

Ressourcesinformatiques



+ QUIZ

Version en ligne

OFFERTE !

pendant 1 an

Langage C++

De l'héritage C au C++ moderne

(avec programmes d'illustration)

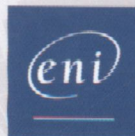
En téléchargement

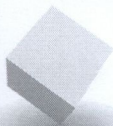


le code source

2^e édition

Frédéric DROUILLON





Ressourcesinformatiques



Langage C++

De l'héritage C
au C++ moderne

(avec programmes d'illustration)

2^e édition

جامعة أمحمد بوقرة - بومرداس
Université M'hamed Bougara - Boumerdes

المكتبة الجامعية

رقم الجرد: 0144517

جامعة أمحمد بوقرة - بومرداس
Université M'hamed Bougara - Boumerdes

المكتبة الجامعية

رقم: 004.431.120

12 ap.

Les éléments à télécharger sont disponibles à l'adresse suivante :

<http://www.editions-eni.fr>

Saisissez la référence ENI de l'ouvrage **RI2LCP** dans la zone de recherche et validez. Cliquez sur le titre du livre puis sur le bouton de téléchargement.

Introduction

1. L'univers C++	25
2. Cartographie pédagogique de la grammaire C++	26
3. Organisation du livre	27
4. Public visé	28
5. Clés pour apprendre	28
6. Documentations, versions, environnements	30

Partie 1 : Fondamentaux algorithmiques

Chapitre 1-1

Premiers programmes

1. C inclus en C++	33
2. Premiers programmes	35
2.1 La fonction main() : entrée du programme	35
2.2 "Hello world !" en C	36
2.3 "Hello world !" en C++	37
3. Bibliothèques, espaces de noms, instruction using	38
3.1 Bibliothèque <iostream>, directive #include	38
3.2 Espace de noms et instruction using	39
3.3 L'espace de noms standard std	41
3.4 Inclusion de bibliothèques C en C++	42
4. Activer C++20	42

Chapitre 1-2**Variables simples**

1. Introduction	45
2. Hérité du C	46
2.1 Différents types de variables	46
2.2 Déclaration des variables	48
2.3 Affecter et afficher une valeur	49
2.4 Opérateur sizeof	50
2.5 Préfixes 0b, 0x, 0 et séparateur (')	51
2.6 Suffixes L, LL et f	52
2.7 Caractères, codage ASCII	53
2.8 Renommer des types avec typedef	54
3. Apports du C++	55
3.1 Initialisations des variables	55
3.2 Type auto	57
3.3 Opérateur decltype	58
3.4 Opérateur typeid, bibliothèque <typeinfo>	59

Chapitre 1-3**Constantes**

1. Constantes const et constexpr	61
1.1 Hérité du C	61
1.2 Apports du C++	62
1.2.1 Plus de rigueur sur const	62
1.2.2 Valeurs constantes définies à la compilation : mot-clé constexpr	63
2. Énumérations : créations de types enum	65
2.1 Hérité du C	65
2.2 Apports C++	66
3. Directive macroprocesseur #define	68

Chapitre 1-4

Affichage et saisie console

1. Hérité du C	71
1.1 Affichage : fonction printf	71
1.1.1 Chaînes de caractères	71
1.1.2 Convertir des valeurs en caractères avec des formats ...	72
1.1.3 Paramétrer l'affichage de printf	74
1.2 Formatage : fonctions sprintf, snprintf, sprintf_s, snprintf_s ..	75
1.3 Saisie de valeurs : fonctions scanf, scanf_s	76
2. Apports C++	78
2.1 Utiliser cout et cin	78
2.1.1 Écrire dans la console avec cout	78
2.1.2 Entrer des valeurs avec cin	79
2.1.3 Contrôle d'erreur simple	81
2.2 Instructions de formatage en sortie	86
2.2.1 Afficher des valeurs booléennes	86
2.2.2 Largeur minimum de l'affichage	87
2.2.3 Alignement des sorties	87
2.2.4 Choisir un caractère de remplissage	89
2.2.5 Afficher ou non les zéros après la virgule	90
2.2.6 Afficher le signe des nombres positifs	91

Chapitre 1-5

Opérations

1. Hérité du C	93
1.1 Notion d'expression	93
1.2 Opérations arithmétiques	94
1.3 Valeurs aléatoires	95
1.3.1 Avoir des suites différentes	96
1.3.2 Définir une fourchette	97

