralités	436
Statique ou dynamique ?	436
<i>Debug</i> ou <i>release</i> ?	437
32 ou 64 bits ?	437
Bibliothèques <i>header-only</i>	437

23.3. Installer Boost	438
GNU/Linux	438
Visual Studio	441
23.4. Installer SFML	444
GNU/Linux	445
Visual Studio	445
23.5. Vos bibliothèques en un clic (ou presque)	449
23.6. Récapitulons !	449
24. Construisons mieux avec CMake	450
24.1. CMake, l'ingénieur en chef de vos compilations	450
Visual Studio	451
Visual Studio Code	451
24.2. Notre premier projet avec CMake	452
24.3. Compilation multi-fichiers	459
24.4. Inclure des bibliothèques externes	461
Bibliothèques installées localement	461
Que ferait-on sans Internet ?	464
24.5. Aller plus loin avec CMake	467
24.6. Récapitulons !	468
25. Pour une poignée d'outils	469
25.1. git – sauvegarder et versionner son code	469
Installer git	470
Initialiser git	470
Créer un dépôt	470
Connaître l'état de ses fichiers	471
Ajouter un fichier	471
Valider les modifications	472
Voir l'historique	472
Bloquer certains fichiers	473
Aller plus loin	475
25.2. GitLab – partager son code	475
Création d'un projet	476
Synchroniser les modifications	477
Aller plus loin	478
25.3. Encore quelques outils	478
GitLab CI – de l'intégration continue	478
De vrais tests unitaires	478
CppCheck – vérification du code	479

StackOverflow – la réponse à quasi toutes les questions	480
25.4. Récapitulons !	480
La programmation orientée objet	481
26. Premiers pas avec la POO	482
26.1. Le principe : des objets bien serviables	482
Penser en termes de données... ..	482
...ou de services ?	483
Exemple concret	483
26.2. Un peu de vocabulaire	484
L'objet	484
Le type	484
L'interface d'une classe	484
26.3. En C++, ça donne quoi ?	485
Penser services	485
Penser tests	485
Fonctions membres	486
Fonctions libres	487
Les attributs	487
26.4. Désolé, cet objet n'est pas modifiable	488
26.5. On ne fait pas d'exception	491
26.6. Découpage en fichiers	493
26.7. Entraînez-vous !	495
Cercle	495
Informations personnelles	496
26.8. Récapitulons !	496
27. Et qui va construire tout ça ?	500
27.1. Encapsulation et invariants	500
Cette classe, c'est open bar	500
Tout est question d'invariant	501
Encapsulons tout ça	502
Modifier sa visibilité	502
27.2. On cherche un constructeur	503
Le principe	503
La syntaxe	505
27.3. Constructeur par défaut	507
27.4. Soyons explicites	510
27.5. En toute amitié	513

Attributs publics	516
Accesseurs	516
La troisième, c'est la bonne	517
27.6. Entraînez-vous !	518
Un autre constructeur pour <i>Fraction</i>	518
Une pile	519
27.7. Récapitulons !	520
28. Une classe de grande valeur	523
28.1. Une histoire de sémantique	523
28.2. La sémantique de valeur, c'est quoi ?	525
28.3. Égalité	525
28.4. Le retour des opérateurs	526
Les opérateurs d'affectation intégrés	526
Du deux en un	528
Les opérateurs de manipulation des flux	529
Quelques bonnes pratiques	529
Les autres opérateurs	530
28.5. Copier des objets	530
Constructeur de copie	530
Opérateur d'affectation par copie	531
28.6. Récapitulons !	533
29. [Pratique] Entrons dans la matrice	534
29.1. Qu'est-ce qu'une matrice ?	534
29.2. Cahier des charges	536
Les services	536
Les détails d'implémentation	536
Les tests	537
29.3. Réalisation	541
Les services	541
Les détails d'implémentation	543
29.4. Code complet	550
29.5. Entraînez-vous !	557
Entiers infinis	557
Des polynômes	558
29.6. Récapitulons !	559
30. Classes à sémantique d'entité	560
30.1. Des objets uniques !	560
Conséquences sur l'interface	560

