

# Table des matières

---

À propos des auteurs .....	xx
<b>Bienvenue ! .....</b>	<b>1</b>
1. Codes sources des exemples .....	1
2. Remerciements .....	1
<b>1. C'est décidé, je m'y mets ! .....</b>	<b>3</b>
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 ? .....	6
1.6. L'anglais, indispensable ? .....	6
1.7. La documentation .....	6
1.8. Récapitulons ! .....	7
<b>Le début du voyage .....</b>	<b>8</b>
<b>2. Le minimum pour commencer .....</b>	<b>9</b>
2.1. Des outils en ligne .....	9
2.2. Des outils plus poussés .....	11
2.3. Un mot concernant Windows .....	15
2.4. Un mot concernant GNU/Linux .....	16
Éditeur de texte .....	16
Compilateur .....	16
2.5. Récapitulons ! .....	17
<b>3. Rencontre avec le C++ .....</b>	<b>18</b>
3.1. Compilons notre premier programme .....	18
3.2. Démystifions le code .....	19
Utiliser des fonctionnalités déjà codées .....	19
Le point d'entrée .....	19
Voici mes instructions... .....	20
3.3. Les commentaires .....	21
3.4. Récapitulons ! .....	22
<b>4. Tout ça est bien variable .....</b>	<b>24</b>
4.1. Les littéraux .....	24

Les caractères .....	24
Les caractères spéciaux .....	26
Les nombres .....	27
4.2. Les variables .....	29
Comment créer des variables en C++ ? .....	30
Un peu de constance, voyons ! .....	34
Manipulation de variables .....	35
4.3. Simple déduction ? .....	37
Avec <i>const</i> .....	38
Avec <i>std::string</i> .....	38
Quel intérêt ? .....	39
4.4. Les entrées .....	39
4.5. Récapitulons ! .....	41
<b>5. Le conditionnel conjugué en C++ .....</b>	<b>44</b>
5.1. Les booléens .....	44
5.2. <i>if</i> – si, et seulement si... .....	46
5.3. <i>else</i> – sinon... .....	48
5.4. <i>else if</i> – la combinaison des deux précédents .....	50
5.5. Conditions imbriquées .....	51
5.6. [Pratique] Gérer les erreurs d'entrée • Partie I .....	52
5.7. La logique booléenne .....	53
AND – tester si deux conditions sont vraies .....	53
OR – tester si au moins une condition est vraie .....	54
NOT – tester la négation .....	55
5.8. Tester plusieurs expressions .....	56
5.9. Entraînez-vous ! .....	59
Une pseudo-horloge .....	59
Score .....	59
XOR – le OU exclusif .....	60
5.10. Récapitulons ! .....	60
<b>6. Des boucles qui se répètent, répètent, répètent... .....</b>	<b>67</b>
6.1. <i>while</i> – tant que... .....	67
6.2. Entraînez-vous ! .....	69
Une laverie .....	69
PGCD .....	70
Somme de nombres de 1 à <i>n</i> .....	70
6.3. <i>do while</i> – répéter ... tant que .....	70
6.4. [Pratique] Gérer les erreurs d'entrée • Partie II .....	71

6.5. <i>for</i> – une boucle condensée .....	72
6.6. Boucles imbriquées .....	73
6.7. Convention de nommage .....	74
6.8. Contrôler plus finement l'exécution de la boucle .....	74
<i>break</i> – je m'arrête là .....	74
<i>continue</i> – saute ton tour ! .....	76
6.9. Récapitulons ! .....	76
<b>7. Au tableau ! .....</b>	<b>83</b>
7.1. Un tableau c'est quoi ? .....	83
7.2. <i>std::vector</i> – mais quel dynamisme ! .....	83
Déclarer un <i>std::vector</i> .....	84
Manipuler les éléments d'un <i>std::vector</i> .....	85
7.3. Entraînez-vous ! .....	93
Calcul de moyenne .....	93
Minimum et maximum .....	93
Séparer les pairs des impairs .....	93
Compter les occurrences d'une valeur .....	93
7.4. <i>std::array</i> – le tableau statique .....	94
Déclarer un <i>std::array</i> .....	94
Remplir un tableau .....	95
Accéder aux éléments .....	95
Connaître la taille .....	96
Vérifier si le tableau est vide .....	97
7.5. <i>std::string</i> – un type qui cache bien son jeu .....	97
Connaître la taille d'une chaîne .....	97
Accéder à un caractère .....	98
Premier et dernier caractère .....	98
Vérifier qu'une chaîne est vide .....	99
Ajouter ou supprimer un caractère à la fin .....	99
Supprimer tous les caractères .....	100
Boucler sur une chaîne .....	100
7.6. Récapitulons ! .....	101
<b>8. Déployons la toute puissance des conteneurs.....</b>	<b>106</b>
8.1. La pièce maîtresse : les itérateurs .....	106
Déclaration d'un itérateur .....	106
Début et fin d'un conteneur .....	107
Accéder à l'élément pointé .....	108
Se déplacer dans une collection .....	108