1.4	Opérations bit à bit	97
1.4.1	ET : opérateur &	97
1.4.2	OU exclusif : opérateur ^	99
1.4.3	OU inclusif : opérateur 	99
1.4.4	Complément : opérateur ~	100
1.4.5	Décalages gauche et droite : opérateurs >> et << ...	100
1.4.6	Priorités des opérateurs bit à bit	100
2.	Apports C++	101
2.1	Surcharger les opérateurs	101
2.2	La bibliothèque <random>	101

Chapitre 1-6**Conversions de types**

1.	Hérité du C	105
1.1	Le principe	105
1.2	Opérateur de conversion	107
1.3	Exemple : nombre aléatoire avec décimales	107
2.	Apports du C++	108
2.1	static_cast<type>	109
2.2	const_cast<type>	110
2.3	reinterpret_cast<type>	110
2.4	dynamic_cast<type>	112

Chapitre 1-7**Structures de contrôle**

1.	Hérité du C	113
1.1	Bloc d'instructions	114
1.2	L'instruction conditionnelle if	114
1.3	Définir une condition	116
1.3.1	Opérateurs de comparaisons	116
1.3.2	L'opérateur unaire NON : !	117

1.4	Le couple d'instructions if-else	117
1.5	La forme contractée du if-else, opérateur conditionnel "?" . . .	119
1.6	La cascade d'instructions if - else if - else	120
1.7	Les tests multiconditions (ET/OU)	121
1.7.1	Conjonction ET : opérateur &&	121
1.7.2	ET avec plus de deux expressions membres	122
1.7.3	Disjonction OU, opérateur 	124
1.7.4	OU avec plus de deux expressions membres	125
1.7.5	ET prioritaire sur OU	126
1.8	Branchement : switch	126
1.9	Rupture de séquence : goto avec étiquette	129
1.10	Les trois boucles : while, do-while et for	130
1.10.1	Boucle TANT QUE : le while	130
1.10.2	Boucle FAIRE {...} TANT QUE : le do-while	132
1.10.3	Boucle comptée POUR : le for	133
1.10.4	Boucles imbriquées	135
1.11	Sortie et saut forcés dans une boucle	136
1.11.1	Sortir avec l'instruction break	136
1.11.2	Passer à l'itération suivante avec l'instruction continue	137
1.11.3	Sortir d'une ou de plusieurs boucles imbriquées avec l'instruction goto	137
2.	Apports du C++	138
2.1	Condition if ou else-if marquée constexpr	138
2.2	Boucle for (:) "pour chaque"	139

Chapitre 1-8

Fonctions

1. Hérité du C	141
1.1 Principe	141
1.2 Fonction sans retour et sans paramètre	143
1.3 Déclaration et visibilité d'une fonction	144
1.4 Fonctions avec paramètres	146
1.4.1 Copie de valeurs et passage par valeur	146
1.4.2 Déplacer le curseur en écriture de la fenêtre console ..	147
1.4.3 Changer la couleur des caractères	150
1.4.4 À propos de la couleur en console	151
1.4.5 Afficher une lettre à une position et d'une couleur donnée dans la console	152
1.4.6 Tracer une ligne horizontale	153
1.4.7 Le programme complet	154
1.5 Fonctions avec retour	155
1.5.1 Retourner un nombre aléatoire entre 0 et 1	155
1.5.2 Retourner le résultat d'un jeté de deux dés à six faces ..	156
1.5.3 Programme complet	156
1.5.4 Types des valeurs de retour	157
1.6 Fonctions avec retour et avec paramètres	158
1.6.1 Conversion en chiffres romains	158
1.7 Fonction et qualificatif static	162
1.7.1 Qualificatif static et fonctions	162
1.8 Fonctions récursives	164
1.9 Nommer les fonctions	165
1.9.1 Contraintes syntaxiques	165
1.9.2 Sens des noms choisis	166
1.9.3 Normes et styles pour la mise en forme du code source	166
2. Apports du C++	167
2.1 Fonctions embarquées "inline"	167
2.2 Fonctions constantes marquées constexpr	168

