

Alain Gibaud

Programmer en C++

Des premiers pas à la maîtrise de C++20



ellipses

Table des matières

I Généralités	15
1 À propos de ce livre	17
1.1 À qui s'adresse-t-il?	17
1.2 Que contient-il?	17
1.3 Les exemples	18
1.4 Pictogrammes utilisés	20
2 Évolution des langages C et C++	21
3 Informations à l'usage des programmeurs débutants	23
3.1 Qu'est-ce qu'un ordinateur?	23
3.2 Langages de programmation	31
4 Les environnements de développement	39
4.1 Choix d'un environnement	39
4.2 Choix d'un compilateur	40
4.3 Exemples d'utilisation du compilateur gcc	40
II Débuter avec le langage C	43
5 Composition des programmes en langage C	45
5.1 La notion de fonction en programmation	45
5.2 La fonction <code>main</code>	45
6 Les données et leur type	47
6.1 Variables et constantes	47
6.2 Déclaration des variables	47
6.3 Qu'est-ce qu'un type?	47
6.4 Les types primitifs	48
6.5 Données constantes	59
6.6 Données volatiles	62
7 Opérateurs	63
7.1 Opérateurs arithmétiques	63
7.2 Opérateurs sur bits	65
7.3 Opérateurs de décalage	66
7.4 Opérateurs logiques	67
7.5 Opérateurs relationnels	69
7.6 Opérateurs d'affectation	70
7.7 Opérateurs relatifs aux pointeurs	71
7.8 Opérateurs divers	72
7.9 Priorité et associativité des opérateurs	75
7.10 Ordre d'évaluation des opérandes	76
7.11 Expressions hétérogènes et conversion automatique de type	77
7.12 Promotion entière	78

8 Fonctions	79
8.1 Fonctions et sous-programmes	79
8.2 Paramètres réels et formels	80
8.3 Fonctions sans paramètre	83
8.4 Sous-programmes (fonctions ne retournant rien)	84
8.5 Fonctions à paramètres constants	85
8.6 Fonctions <code>inline</code>	85
9 Entrées et sorties	87
9.1 Qu'est-ce qu'une entrée ou une sortie?	87
9.2 Sorties : afficher à l'écran avec <code>printf</code>	87
9.3 Entrées : saisir des données au clavier avec <code>scanf</code>	90
10 Organiser les données	95
10.1 Tableaux	95
10.2 Structures	102
10.3 Unions	104
10.4 Choisir entre un tableau et une structure	105
10.5 Tableaux et structures comme paramètres de fonctions	106
11 Instructions	109
11.1 Les prédicats en C	109
11.2 Instructions conditionnelles (tests)	109
11.3 Instructions de répétition (boucles)	112
11.4 Sélection (<code>switch/case/default</code>)	118
11.5 Autres instructions contrôlant le flux d'exécution	121
III Approfondir le langage C	125
12 La fonction <code>main</code>	127
12.1 Arguments des programmes C (et C++)	127
12.2 Arguments comportant des caractères blancs	129
12.3 Valeur retournée par la fonction <code>main</code>	129
12.4 La fonction standard <code>exit</code>	129
13 Le préprocesseur du langage C	131
13.1 Qu'est-ce que le préprocesseur?	131
13.2 Inclure un fichier avec <code>#include</code>	132
13.3 Définir une macro avec <code>#define</code>	133
13.4 Suppression de macro avec <code>#undef</code>	138
13.5 Compilation conditionnelle avec <code>#if</code>	139
13.6 Autres <code>#</code> directives	141
14 Déclarations et définitions	143
14.1 Concepts de déclaration et de définition	143
14.2 Concevoir et analyser des déclarations complexes	144
15 Visibilité et durée de vie	151
15.1 Visibilité	151
15.2 Durée de vie	154

16 Pointeurs et tableaux	157
16.1 Arithmétique des pointeurs	157
16.2 Relations entre pointeurs et tableaux	160
16.3 Alors, tableaux et pointeurs sont la même chose?	162
16.4 Tableaux à indices décalés	162
16.5 Tableaux comme paramètres de fonction	163
17 Allocation dynamique de mémoire	175
17.1 Contexte	175
17.2 La fonction <code>malloc</code>	175
17.3 La fonction <code>free</code>	176
17.4 La fonction <code>realloc</code>	177
17.5 Erreurs relatives à l'allocation dynamique	178
17.6 Exemple	179
18 Récursivité	183
18.1 Coût de la récursivité	184
18.2 Quand utiliser la récursivité?	185
18.3 Exemples	186
18.4 Fonctions à la fois <code>inline</code> et récursives	190
IV Débuter avec le langage C++	191
19 Utiliser C++ sans créer de classe	193
19.1 Différences mineures entre C et C++	193
19.2 Invocation de fonctions C à partir de code C++	194
19.3 Création de fonctions C à partir de code C++	195
19.4 Opérateurs spécifiques	195
19.5 Priorité et associativité des opérateurs C++	196
19.6 Ordre d'évaluation des opérandes	196
19.7 La surcharge des fonctions	197
19.8 Fonctions avec arguments à valeur par défaut	198
19.9 Espaces de noms (<code>namespace</code>)	200
19.10 Introduction aux entrées-sorties sur flux	203
19.11 Les références	206
19.12 Quelques éléments de C++ moderne (C++11 à C++20)	213
19.13 Allocation dynamique de mémoire	222
20 Programmation par objets : les concepts	227
20.1 Classes et objets	227
20.2 Héritage	228
20.3 Encapsulation	230
20.4 Polymorphisme	231
21 Organisation des classes en C++	233
21.1 Des structures aux classes	233
21.2 Encapsulation	233
21.3 Modélisation des données	234
21.4 Modélisation des comportements	234
21.5 Organisation du code des classes	236