const et les itérateurs .....	111
Itérer depuis la fin .....	112
Utilisation conjointe avec les conteneurs .....	113
8.2. Des algorithmes à gogo ! .....	114
std::count – compter les occurrences d’une valeur .....	115
std::find – trouver un certain élément .....	116
std::sort – trier une collection .....	117
std::reverse – inverser l’ordre des éléments d’une collection ....	118
std::remove – suppression d’éléments .....	119
std::search – rechercher un sous-ensemble dans un en- semble .....	119
std::equal – vérifier l’égalité de deux ensembles .....	120
Et tant d’autres .....	121
8.3. Personnalisation à la demande .....	121
Le prédicat, pivot de la personnalisation des algorithmes .....	121
Trier une liste dans l’ordre décroissant .....	122
Somme et produit .....	123
Prédicats pour caractères .....	124
8.4. Entraînez-vous ! .....	125
Palindrome .....	125
string_trim – suppression des espaces .....	125
Couper une chaîne .....	126
8.5. Récapitulons ! .....	126
<b>9. Des flux dans tous les sens .....</b>	<b>131</b>
9.1. Avant toute chose .....	131
Prendrez-vous une extension ? .....	131
Vois sur ton chemin... .....	132
Pas d’interprétation, on est des brutes .....	133
9.2. std::ofstream – écrire dans un fichier .....	134
Ouvrir le fichier .....	134
Écrire .....	134
Ouvrir sans effacer .....	137
9.3. std::ifstream – lire dans un fichier .....	138
Ouvrir le fichier .....	138
Lire .....	138
Tout lire .....	141
9.4. Entraînez-vous ! .....	141
Statistiques sur des fichiers .....	141
9.5. Encore plus de flux ! .....	142

Un flux c'est quoi ? .....	142
Un <i>buffer</i> dites-vous ? .....	143
Les modificateurs de flux .....	144
Même les chaînes y passent ! .....	145
9.6. Récapitulons ! .....	146

## On passe la deuxième ! ..... 149

### 10. Découpons le code — les fonctions..... 150

10.1. Les éléments de base d'une fonction .....	150
Une fonction, c'est quoi ? .....	150
Une fonction incontournable .....	152
Les composants d'une fonction .....	153
10.2. Entraînez-vous ! .....	156
Afficher un rectangle .....	156
Distributeur d'argent .....	156
Parenthésage .....	157
10.3. Quelles sont vos références ? .....	157
La problématique .....	157
Les références .....	159
Paramètres de fonctions .....	160
Valeur de retour .....	163
Un mot sur la déduction de type .....	164
10.4. Nos fonctions sont surchargées ! .....	166
10.5. [Pratique] Gérer les erreurs d'entrée • Partie III .....	167
10.6. Dessine-moi une fonction .....	168
Le problème .....	168
La solution .....	169
Utilisons les prototypes .....	170
10.7. Récapitulons ! .....	171

### 11. Erreur, erreur, erreur... ..... 176

11.1. L'utilisateur est un idiot .....	176
11.2. À une condition... ou plusieurs .....	177
Contrats assurés par le compilateur .....	178
Vérifions nous aussi .....	178
11.3. Le développeur est un idiot .....	178
Préconditions et assertions .....	181
11.4. Les tests unitaires à la rescousse .....	182
Pourquoi tester ? .....	182