2.3	Fonctions marquées const	170
2.4	Fonctions déclarées noexcept	171
2.5	Valeurs par défaut de paramètres	171
2.6	Surcharge des fonctions	173
2.7	Références et passage par référence	174
2.8	Expression lambda (fonction anonyme)	176
2.8.1	Principe	176
2.8.2	Syntaxe générale	177
2.8.3	Les clauses de capture	179
2.8.4	Clauses de capture multiples	181
2.8.5	Spécification mutable	182
2.8.6	Listes de paramètres	183
2.8.7	Spécification d'une exception	183
2.8.8	Le type de retour	184
2.8.9	Lambda constexpr	186
2.8.10	Récupérer une lambda et appels possibles	187
2.8.11	Quelques précisions	188
2.9	Fonction operator et surcharge des opérateurs	191
2.10	Fonctions génériques (template, auto)	191

Chapitre 1-9

Précisions sur les variables, attributs

1.	Précisions sur les variables	193
1.1	Visibilité et durée de vie d'une variable	193
1.2	Masquage d'une variable	195
1.3	Variable static	197
2.	Attributs	198
2.1	Principe et syntaxe	198
2.2	Exemples d'utilisation	200
2.2.1	Expérimentation GNU	200
2.2.2	Expérimentation GSL Microsoft	200

2.3	Attributs standards	204
2.3.1	[[noreturn]] (depuis C++11)	204
2.3.2	[[fallthrough]] (depuis C++17)	205
2.3.3	[[deprecated]], [[deprecated("reason")]] (depuis C++14)	207
2.3.4	[[nodiscard]] (depuis C++17) [[nodiscard("reason")]] (depuis C++20)	208
2.3.5	[[maybe_unused]] (depuis C++17)	208
2.3.6	[[likely]] et [[unlikely]] (depuis C++20)	209
2.3.7	[[no_unique_address]] (depuis C++20)	210

Chapitre 1-10

Structures et classes

1.	Hérité du C	213
1.1	Principe de la structure.	213
1.2	Disposer d'une structure	214
1.2.1	Définir un type de structure.	214
1.2.2	Déclarer une variable structure.	214
1.2.3	Initialiser à la déclaration	215
1.2.4	Accéder aux éléments avec l'opérateur point.	216
1.2.5	Copier deux structures	216
1.3	Structures et fonctions.	216
1.3.1	Fonction d'initialisation d'une entité	217
1.3.2	Fonctions d'affichage et d'effacement	217
1.3.3	Fonction pour avancer.	218
1.3.4	Boucle événementielle.	219
1.3.5	Contrôle du rythme de l'animation	220
1.3.6	Rendre invisible le curseur en écriture	221
1.3.7	Démo entité mobile.	222
1.4	Structures imbriquées	225
2.	Apports du C++ : la structure devient une classe.	226

Partie 2 : Entrée en territoire Objet

Chapitre 2-1 Classes

1. De la structure à la classe	227
1.1 Des droits d'accès : public, protected, private	228
1.2 Des fonctions membres	231
1.2.1 Cas général	231
1.2.2 Fonctions membres déclarées const	232
1.3 Une initialisation renforcée	233
1.3.1 Initialisations à la déclaration	233
1.3.2 Initialisations par défaut dans la classe	234
1.4 Des constructeurs et des initialiseurs	235
1.4.1 Initialiser avec des constructeurs	235
1.4.2 Constructeur par défaut	236
1.4.3 Initialisations contraintes par les constructeurs existants	237
1.4.4 Constructeurs avec initialiseurs	238
1.4.5 L'initialiseur initialise les constantes	239
1.5 Le pointeur this	240
1.6 Un programme C muté objet C++	242
1.6.1 Réécriture des entités mobiles	242
1.6.2 Amélioration du programme	248
1.7 Que dire des structures en C++ ?	253
1.7.1 Des aspects pratiques	253
1.7.2 Réfléchir l'interface des objets	254
2. Accesseurs (property en anglais)	254
2.1 Lire une variable private ou protected	255
2.2 Modifier une variable private ou protected	256
2.3 Intérêt d'un appel de fonction	256
3. Qualificatif static dans une classe	259