22 Modéliser un concept : la classe polynôme	239
22.1 Concept à modéliser	239
22.2 Données caractérisant ce concept	239
22.3 Modéliser les responsabilités : méthodes	240
22.4 Différence entre fonction et méthode	242
22.5 Le pointeur <code>this</code>	243
22.6 Méthodes statiques	245
22.7 Méthodes constantes	246
22.8 Fonctions, méthodes et classes amies (<code>friend</code>)	248
22.9 Méthodes <code>inline</code>	250
23 Construction et destruction	253
23.1 Problématique de l'initialisation	253
23.2 Construction	253
23.3 Destruction	263
23.4 Allocation dynamique de mémoire et construction	264
23.5 Restitution de mémoire et destruction	265
24 Accesseurs	267
24.1 Qu'est-ce qu'un accesseur?	267
24.2 Mutateur simple (<code>setter</code>)	267
24.3 Accesseur simple (<code>getter</code>)	267
24.4 Accesseur et attributs calculés	268
24.5 Accesseur et auto-extensibilité	268
24.6 Accesseur pour objet constant	271
24.7 Constance physique et logique, attribut <code>mutable</code>	272
24.8 <code>const_cast</code> contre <code>mutable</code>	273
24.9 Périmètre d'utilisation des accesseurs	274
25 Surcharge des opérateurs	275
25.1 Qu'est-ce que la surcharge des opérateurs?	275
25.2 Périmètre de la surcharge	275
25.3 Techniques de surcharge	276
25.4 Exemple de surcharge d'opérateurs par méthodes	278
25.5 Exemple de surcharge d'opérateurs par fonctions	279
25.6 Exemple d'utilisation des opérateurs	281
25.7 Choisir le mode d'implantation d'un opérateur	281
25.8 Choisir le mécanisme de passage des arguments	282
25.9 Choisir le mécanisme de passage du résultat de l'opération	283
25.10 Les opérateurs importants	283
25.11 Autres opérateurs	294
26 Conversions de types	303
26.1 Conversion implicite	303
26.2 Conversion par constructeur, spécification <code>explicit</code>	305
26.3 Conversion par opérateur	307
27 Héritage	311
27.1 Réutilisation d'une classe	313
27.2 Héritage public	314
27.3 Héritage privé	317
27.4 Héritage multiple	321

28 Polymorphisme	323
28.1 Type statique et type dynamique	323
28.2 Méthodes virtuelles	324
28.3 Périmètre du polymorphisme	325
28.4 Opérateurs polymorphes	325
28.5 Polymorphisme et destruction	326
28.6 Classes abstraites et interfaces	328
28.7 Le polymorphisme en action	330
28.8 Du bon usage du polymorphisme	336
29 Généricité	339
29.1 Fonctions génériques	339
29.2 Comment la généricité fonctionne-t-elle en C++ ?	340
29.3 Périmètre de la généricité	341
29.4 Instanciations implicite et explicite	341
29.5 Classes et structures génériques	343
29.6 Composants génériques à paramètres entiers	347
29.7 Paramètres de modèles avec valeurs par défaut	352
29.8 La généricité en action	353
30 Exceptions	365
30.1 Problématique de gestion des erreurs	365
30.2 Gestion des erreurs basée sur les exceptions	368
31 Bibliothèque d'entrées-sorties	379
31.1 Structure et composition de la bibliothèque	379
31.2 Les différents états des flux	383
31.3 Les différents modes d'ouverture des flux	387
31.4 <code>istream</code>	387
31.5 <code>ifstream</code>	392
31.6 <code>istringstream</code>	393
31.7 <code>ostream</code>	395
31.8 <code>ofstream</code>	396
31.9 <code>ostringstream</code>	396
31.10 <code>iostream</code> , <code>fstream</code> et <code>stringstream</code>	397
31.11 Manipulateurs d'entrées-sorties	398
31.12 Créer ses propres manipulateurs	404
32 Expressions lambda (lambda-fonctions)	407
32.1 Qu'est-ce qu'une expression lambda ?	408
32.2 Expressions lambda « nommées »	409
32.3 Expressions lambda génériques	410
32.4 Type de retour d'une expression lambda	411
32.5 Expression lambda avec paramètres par défaut	411
32.6 Capture de contexte dans une expression lambda	412
32.7 Exemples	415
33 Bibliothèque générique standard (STL)	419
33.1 Qu'est-ce que la STL ?	419
33.2 Introduction aux conteneurs	419
33.3 Introduction aux itérateurs	420
33.4 Propriétés des conteneurs	424