Un test unitaire, qu'est-ce que c'est ? .....	182
Écrire des tests unitaires .....	183
Les tests unitaires et les postconditions .....	184
11.5. [Pratique] Gérer les erreurs d'entrée • Partie IV .....	184
11.6. L'exception à la règle .....	186
La problématique .....	186
C'est quoi une exception ? .....	187
Lancement des exceptions dans 3, 2, 1... .....	188
Attends que je t'attrape ! .....	189
Attrapez-les tous ! .....	191
Un type pour les gouverner tous et dans le catch les lier .....	192
11.7. [Pratique] Gérer les erreurs d'entrée • Partie V .....	194
11.8. Récapitulons ! .....	194
<b>12. Des fonctions somme toute lambdas.....</b>	<b>199</b>
12.1. Une lambda, c'est quoi ? .....	199
12.2. Pourquoi a-t-on besoin des lambdas ? .....	199
12.3. Syntaxe .....	200
Quelques exemples simples .....	200
12.4. Entraînez-vous ! .....	203
Vérifier si un nombre est négatif .....	203
Tri par valeur absolue .....	203
12.5. On va stocker ça où ? .....	203
12.6. Paramètres génériques .....	204
12.7. [Pratique] Gérer les erreurs d'entrée • Partie VI .....	206
12.8. [Pratique] Gérer les erreurs d'entrée • Partie VII .....	206
12.9. Capture en cours... .....	207
Capture par valeur .....	207
Capture par référence .....	209
12.10. Récapitulons ! .....	210
<b>13. Envoyez le générique ! .....</b>	<b>213</b>
13.1. Quel beau modèle ! .....	213
La bibliothèque standard : une fourmière de modèles .....	218
Un point de vocabulaire .....	218
13.2. [Pratique] Gérer les erreurs d'entrée • Partie VIII .....	219
13.3. Instanciation explicite .....	220
13.4. Ce type-là, c'est un cas ! .....	224
13.5. C++20 et les lambdas templates .....	228
13.6. Récapitulons ! .....	229

<b>14. De nouvelles structures de données.....</b>	<b>230</b>
14.1. <i>struct</i> – un agrégat de données .....	230
Stockage d'informations personnelles .....	230
Analyse de la méthode .....	231
Introduction aux structures .....	231
Et maintenant, structurons ! .....	233
14.2. <i>std::tuple</i> – une collection hétérogène .....	234
Déclaration .....	234
Accès aux éléments .....	234
<i>std::tuple</i> et fonctions .....	236
Équations horaires .....	237
14.3. <i>std::unordered_map</i> – une table associative .....	239
Un dictionnaire de langue Zestienne .....	240
Toujours plus de clés .....	242
Cette clé est unique ! .....	243
Cherchez un élément .....	245
Un exemple concret .....	247
14.4. <i>std::unordered_set</i> – représenter un ensemble .....	248
Inscription sur Zeste de Savoir .....	248
14.5. Un peu d'ordre, voyons ! .....	249
14.6. Une longue énumération .....	251
14.7. Récapitulons ! .....	253
<b>15. Reprenez-vous un peu de sucre syntaxique ? .....</b>	<b>257</b>
15.1. Ce type est trop long ! .....	257
15.2. Décomposons tout ça .....	258
En pleine décomposition .....	258
Et avant, on faisait comment ? .....	260
15.3. La surcharge d'opérateurs .....	262
Le problème... .....	262
...et la solution .....	264
Mise en pratique .....	264
Et la bibliothèque standard ? .....	271
Entraînez-vous ! .....	272
Sur le bon usage de la surcharge .....	273
15.4. Récapitulons ! .....	273
<b>16. [Pratique] Un gestionnaire de discographie .....</b>	<b>278</b>
16.1. Cahier des charges .....	278
Ajout de morceaux .....	278