4.	default et delete pour les fonctions spéciales de classe.	260
4.1	Fonctions spéciales.	260
4.2	Utiliser default (rappel et précision)	261
4.3	Utiliser delete	263
5.	Surcharge des opérateurs	264
5.1	Principe	264
5.2	Fonction operator hors classe	265
5.2.1	Exemple addition : operator+	265
5.2.2	Exemple décalage à gauche : operator<<	266
5.3	Fonction operator dans une classe.	269
5.3.1	operator+ sans retour	269
5.3.2	operator+ avec retour de l'objet courant (*this)	270
5.3.3	opérateur+ avec retour d'un nouvel objet résultant.	272
5.4	Transformer un objet en une fonction (objet fonction ou fonction objet).	273
5.5	Utiliser l'opérateur bool	275
6.	L'instruction friend ("ami")	276

Chapitre 2-2

Unions, unions illimitées

1.	Principe	279
2.	Union de structures	282
3.	Union discriminée	283
4.	Union illimitée (C++11)	287

Chapitre 2-3**Généricité, template, auto**

1. Principe du template.....	291
2. Template de fonction.....	292
2.1 Définir une fonction générique.....	292
2.2 Fonction avec plusieurs types génériques.....	294
2.3 Évaluation des types génériques à la compilation.....	295
2.4 Expressions lambda et template.....	297
2.5 Template pour passer des valeurs.....	298
3. Template de classe.....	300
3.1 Syntaxe.....	300
3.1.1 Syntaxe générale.....	300
3.1.2 Syntaxe des constructeurs.....	301
3.1.3 Syntaxe avec plusieurs types génériques.....	302
3.1.4 Déduction de type à l'initialisation.....	302
3.1.5 Types par défaut.....	303
3.1.6 L'instance du template influence le type de la classe.....	304
3.2 Paramétrages.....	305
3.2.1 Paramétrage avec des valeurs.....	305
3.2.2 Template en paramètre de template.....	308
3.2.3 Spécialiser une fonction pour un type donné.....	310
3.2.4 Spécialiser une classe complète.....	312
3.2.5 Spécialiser une classe avec plusieurs types génériques.....	314
3.2.6 Paramétrage avec des objets fonctions.....	316
4. Template variadique et paquet de paramètres.....	317
4.1 Principe : l'opérateur variadique (...).	317
4.2 Template variadique de fonction.....	320
4.2.1 Cas général.....	320
4.2.2 Itérations simplifiées (fold expression, C++17).....	322
4.2.3 Deux paramètres variadiques.....	323
4.3 Template variadique de classe.....	326

5. Utilités standards à base de templates	327
5.1 La classe bitset	328
5.2 La classe tuple<...>	329
5.3 La classe pair<>	333
5.4 Alternatives à l'union	336
5.4.1 La classe variant	336
5.4.2 La classe optional	339
5.4.3 La classe any	341
5.5 La classe std::function<>	343
6. Concepts (C++20)	344
6.1 Définir un concept	345
6.2 L'expression requires	348
6.2.1 Exemple d'utilisation sans paramètre	348
6.2.2 Exemple d'utilisation avec paramètres	350
6.3 Établir des contraintes	351
6.3.1 Conjonction ET (&&)	352
6.3.2 Disjonction OU ()	354
7. Spécification auto	356
7.1 Principes généraux	356
7.1.1 Variables simples auto	356
7.1.2 Listes auto	359
7.2 Templates et auto	360
7.2.1 Spécification decltype(auto)	360
7.2.2 Comparaison entre auto et template	362
7.2.3 Utilisation de auto dans un template	363

Chapitre 2-4**Bibliothèques, modules et espaces de noms**

1. Créer une bibliothèque classique (.h)	367
1.1 Contenu du fichier outils.h	370
1.2 Contenu du fichier outils.cpp	371
1.3 Contenu du fichier main.cpp	372
1.4 Problèmes de redéfinitions, instruction extern	375
2. Espaces de noms (namespace)	377
2.1 Cartographier du code	377
2.2 Accéder au contenu	378
2.3 Répartition sur plusieurs fichiers	379
2.3.1 Espace de noms dans une bibliothèque	380
2.3.2 Espace de noms réparti entre plusieurs fichiers	381
2.3.3 Espace de noms ToutpourlesChiens réorganisé	385
2.4 Imbriquer des espaces de noms	387
2.5 Espaces de noms inline	388
2.6 Espace de noms anonymes	391
2.7 Remplacer un nom trop long par un alias	392
3. La directive using	392
3.1 Faciliter l'accès à un espace de noms	392
3.2 Constituer un alias de type	395
4. Modules (C++20)	397
4.1 Principe	397
4.2 Créer une unité d'interface de module (.ixx)	398
4.3 Définir le contenu du module	400
4.4 Importer le module dans un fichier source	401
4.5 Utiliser des bibliothèques standards dans un module	402
4.6 Utiliser une bibliothèque personnelle dans un module	405
4.7 Éléments non exportables dans un module	405
4.8 Partitionner un module	406
4.9 Espaces de noms dans un module	410
4.10 Créer un fichier d'implémentation de module	412

