



Ressourcesinformatiques

C++

Les fondamentaux du langage

2^e édition

Brice Arnaud GUÉRIN

Fichiers complémentaires
à télécharger



Les éléments à télécharger sont disponibles à l'adresse suivante :
<http://www.editions-eni.fr>
Saisissez la référence ENI de l'ouvrage **RI2CPP** dans la zone de recherche et validez. Cliquez sur le titre du livre puis sur le bouton de téléchargement.

Avant-propos

- 1. Objectifs de ce livre 11
- 2. Travaux pratiques. 12
 - 2.1 Le langage de script Lambda Basic. 12
 - 2.2 Le tableur InCell 13
 - 2.3 Les pages web dynamiques EZ-Pages 14
- 3. À qui s'adresse ce livre ? 15

Chapitre 1 Introduction

- 1. Notions clés. 17
 - 1.1 Principales caractéristiques du langage C++ 17
 - 1.2 Programmation orientée objet 19
 - 1.3 Environnement de développement et fichier makefile 21
 - 1.3.1 Choix d'un EDI 21
 - 1.3.2 Construction d'un fichier makefile 22
 - 1.4 Organisation d'un programme C++ 26
 - 1.4.1 Codes sources 27
 - 1.4.2 Modules objets 30
 - 1.4.3 Bibliothèques (librairies) 33
 - 1.4.4 Exécutable 34
 - 1.5 Préprocesseur. 35
 - 1.6 Choix d'un compilateur 36
 - 1.7 Éditeur de liens 37

2.	Bases de la programmation C++	37
2.1	Déclaration de variables	38
2.1.1	Utilité des variables	38
2.1.2	Portée des variables	40
2.1.3	Syntaxe de déclaration	42
2.1.4	Types de données	42
2.2	Instructions de tests et opérateurs	51
2.2.1	Instructions de tests	51
2.2.2	Opérateurs	54
2.3	Instructions de boucle	60
2.3.1	La boucle for	60
2.3.2	La boucle while	62
2.3.3	La boucle do	62
2.3.4	Les instructions de débranchement	63
2.4	Tableaux	63
2.5	Fonctions et prototypes	65
2.5.1	Déclaration d'une fonction	66
2.5.2	Fonctions et procédures	66
2.5.3	Appel des fonctions	68
2.5.4	Gestion des variables locales	68
2.5.5	Définition de fonctions homonymes (polymorphisme)	69
2.5.6	Fonctions à nombre variable d'arguments	70
2.5.7	Attribution de valeurs par défaut aux arguments	72
2.5.8	Fonctions en ligne	73
2.5.9	Fonctions externes de type C	73
2.5.10	Fonctions récursives	74
2.5.11	La fonction main()	75
2.6	Pointeurs	77
2.6.1	Pointeurs sur des variables	78
2.6.2	Pointeurs et tableaux	82
2.6.3	Allocation de mémoire	84
2.6.4	Arithmétique des pointeurs	87
2.6.5	Pointeurs de pointeurs	88

2.6.6	Pointeurs de fonctions	89
2.7	Références	97
2.8	Constantes	100
2.8.1	Constantes symboliques	100
2.8.2	Le type void.	101
2.8.3	Les alias de type : typedef	102
2.8.4	Constantes et énumérations	102
3.	Exceptions	103
3.1	Les approches de bas niveau.	103
3.1.1	Drapeaux et interruptions	103
3.1.2	Traitement des erreurs en langage C.	105
3.2	Les exceptions plus sûres que les erreurs.	107
3.3	Propagation explicite	108
3.4	Types d'exceptions personnalisés	109
3.4.1	Définition de classes d'exception.	109
3.4.2	Instanciation de classes	110
3.4.3	Classes d'exception dérivées.	111
3.5	Prise en charge d'une exception et relance	112
3.6	Exceptions non interceptées	113
3.7	Acquisition de ressources	113
4.	Travaux pratiques.	116
4.1	Prise en main de l'interprète Lab	116
4.1.1	Structure de la solution	116
4.1.2	Le dossier framework	117
4.1.3	Le dossier langage	118
4.1.4	Le dossier scriptboxes.	118
4.1.5	Utiliser l'interprète	119
4.2	Le code de la boucle principale.	120
4.3	Affichage dans Labshow.	122

Chapitre 2

De C à C++

1. Programmation structurée	125
1.1 Structures	126
1.1.1 Constitution d'une structure	127
1.1.2 Instanciation de structures	128
1.1.3 Instanciation avec l'opérateur new	129
1.1.4 Pointeurs et structures	130
1.1.5 Organisation de la programmation	131
1.2 Unions	132
1.3 Copie de structures	135
1.4 Création d'alias de types de structure	137
1.5 Structure et fonction	138
1.5.1 Passer une structure par valeur comme paramètre	138
1.5.2 Passer une structure par référence comme paramètre	139
1.5.3 Passer une structure par adresse comme paramètre	139
1.5.4 De la programmation fonctionnelle à la programmation objet	140
2. Gestion de la mémoire	141
2.1 Alignement des données	142
2.2 Allocation de mémoire interprocessus	143
3. La bibliothèque standard du C	143
3.1 Les fonctions communes du langage C <stdlib.h>	143
3.2 Chaînes <string.h>	145
3.3 Fichiers <stdio.h>	147
4. Travaux pratiques	152
4.1 Chargement de scripts dans Lab	152
4.2 Supprimer les erreurs liées à la librairie non sécurisée	154