Affichage de la discographie .....	279
Enregistrement et chargement d'une discographie .....	281
Fermeture du programme .....	281
Gestion des erreurs .....	281
Dernières remarques .....	281
16.2. Guide de résolution .....	282
Analyse des besoins .....	282
Attaquons le code ! .....	283
16.3. Code complet .....	287
16.4. Conclusion et pistes d'améliorations .....	297
16.5. Récapitulons ! .....	298
<b>17. Découpons du code – les fichiers .....</b>	<b>313</b>
17.1. Le principe .....	313
Le fichier d'en-tête .....	313
Le fichier source .....	316
Créons un lien .....	317
17.2. La sécurité, c'est important .....	318
17.3. [Pratique] Découpons notre programme ! .....	319
17.4. Les avantages du découpage en fichiers .....	321
17.5. Le cas des templates .....	323
17.6. Récapitulons ! .....	326
<b>Interlude : être développeur .....</b>	<b>350</b>
<b>18. Un coup d'œil dans le rétro .....</b>	<b>351</b>
18.1. Au-delà du code .....	351
18.2. Tout est une question de paradigme .....	352
Le paradigme impératif .....	352
Le paradigme fonctionnel .....	353
Le paradigme générique .....	354
La programmation par contrat .....	355
18.3. L'art de la conception .....	355
18.4. De la persévérance avant tout .....	356
18.5. Récapitulons ! .....	357
<b>19. Mais où est la doc ? .....</b>	<b>358</b>
19.1. Lire une page de doc .....	358
vector – retour sur le plus célèbre des conteneurs .....	359
Les algorithmes .....	362

Les chaînes de caractères au grand complet .....	363
19.2. Entraînez-vous .....	364
Remplacer une chaîne de caractères par une autre .....	364
Norme d'un vecteur .....	364
Nombres complexes .....	365
Transformations .....	365
19.3. Documenter son code avec Doxygen .....	366
Installation d'un outil dédié .....	366
Écrire la documentation .....	369
19.4. Quelques bonnes pratiques .....	375
Ce qu'il faut documenter .....	375
Les fichiers d'en-tête ou sources ? .....	376
Rien ne vaut un exemple .....	376
Commentaire vs documentation .....	377
19.5. Récapitulons ! .....	379
<b>20. Compilation en cours...</b> .....	<b>384</b>
20.1. Le préprocesseur .....	384
Inclure des fichiers .....	384
Conditions .....	385
Debug ou release ? .....	385
20.2. La compilation .....	387
Les templates .....	387
constexpr .....	388
La compilation à proprement parler .....	391
Une étape intermédiaire cachée .....	392
Influer sur la compilation .....	392
20.3. Le linker .....	397
Une question de symboles .....	398
20.4. Schéma récapitulatif .....	400
20.5. Récapitulons ! .....	401
<b>21. Chasse aux bugs ! .....</b>	<b>402</b>
21.1. Le principe .....	402
21.2. Un code d'exemple .....	403
21.3. Avec Visual Studio .....	403
Interface .....	404
Pas-à-pas .....	406
Point d'arrêt conditionnel .....	407
21.4. Avec Qt Creator .....	408

Interface .....	408
Pas-à-pas .....	410
Point d'arrêt conditionnel .....	411
21.5. En ligne de commande avec gdb .....	412
Poser un point d'arrêt .....	413
Supprimer des points d'arrêt .....	413
Désactiver des points d'arrêt .....	414
Afficher l'état d'une variable .....	414
Pas-à-pas .....	414
Conditions .....	415
21.6. Récapitulons ! .....	416
<b>22. Une foule de bibliothèques .....</b>	<b>417</b>
22.1. Quelles bibliothèques choisir ? .....	417
22.2. Généralités .....	418
Statique ou dynamique ? .....	418
Debug ou release ? .....	418
32 ou 64 bits ? .....	419
22.3. Installer Boost .....	419
GNU/Linux .....	420
Visual Studio 2019 .....	421
Qt Creator et MinGW sous Windows .....	423
22.4. Installer SFML .....	425
GNU/Linux .....	426
Windows .....	427
22.5. Récapitulons ! .....	432
<b>23. Améliorer ses projets .....</b>	<b>433</b>
23.1. git – sauvegarder et versionner son code .....	433
Installer git .....	434
Initialiser git .....	434
Créer un dépôt .....	434
Connaître l'état de ses fichiers .....	435
Ajouter un fichier .....	436
Valider les modifications .....	436
Voir l'historique .....	437
Bloquer certains fichiers .....	437
Aller plus loin .....	439
23.2. GitLab – partager son code .....	439
Création d'un projet .....	440