4.11 Création d'un module Outils (version 1)	414
4.12 Création d'un module Outils (version 2)	416
4.13 Modules ou bibliothèques ?	417
5. Précision sur la liaison entre C++ et C	418

Partie 3 : Puissance du pointeur et gestion de la mémoire

Chapitre 3-1

Tableaux statiques, introduction conteneurs

1. Hérité du C	421
1.1 Principe du tableau	421
1.2 Disposer d'un tableau	422
1.2.1 Définir et déclarer un tableau	422
1.2.2 Des constantes pour les tailles	423
1.2.3 Accéder aux éléments du tableau avec l'opérateur [] ..	423
1.2.4 Débordement de tableau	425
1.2.5 Parcourir un tableau avec une boucle for	425
1.2.6 Initialiser un tableau à la déclaration	428
1.3 Tableaux à plusieurs dimensions	430
1.3.1 Matrice à deux dimensions	430
1.3.2 Tableaux à n dimensions	431
1.3.3 Initialiser à la déclaration	432
1.3.4 Parcourir un tableau à plusieurs dimensions	432
1.4 Tableaux en paramètre de fonctions	433
2. Apports du C++	434
2.1 Boucle for (:) "pour chaque"	434
2.2 Récupérer une liste variadique d'éléments dans un tableau statique	436
2.3 Tableaux d'objets, exemple fourmilière (version 1)	438
2.4 Tableau dans une classe, fourmilière (version 2)	443
2.4.1 Paramétrer la classe	443
2.4.2 La classe fourmilière	444

2.5	Tableau dans une classe, exemple pile générique	446
2.5.1	Création de la pile	446
2.5.2	Tri paramétrable de la pile (objet fonction)	448
2.5.3	Spécialisation sur un type	452
3.	Introduction des conteneurs	454
3.1	La classe array	455
3.2	La classe vector	457
3.3	La classe list	460
3.4	Contenus variadiques	463
3.5	Boucle <code>for_each</code> de la bibliothèque <code><algorithm></code>	465

Chapitre 3-2

Chaînes de caractères, la classe `string`

1.	Hérité du C	469
2.	Apports du C++ : la classe <code>string</code>	472

Chapitre 3-3

Pointeurs

1.	Principes généraux	475
1.1	Adresse et mémoire	475
1.2	Une variable pointeur	477
1.3	Quatre opérateurs dédiés	479
1.4	Cinq utilisations classiques	479
2.	Hérité du C	481
2.1	Déclarer un pointeur dans un programme	481
2.2	Opérateur adresse : <code>&</code>	482
2.3	Opérateur étoile : <code>*</code>	483
2.4	Opérateur flèche : <code>-></code>	484
2.5	Opérateur crochets : <code>[]</code>	486
2.6	Priorité des quatre opérateurs	487

2.7	Pointeurs et constantes	487
2.7.1	Pointeur variable sur un objet constant	487
2.7.2	Pointeur constant sur un objet variable	488
2.7.3	Pointeur constant sur un objet constant	489
2.8	Le pointeur générique void*	490
2.9	Pointeurs de fonction	491
2.9.1	Une fonction est une adresse	491
2.9.2	Reconnaître le type d'une fonction	492
2.9.3	Appeler une fonction via un pointeur du bon type. ...	493
2.9.4	Cast nécessaire si le pointeur est un void*	493
2.9.5	Pourquoi des pointeurs de fonction ? Les « callbacks » ..	494
3.	Apports du C++	497
3.1	Un souci de rigueur	497
3.1.1	Plus de rigueur	497
3.2	Opérateurs new et delete, new[] et delete[]	499
3.2.1	Principe	499
3.2.2	Respecter les couples new-delete et new[]-delete[]	501
3.2.3	Allocation d'un tableau de 0 élément ?!	501
3.2.4	Précisions sur l'opérateur new	503
3.3	La valeur nullptr	508
3.4	Type référence & (pointeur constant et simplifié)	510
3.4.1	Principe	510
3.4.2	Une référence est constante	513
3.4.3	Référence déclarée const	513
3.4.4	Connaître la valeur de la référence	514
3.5	Retourner une référence	515
3.5.1	Piège à éviter	516
3.5.2	Masquer un tableau avec une fonction	517
3.6	Type référence && (Right value)	518
3.6.1	Des catégories d'expressions	518
3.6.2	lvalues (objets persistants) et rvalues (objets temporaires)	519
3.6.3	Affectations d'objets temporaires rvalues	521