Un exemple	561
30.2. Des sous-types... ..	563
Exemple	563
Héritage et constructeurs	564
Une question de visibilité	567
30.3. ...substituables	569
Une histoire de manchots	571
Le principe de substitution de Liskov	572
Substitution dans la bibliothèque standard	572
Substitution et programmation par contrat	573
Héritage public et LSP	575
30.4. Un monde virtuel	577
Un problème... ..	577
...une solution	580
En interne	583
Faisons le point	586
30.5. Interfaces et classes abstraites	586
Des fonctions virtuelles vraiment pures	586
Classe abstraite et implémentation	588
30.6. La copie, une vraie source de problème	589
30.7. Récapitulons !	591
31. Ressources, indirections et handles	592
31.1. Retour sur la mémoire	592
Du stockage pour tous !	592
C'est à quelle adresse ?	595
31.2. Les indirections	596
31.3. Les ressources et les handles	598
31.4. Restitution garantie 100% satisfaction	599
Le travail manuel ne paie pas toujours... ..	599
Resource Acquisition Is Initialization (RAII)	601
L'exemple corrigé	602
31.5. <code>std::unique_ptr</code> – pour surveiller la mémoire	604
Mais quel rôle ce pointeur joue-t-il ?	604
Un seul propriétaire... ..	605
...et plusieurs observateurs	606
Liens entre propriétaire et observateurs	608
Transfert de responsabilité	608
Plusieurs propriétaires, plusieurs observateurs	610
31.6. Les entités et les handles	610

31.7. Un problème de destruction	614
31.8. Les pointeurs nus, on oublie !	616
std::function – un remplaçant pour manipuler des fonctions	616
std::optional – un remplaçant pour valeur optionnelle	617
31.9. De l'importance de prendre ses responsabilités	618
31.10. Récapitulons !	619
32. La sémantique de déplacement	621
32.1. T'as saisi la référence ?	621
Le besoin... ..	621
...la cause... ..	622
...et la solution	623
std::move, la mal-nommée	624
32.2. ♪ I like to move it, move it ♪	625
32.3. En interne	628
32.4. Comme les cinq doigts de la main	631
32.5. Récapitulons !	631
33. Oh, le bel héritage	633
33.1. NVI – Un contrat est un contrat	633
Des problèmes d'héritage... ..	633
Non Virtual Interface	633
Changements dans les contrats	636
33.2. <i>protected</i> – la portée intermédiaire	638
33.3. Mettons un point final à cet héritage	641
33.4. L'héritage multiple	643
33.5. L'héritage privé	646
33.6. L'héritage, solution miracle ?	649
33.7. Récapitulons !	650
34. Les classes templates	652
34.1. Les classes génériques	652
Définir une classe générique	652
Référencer une classe générique	653
Arguments types et non-types	654
Valeur par défaut	655
34.2. [Pratique] Un wrapper pour les pointeurs nus	655
34.3. Les traits	656
Exemple simple	657
Traits et programmation par contrat	657
En interne	660

34.4. Ces modèles sont bien trop génériques...	661
En voilà un bon concept !	662
Créer ses propres concepts	663
34.5. Un peu de politique	666
À la base, une classe très hospitalière	667
Points de variation	667
Lier l'implémentation et les points de variation	669
34.6. Récapitulons !	671
35. Ça, c'est du SOLID !	672
35.1. S – Single responsibility principle	672
Le principe	672
Un exemple problématique...	673
...et sa solution	674
35.2. O – Open-closed principle	674
Le principe	674
Un exemple problématique...	675
...et sa solution	676
35.3. L – Liskov substitution principle	678
Le principe	678
Un exemple problématique...	679
...et sa solution	679
35.4. I – Interface segregation principle	680
Le principe	680
Un exemple problématique...	680
...et sa solution	682
35.5. D – Dependency inversion principle	683
Le principe	683
Un exemple problématique...	684
...et sa solution	685
35.6. Pour quelques principes de plus	686
35.7. Récapitulons !	687
36. Le voyage ne fait que commencer	688
36.1. Davantage sur C++	688
36.2. Davantage sur le développement	689
[Pratique] Gérer les erreurs d'entrée • Solutions	690
Index	699