Pousser nos modifications .....	441
Aller plus loin .....	442
23.3. CMake – automatiser la compilation de ses programmes .....	442
Un exemple simple .....	442
Lier des bibliothèques externes .....	446
Définir des variables dynamiquement .....	453
Aller plus loin .....	453
23.4. Encore quelques outils .....	454
GitLab CI – de l’intégration continue .....	454
CppCheck – vérification du code .....	454
StackOverflow – la réponse à quasi toutes les questions .....	454
23.5. Récapitulons ! .....	455
<b>La programmation orientée objet .....</b>	<b>456</b>
<b>24. Premiers pas avec la POO .....</b>	<b>457</b>
24.1. Le principe : des objets bien serviables .....	457
Penser en termes de données... .....	457
...ou de services ? .....	458
Exemple concret .....	458
24.2. Un peu de vocabulaire .....	459
L’objet .....	459
Le type .....	459
L’interface d’une classe .....	459
24.3. En C++, ça donne quoi ? .....	460
Penser services .....	460
Penser tests .....	460
Fonctions membres .....	461
Fonctions libres .....	462
Les attributs .....	463
24.4. Désolé, cet objet n'est pas modifiable .....	464
24.5. On ne fait pas d'exception .....	467
24.6. Découpage en fichiers .....	468
24.7. Entraînez-vous ! .....	470
Cercle .....	470
Informations personnelles .....	471
24.8. Récapitulons ! .....	471
<b>25. Et qui va construire tout ça ? .....</b>	<b>475</b>
25.1. Encapsulation et invariants .....	475

Cette classe, c'est open bar .....	475
Tout est question d'invariant .....	476
Encapsulons tout ça .....	477
Modifier sa visibilité .....	477
25.2. On cherche un constructeur .....	479
Le principe .....	479
La syntaxe .....	481
25.3. Constructeur par défaut .....	482
25.4. Soyons explicites .....	485
25.5. En toute amitié .....	489
Attributs publics .....	491
Accesseurs .....	491
La troisième, c'est la bonne .....	493
25.6. Entraînez-vous ! .....	494
Un autre constructeur pour <i>Fraction</i> .....	494
Une pile .....	494
25.7. Récapitulons ! .....	496
<b>26. Une classe de grande valeur .....</b>	<b>499</b>
26.1. Une histoire de sémantique .....	499
26.2. La sémantique de valeur, c'est quoi ? .....	501
26.3. Égalité .....	501
Libre ou membre ? .....	502
Soyons intelligents, réutilisons .....	506
26.4. Le retour des opérateurs .....	507
Les opérateurs d'affectation intégrés .....	507
Du deux en un .....	508
Les opérateurs de manipulation des flux .....	509
Quelques bonnes pratiques .....	513
Les autres opérateurs .....	513
26.5. Copier des objets .....	514
Constructeur de copie .....	514
Opérateur d'affectation par copie .....	515
26.6. Récapitulons ! .....	515
<b>27. [Pratique] Entrons dans la matrice .....</b>	<b>517</b>
27.1. Qu'est-ce qu'une matrice ? .....	517
27.2. Cahier des charges .....	519
Les services .....	519
Les détails d'implémentation .....	519

Les tests .....	520
27.3. Réalisation .....	524
Les services .....	524
Les détails d'implémentation .....	526
27.4. Code complet .....	531
27.5. Entraînez-vous ! .....	538
Entiers infinis .....	538
Des polynômes .....	539
27.6. Récapitulons ! .....	540
<b>28. Classes à sémantique d'entités .....</b>	<b>541</b>
28.1. Des objets uniques ! .....	541
Conséquences sur l'interface .....	541
Un exemple .....	542
28.2. Des sous-types... .....	544
Exemple .....	544
Héritage et constructeurs .....	545
Une question de visibilité .....	547
28.3. ...substituables .....	549
Une histoire de manchots .....	551
Le principe de substitution de Liskov .....	552
Substitution dans la bibliothèque standard .....	553
Substitution et programmation par contrat .....	553
Héritage public et LSP .....	555
28.4. Un monde virtuel .....	557
Un problème... .....	557
...une solution .....	560
En interne .....	564
Faisons le point .....	566
28.5. Interfaces et classes abstraites .....	567
Des fonctions virtuelles vraiment pures .....	567
Classe abstraite et implémentation .....	568
28.6. La copie, une vraie source de problème .....	569
28.7. Récapitulons ! .....	571
<b>29. Ressources, indirections et handles.....</b>	<b>573</b>
29.1. Retour sur la mémoire .....	573
Du stockage pour tous ! .....	573
C'est à quelle adresse ? .....	576
29.2. Les indirections .....	576