3.6.4	Objets rvalues passés en paramètre de fonction	524
3.6.5	Remarque : une ambiguïté possible	526
3.6.6	Déclarateur de référence rvalue : &&	527
3.6.7	Référence rvalue && en paramètre de fonction	529
3.6.8	Référence rvalue && en paramètre de constructeur	530
3.7	Conversions en rvalue	531
3.7.1	Un <code>static_cast<></code>	531
3.7.2	La fonction de déplacement <code>std::move</code>	532
3.7.3	La fonction de transfert <code>std::forward<></code>	534
3.7.4	Situation de conversion implicite de rvalue(&&) à lvalue (&)	538
3.7.5	Liste variadique et transfert parfait	540
3.8	Spécificateur <code>decltype</code> et références	541

Chapitre 3-4

Pointeurs et références dans la classe

1.	Introduction	547
2.	Le destructeur	548
2.1	Les éléments non dynamiques s'autodétruisent (rappel)	548
2.2	Problème des éléments dynamiques perdus en mémoire	549
2.3	Écrire un destructeur des éléments dynamiques	550
2.4	Appels explicites du destructeur	552
2.5	Destructeur appelé avec <code>delete</code>	554
3.	Constructeur de copie	558
3.1	Principe	558
3.2	Copier un objet à la déclaration	558
3.3	Copier des objets possédant des données dynamiques	559
3.3.1	Problème des données dynamiques	559
3.3.2	Interdire la copie	561
3.3.3	Implémenter un constructeur de copie	561

4. Constructeur de déplacement	564
4.1 Principe	565
4.2 Possible avertissement noexcept	567
4.3 Constructeur de déplacement avec données dynamiques	568
5. Surcharge des opérateurs et données dynamiques	573
5.1 Affectation de copie (operator=)	573
5.2 Affectation de déplacement (move)	579
5.3 Surcharge operator+	584
6. Questions diverses	586
6.1 Spécialiser une classe générique en pointeur	586
6.2 Créer un singleton	588
6.3 Références dans une classe	590

Chapitre 3-5

Pointeurs-objets ou "smart pointers"

1. Introduction	593
1.1 La bibliothèque memory	593
1.2 Transformer un pointeur en objet	594
2. La classe std::unique_ptr	597
2.1 Expérimentation du pointeur unique	597
2.2 Survol du contenu principal	599
2.2.1 Constructeurs	599
2.2.2 Destructeur	600
2.2.3 Six surcharges d'opérateurs	602
2.2.4 Cinq fonctions membres	605
2.2.5 Quatre fonctions hors classe	607
3. La classe std::shared_ptr	608
3.1 Expérimentation du pointeur partagé	609
3.2 Survol du contenu principal	611
3.2.1 Constructeurs	611
3.2.2 Destructeur	613

3.2.3	Six surcharges d'opérateurs	616
3.2.4	Cinq fonctions membres	619
3.2.5	Fonctions hors classe	623
3.2.6	Fonctions hors classe de conversion	629
4.	La classe <code>std::weak_ptr</code>	638
4.1	Expérimentation du pointeur "simple observateur"	638
4.2	Survol du contenu principal	640
4.2.1	Constructeurs	640
4.2.2	Destructeur	643
4.2.3	Six fonctions membres	644
4.2.4	Surcharge opérateur <code>=</code>	649
5.	Allocateur de mémoire : la classe <code>std::allocator</code>	651