34 Conteneurs de la STL	425
34.1 Le conteneur <code>vector</code>	425
34.2 Le conteneur <code>deque</code>	432
34.3 Le conteneur <code>array</code>	437
34.4 Le conteneur <code>string (basic_string<char>)</code>	439
34.5 Les conteneurs <code>forward_list</code> et <code>list</code>	445
34.6 Les conteneurs <code>map</code> et <code>multimap</code>	450
34.7 Les conteneurs <code>unordered_map</code> et <code>unordered_multimap</code>	459
34.8 Les conteneurs <code>set</code> et <code>multiset</code>	467
34.9 Les conteneurs <code>unordered_set</code> et <code>unordered_multiset</code>	468
34.10 Le conteneur <code>stack</code>	468
34.11 Le conteneur <code>queue</code>	472
34.12 Le conteneur <code>priority_queue</code>	473
34.13 Le mandataire <code>span (C++20)</code>	477
34.14 Le mandataire <code>string_view</code>	480
34.15 La bibliothèque <code>ranges (C++20)</code>	483
34.16 Composition de vues	486
35 Algorithmes de la STL	489
35.1 Introduction aux algorithmes	489
35.2 Algorithmes non modifiants	491
35.3 Algorithmes modifiants	495
35.4 Algorithmes de partitionnement et de tri	503
35.5 Algorithmes sur intervalles triés	507
35.6 Algorithmes ensemblistes	508
35.7 Algorithmes sur les tas (<i>heaps</i>)	512
36 Type <code>initializer_list</code>	515
36.1 Création d'un <code>initializer_list</code>	515
36.2 Interface du modèle <code>initializer_list</code>	515
36.3 Exemple	516
37 Liaison structurée (<i>Structured binding</i>)	517
37.1 Collections autorisant les liaisons structurées	518
37.2 Usages typiques	520
38 Pointeurs sur membres	521
38.1 Introduction	521
38.2 Exemple	522
39 Pratiquez le « C++ facile »	527
39.1 Limitez l'usage des pointeurs	527
39.2 Passez les paramètres par valeur ou par référence	528
39.3 N'utilisez pas le système d'entrées-sorties du C	528
39.4 N'utilisez pas les tableaux	528
39.5 N'utilisez pas les chaînes de caractères du langage C	529
39.6 Limitez l'usage du préprocesseur	530
39.7 Ne gérez pas la mémoire vous-même	531
39.8 Utilisez les conteneurs de la STL	531
39.9 Utilisez les exceptions	531
39.10 Pratiquez l'idiome RAII	531

V	Approfondir le langage C++	533
40	Construction et destruction	535
40.1	Les différentes sortes d'objets	535
40.2	Construction et destruction d'objets automatiques	536
40.3	Construction et destruction d'objets statiques	536
40.4	Construction et destruction d'objets dynamiques	536
40.5	Construction et destruction d'objets membres ou de base	536
40.6	Construction et destruction d'éléments d'un tableau	537
40.7	Construction et exception	538
41	Allocation dynamique	539
41.1	Opérateur <code>new</code> avec placement	539
41.2	Personnaliser <code>new</code> , <code>new[]</code> , <code>delete</code> et <code>delete[]</code>	541
42	Opérateurs	547
42.1	Opérateur <code>=</code>	547
42.2	Opérateurs <code>++</code> et <code>--</code>	549
42.3	Opérateur <code>-></code>	554
42.4	Opérateur <code>""</code>	558
42.5	Opérateur de comparaison à trois voies <code><=></code> (C++20)	563
42.6	Repliement d'opérateur	568
42.7	Les quatre manières de replier une expression	569
42.8	Quelques cas d'utilisation	570
42.9	Opérateurs nommés	573
43	Transtypages	577
43.1	<code>static_cast</code>	577
43.2	<code>const_cast</code>	578
43.3	<code>reinterpret_cast</code>	578
43.4	<code>dynamic_cast</code>	580
44	Identification de type à l'exécution (RTTI)	585
44.1	L'opérateur <code>typeid</code>	585
44.2	La classe <code>type_info</code>	586
45	Support du polymorphisme	591
45.1	Principe général	591
45.2	Organisation de la table des méthodes virtuelles	593
45.3	Invocation des méthodes virtuelles	594
45.4	Obtention du type d'un objet polymorphe	595
45.5	Conclusion	596
46	Héritage	597
46.1	Héritage, encapsulation et pointeurs	597
46.2	Héritage et passage de paramètres	600
46.3	Héritage et affectation	601
46.4	Affectation polymorphe	603
46.5	Héritage multiple	606
46.6	Héritage dynamique	615
46.7	Construction et destruction en cas d'héritage multiple ou virtuel	624