29.3. Les ressources et les handles .....	578
29.4. Restitution garantie 100% satisfaction .....	579
Le travail manuel ne paie pas toujours... .....	579
Resource Acquisition Is Initialization .....	581
L'exemple corrigé .....	583
29.5. <code>std::unique_ptr</code> – pour surveiller la mémoire .....	585
Mais quel rôle ce pointeur joue-t-il ? .....	585
Un seul propriétaire... .....	585
...et plusieurs observateurs .....	587
Liens entre propriétaire et observateurs .....	588
Transfert de responsabilité .....	589
Plusieurs propriétaires, plusieurs observateurs .....	590
29.6. Les entités et les handles .....	591
29.7. Un problème de destruction .....	594
29.8. Les pointeurs nus, on oublie ! .....	596
<code>std::function</code> – un remplaçant pour manipuler des fonctions ....	596
<code>std::optional</code> – un remplaçant pour valeur optionnelle .....	597
29.9. De l'importance de prendre ses responsabilités .....	598
29.10. Récapitulons ! .....	599
<b>30. La sémantique de déplacement .....</b>	<b>601</b>
30.1. T'as saisi la référence ? .....	601
Le besoin... .....	601
...la cause... .....	603
...et la solution .....	604
<code>std::move</code> , la mal-nommée .....	605
30.2. ♪ I like to move it, move it ♪ .....	606
30.3. En interne .....	609
30.4. Comme les cinq doigts de la main .....	612
30.5. Récapitulons ! .....	612
<b>31. Oh, le bel héritage .....</b>	<b>614</b>
31.1. NVI – Un contrat est un contrat .....	614
Des problèmes d'héritage... .....	614
Non Virtual Interface .....	614
Changements dans les contrats .....	618
31.2. <code>protected</code> – la portée intermédiaire .....	619
31.3. Mettons un point final à cet héritage .....	622
31.4. L'héritage multiple .....	624
31.5. L'héritage, solution miracle ? .....	627

31.6. Récapitulons ! .....	628
<b>32. Les classes templates .....</b>	<b>630</b>
32.1. Les classes génériques .....	630
Définir une classe générique .....	630
Référencer une classe générique .....	632
Arguments types et non-types .....	632
Valeur par défaut .....	633
32.2. [Pratique] Un wrapper pour les pointeurs nus .....	634
32.3. Les traits .....	635
Exemple simple .....	635
Traits et programmation par contrat .....	636
En interne .....	638
32.4. Ces modèles sont bien trop génériques... .....	639
En voilà un bon concept ! .....	640
Créer ses propres concepts .....	641
32.5. Un peu de politique .....	644
À la base, une classe très hospitalière .....	645
Points de variation .....	645
Lier l'implémentation et les points de variation .....	647
32.6. Récapitulons ! .....	651
<b>33. Ça, c'est du SOLID ! .....</b>	<b>652</b>
33.1. S – Single responsibility principle .....	652
Le principe .....	652
Un exemple problématique... .....	653
...et sa solution .....	654
33.2. O – Open-closed principle .....	654
Le principe .....	654
Un exemple problématique... .....	655
...et sa solution .....	656
33.3. L – Liskov substitution principle .....	658
Le principe .....	658
Un exemple problématique... .....	658
...et sa solution .....	659
33.4. I – Interface segregation principle .....	659
Le principe .....	659
Un exemple problématique... .....	660
...et sa solution .....	661
33.5. D – Dependency inversion principle .....	663

Le principe .....	663
Un exemple problématique... ..	664
...et sa solution .....	665
33.6. Pour quelques principes de plus .....	666
33.7. Récapitulons ! .....	667
<b>34. Le voyage ne fait que commencer .....</b>	<b>668</b>
34.1. Davantage sur C++ .....	668
34.2. Davantage sur le développement .....	669
<b>[Pratique] Gérer les erreurs d'entrée • Solutions .....</b>	<b>670</b>
Index .....	680