Chapitre 3-6

Pointeurs, utilisations classiques

1.	Introduction	655
2.	Pointeur en paramètre (passage par référence)	655
2.1	Expérimentation avec pointeur classique	656
2.2	Simplification de l'écriture avec des références C++	657
2.3	Tableau en paramètre	658
2.3.1	Tableau à une dimension	658
2.3.2	Tableaux à plusieurs dimensions	660
3.	Allocation dynamique et tableaux	661
3.1	Tableau de pointeurs	661
3.2	Allocation d'un tableau à une dimension	663
3.3	Allocation d'un tableau à plusieurs dimensions	664
3.3.1	Allocation d'une matrice de <code>int</code>	664
3.3.2	Allocation d'une forme à six dimensions de points	666
3.4	Quelques pièges classiques de l'allocation dynamique	669

4. Relier des objets	670
4.1 Associer deux types structures style C	671
4.1.1 Structures Avion et Pilote	671
4.1.2 Avion ou pilote existent-ils ?	672
4.1.3 Avion ou pilote sont-ils libres ?	672
4.1.4 Relier avion et pilote	673
4.1.5 Déliaison un avion et un pilote	673
4.1.6 Constructeurs Avion et Pilote	674
4.1.7 Afficher Avion et Pilote	675
4.1.8 Détruire un Avion ou un Pilote	676
4.1.9 Code complet, main action	676
4.2 Associer des classes de même type	681
4.3 Élaborer une liste, fourmilière (version 3)	684

Partie 4 : Programmation Orientée Objet (POO)

Chapitre 4-1

Associations entre classes

1. Introduction	693
2. Principes des associations pour les relations entre objets	693
2.1 Association simple	694
2.2 Agrégation	694
2.3 Composition	695
2.4 Problème syntaxique en C++	695
3. Associations simples : messages entre objets	696
3.1 Liaison non réciproque entre deux objets	696
3.2 Liaison réciproque entre deux objets	698

4. Agrégations : coopération entre objets	702
4.1 Liaison à sens unique (exemple guitare, guitariste)	702
4.2 Partage d'objets pointés (plusieurs musiciens, une guitare)...	704
4.2.1 Musiciens simultanés	705
4.2.2 Plusieurs musiciens successifs	707
5. Liaisons réciproques entre objets	709
5.1 Problème de syntaxe	709
5.2 Déclaration de type incomplet	710
5.3 Limite du type incomplet	711
5.4 Résolution du problème	712
5.5 Exemple Terminator	716
6. Composition : dépendance entre objets	718
6.1 Choisir entre agrégation ou composition	718
6.2 Techniques envisageables	718
6.3 Pointeur d'objet en propriété	719
6.4 Objet en propriété	721
6.5 Référence d'objet en propriété	725

Chapitre 4-2

Héritage

1. Principe	727
2. Définir une classe dérivée	728
3. Appeler explicitement un constructeur de la classe de base	730
4. Redéfinition de données ou de fonctions	731
5. Spécifier un membre de la classe de base	733
6. Droits d'accès locaux de la classe héritée	734
7. Droits d'accès globaux de la classe héritée	737
8. Héritage multiple	742
8.1 Principe et syntaxe	742
8.2 Exemple : InDominusRex	743

8.3	Relations transversales dans un arbre de classes	745
8.4	Héritage multiple avec une base virtuelle	748
9.	Comment identifier un héritage	752
9.1	Distinction entre héritage et association	752
9.2	Distinction entre attributs et nouvelle classe	754

Chapitre 4-3

Polymorphisme et virtualité

1.	Principe	755
2.	Accès pointeurs limité par son type	755
3.	Autorisation d'accès pour les fonctions virtuelles	757
4.	Destructeur virtuel	758
5.	Intérêt des fonctions virtuelles	759

Chapitre 4-4

Classe abstraite et interface

1.	Classe abstraite, fonctions virtuelles pures	763
2.	Tronc commun pour dériver	766
3.	Interface	770
4.	Récupérer une sous classe depuis une base abstraite	773
5.	Résumé classe abstraite et interface	776
6.	Expérimentation : exemples des super-héros, les Avengers	777
6.1	Classe de base super-héros, interface des fonctions d'action ..	777
6.1.1	Définition et stockage des armes	778
6.1.2	Définition et stockage du paramétrage émotionnel ...	779
6.1.3	Interface de fonctions pour tous les super-héros.	779
6.1.4	Définition de la classe "SuperHeros"	780

6.2	Une dérivée pour chaque Avenger	782
6.2.1	Classe "CaptainAmerica"	783
6.2.2	Classes Hulk, IronMan, BlackWidow et Thor.	784
6.3	Souhait du polymorphisme et impossibilité.	787
6.4	Virtualité, intérêt de l'interface des fonctions d'action	789
6.5	Classe abstraite	792