47	Généricité	625
47.1	Modèles d'alias (alias template)	625
47.2	Spécialisation des modèles	626
47.3	Spécialisations partielles des modèles	627
47.4	Modèles de paramètres de modèles en C++17	630
47.5	Guides de déduction	632
47.6	Modèles variadiques (variadic templates)	636
48	Déduction de type	645
48.1	Déduction de type dans les modèles	645
48.2	Déduction de type avec <code>auto</code>	648
48.3	Différence entre <code>auto</code> et <code>decltype</code>	649
49	Références aux rvalues	653
49.1	rvalue et lvalue : quelques rappels	653
49.2	Passage d'une rvalue en paramètre	654
49.3	Références aux rvalues	655
49.4	Transmissions par déplacement	656
49.5	Conversion en rvalue avec <code>std::move</code>	658
49.6	Méthodes à objet support déplaçable	660
50	Transmissions parfaites	663
50.1	Réduction des références et références universelles	663
50.2	Transtypage conditionnel avec <code>std::forward</code>	666
50.3	Exemple	667
50.4	Les transmissions parfaites dans la STL	671
51	Pointeurs intelligents (smart pointers)	673
51.1	Introduction	673
51.2	Pourquoi utiliser des pointeurs intelligents?	673
51.3	<code>unique_ptr</code>	673
51.4	<code>shared_ptr</code>	682
51.5	<code>weak_ptr</code>	688
52	Expressions lambda et closures	699
52.1	Implantation des expressions lambda	699
52.2	Type d'une closure	699
52.3	Rôle des closures	700
52.4	Capture généralisée	701
52.5	Capture par déplacement	702
52.6	Expressions lambda et références pendantes	703
52.7	Capture par valeur de l'objet support	705
52.8	Expressions lambda génériques et transmissions parfaites	706
53	Exceptions	709
53.1	Gestion des exceptions non interceptées	709
53.2	Spécifications d'exceptions avant C++11	710
53.3	Spécifications d'exceptions à partir de C++11	711
53.4	Méthodes générées automatiquement et exceptions	715
53.5	Écrire du code sûr vis-à-vis des exceptions	717
54	Itérateurs	729
54.1	Un itérateur pour le type intervalle	729
54.2	Propriétés des différentes catégories d'itérateurs	730
54.3	Conception de la classe <code>IRange</code> et de son itérateur	730
54.4	Rendre les itérateurs compatibles avec l'écosystème C++	734

55 Programmation concurrente et threads	737
55.1 Qu'est-ce qu'un thread?	737
55.2 Premiers pas avec les threads	738
55.3 Paramètres de construction d'un thread	741
55.4 Les différentes façons d'implanter un thread	742
55.5 Threads et exclusion mutuelle	744
55.6 Variables de condition et synchronisation	749
55.7 <code>jthread</code> (C++20)	752
56 Programmation concurrente et tâches	757
56.1 Création d'une tâche avec <code>async</code>	757
56.2 Détection de la terminaison d'une tâche	758
56.3 Tâches et exceptions	759
56.4 Fonctionnement des tâches (<code>promise</code> , <code>future</code>)	760
57 Objets exécutables asynchrones	763
57.1 Introduction aux <code>packaged_task</code>	763
57.2 Le modèle <code>function</code>	763
57.3 Le modèle <code>packaged_task</code>	764
57.4 <code>packaged_task</code> et exécution concurrente	765
58 Métaprogrammation	767
58.1 Introduction	767
58.2 SFINAE	769
58.3 Introspection	776
58.4 Métaprogrammation et bibliothèque standard	780
59 Concepts (C++20)	783
59.1 Utilisation d'un concept prédéfini	783
59.2 Création d'un concept	784
60 Modules (C++20)	791
60.1 Qu'est-ce qu'un module?	792
60.2 Fichiers décrivant les modules	793
60.3 Création d'un module simple	793
60.4 Modules et bibliothèque standard	795
60.5 Utilisation d'entités non importables	796
60.6 Sous-modules	798
60.7 Partitions	800
60.8 Modularisation d'un composant préexistant	803
60.9 Exportation et espaces de noms	804
60.10 Conception des modules	805