Chapitre 4-5

Gestion des erreurs

1.	Introduction	803
2.	Socle hérité du C.	804
2.1	Retourner un booléen.	804
2.2	Retourner un numéro d'erreur.	809
2.3	Afficher des informations au moment de l'erreur	811
2.4	Bibliothèques C de contrôle d'erreur	815
2.4.1	La bibliothèque <cerrno> (errno.h)	816
2.4.2	La bibliothèque <cassert> (assert.h)	817
3.	Contrôle d'erreur C++	819
3.1	Introduction	819
3.1.1	Bibliothèques de diagnostics	819
3.1.2	Conseil sur les exceptions	820
3.2	Instructions natives throw, try et catch.	821
3.2.1	Retour throw	821
3.2.2	Saut try et récupération catch	822
3.2.3	Retour throw d'un appel de fonction.	823
3.2.4	Instruction throw sans valeur de retour	826
3.2.5	Bloc catch(...) par défaut	826
3.2.6	Exception non gérée	827
3.2.7	Fonctions déclarées noexcept	829

3.3	L'en-tête <code><exception></code>	830
3.3.1	Principe.	830
3.3.2	Classe <code>std::exception</code>	833
3.3.3	Les fonctions <code>std::terminate</code> , <code>std::set_terminate</code> , <code>std::get_terminate</code>	837
3.3.4	Utiliser la virtualité de la classe <code>std::exception</code>	839
3.4	Déclaration <code>static_assert</code>	844
Index		849

Langage C++

De l'héritage C au C++ moderne

Le C++ fascine par l'infini des réalisations et des manières de programmer qu'il permet. D'une part, il hérite du **langage C** qui lui procure une base pour tout ce qui concerne l'**écriture algorithmique**, et d'autre part il est doté de puissants **outils de structuration orientés objet**.

Ce livre s'adresse aux personnes qui débutent en programmation avec le langage C++ ou ayant déjà programmé dans d'autres langages. Il tient compte des **dernières évolutions de C++** et éclaire la complexité du langage par une exploration systématique de ses **composantes natives** (algorithmique fondamentale, programmation objet, généricité) auxquelles sont ajoutés quelques éléments sélectionnés parmi les **bibliothèques standards** (*smart pointers* ou pointeurs intelligents, conteneurs et contrôle d'erreur avec les exceptions).

La progression pédagogique de l'auteur se déroule en quatre temps.

L'auteur commence par traiter de notions qui concernent l'**écriture algorithmique** et ses fondamentaux : variables, opérations, structures de contrôle (if, switch, boucles, fonctions, attributs). Dans un second temps, le lecteur bascule vers la **programmation objet** avec les structures et les classes puis il se concentre sur la **généricité**, les **bibliothèques personnelles**, les **modules** et les **espaces de noms**. Ensuite, l'étude des **pointeurs**, introduite avec les **tableaux** et les **chaînes de caractères**, est développée d'abord avec les **pointeurs classiques** puis avec les pointeurs-objets dits **smart pointers**. La quatrième et dernière étape vient compléter l'arsenal des outils natifs de la programmation objet avec les concepts d'**héritage**, de **polymorphisme**, de **virtualité**, de **classes abstraites** et d'**interfaces**. Pour finir, le livre s'achève sur la **gestion des erreurs**.

Tout au long du livre, l'auteur fournit des **exemples de programmes** apportant la mise en pratique nécessaire pour s'approprier le langage. Tous les programmes sont en téléchargement sur le site www.editions-eni.fr.

Frédéric DROUILLON est enseignant, chercheur et aussi créateur dans le domaine de l'informatique et des arts numériques. Il envisage la programmation et ses langages comme un moyen d'expression et de formulation d'idées, une véritable écriture, qui fait naître de la recherche et de la créativité. L'objectif de son enseignement en langage C++ est de permettre au lecteur d'acquérir un savoir-faire fondamental en programmation informatique qui lui procurera très certainement les moyens de circuler dans toutes les autres techniques et langages de programmation.



sur www.editions-eni.fr :

→ Le code source des programmes d'illustration.

Pour plus d'informations :



34,50 €

ISBN : 978-2-409-03893-8



9 782409 038938



www.editions-eni.fr