Chapitre 3
Programmation orientée objet

- 1. Classes et instances 157
 - 1.1 Définition de classe..... 158
 - 1.1.1 Les modificateurs d'accès 159
 - 1.1.2 Organisation de la programmation des classes..... 162
 - 1.2 Instanciation..... 164
 - 1.3 Constructeur et destructeur 166
 - 1.3.1 Constructeur..... 166
 - 1.3.2 Le pointeur this..... 167
 - 1.3.3 Destructeur 168
 - 1.3.4 Destructeur virtuel..... 169
 - 1.4 Allocation dynamique 170
 - 1.5 Constructeur de copie 172
- 2. Héritage..... 173
 - 2.1 Dérivation de classe (héritage)..... 173
 - 2.1.1 Exemple de dérivation de classe..... 174
 - 2.1.2 Héritage public, protégé et privé 178
 - 2.1.3 Appel des constructeurs..... 179
 - 2.2 Polymorphisme 180
 - 2.2.1 Méthodes polymorphes 180
 - 2.2.2 Conversions d'objets..... 181
 - 2.3 Méthodes virtuelles et méthodes virtuelles pures 181
 - 2.4 Héritage multiple 186
 - 2.4.1 Notations particulières..... 187
 - 2.4.2 Conséquences sur la programmation 190
- 3. Autres aspects de la POO 192
 - 3.1 Conversion dynamique 192
 - 3.1.1 Conversions depuis un autre type..... 192
 - 3.1.2 Opérateurs de conversion 194
 - 3.1.3 Conversions entre classes dérivées 195

3.2	Champs et méthodes statiques	196
3.2.1	Champs statiques	196
3.2.2	Méthodes statiques	197
3.3	Surcharge d'opérateurs	202
3.3.1	Syntaxe	202
3.3.2	Surcharge de l'opérateur d'indexation	204
3.3.3	Surcharge de l'opérateur d'affectation	204
3.3.4	Surcharge de l'opérateur de conversion	205
3.4	Fonctions amies	205
3.5	Adressage relatif et pointeurs de membres	207
3.5.1	Notations	208
3.5.2	Construction d'un middleware orienté objet	209
3.6	Programmation générique	215
3.6.1	Modèles de fonctions	216
3.6.2	Modèles de classes	220
4.	Travaux pratiques	225
4.1	Utilisation de l'héritage de classes dans l'interprète Lab	225
4.2	Des pointeurs de membres pour des fonctions callback	227

Chapitre 4

La bibliothèque Standard Template Library

1.	Introduction	231
2.	Organisation des programmes	232
2.1	Espaces de noms	232
2.1.1	Utilisation complète d'un espace de noms	234
2.1.2	Espace de noms réparti sur plusieurs fichiers	235
2.1.3	Relation entre classe et espace de noms	236
2.1.4	Déclaration de sous-espaces de noms	238
2.2	Présentation de la STL	239

3.	Flux C++ (entrées-sorties)	239
3.1	Généralités	240
3.2	Flux intégrés	241
3.3	État d'un flux	241
3.4	Mise en forme	241
3.5	Flux de fichiers	243
3.6	Flux de chaînes	245
3.7	Paramètres locaux	246
4.	Classe string pour la représentation des chaînes de caractères	248
4.1	Représentation des chaînes dans la STL	249
4.2	Mode d'emploi de la classe string	250
4.2.1	Fonctions de base	250
4.2.2	Intégration dans le langage C++	252
4.2.3	Fonctions spécifiques aux chaînes	254
5.	Conteneurs dynamiques	256
5.1	Conteneurs	257
5.1.1	Insertion d'éléments et parcours	258
5.1.2	Itérateurs	258
5.1.3	Opérations applicables à un vecteur	259
5.2	Séquences	260
5.2.1	Conteneurs standards	260
5.2.2	Séquences	262
5.2.3	Adaptateurs de séquences	264
5.2.4	Conteneurs associatifs	268
5.3	Algorithmes	270
5.3.1	Opérations de séquence sans modification	270
5.3.2	Opérations de séquence avec modification	271
5.3.3	Séquences triées	272
5.3.4	Algorithmes de définition	273
5.3.5	Minimum et maximum	273

5.4	Calcul numérique	273
5.4.1	Limites des formats ordinaires.	274
5.4.2	Fonctions de la bibliothèque	275
5.4.3	Fonctions de la bibliothèque standard et classe valarray	276
6.	Travaux pratiques.	277
6.1	La classe Variant	277
6.2	La méthode to_string	280
6.3	Prise en charge des tableaux dans Lab	281

Chapitre 5

Les univers de C++

1.	L'environnement Windows	285
1.1	Les programmes Win32	285
1.2	Choix du mode de compilation	286
2.	L'environnement .NET	287
2.1	Le code managé et la machine virtuelle CLR	287
2.2	Les adaptations du langage C++ CLI	288
2.2.1	La norme CTS	288
2.2.2	La classe System::String	290
2.2.3	Le garbage collector	292
2.2.4	Construction et destruction d'objets.	294
2.2.5	La référence de suivi % et le handle ^	298
2.2.6	Le pointeur interne et les zones épinglées.	301
2.2.7	Les tableaux et les fonctions à nombre variable d'arguments.	303
2.2.8	Les propriétés	305
2.2.9	Les délégués et les événements	307
2.2.10	Les méthodes virtuelles	309
2.2.11	Les classes abstraites et les interfaces	311

- 2.3 Le framework .NET 312
 - 2.3.1 Les références d'assemblages 313
 - 2.3.2 L'espace de noms System::IO. 314
 - 2.3.3 L'espace de noms System::Xml 315
 - 2.3.4 L'espace de noms System::Data 316
 - 2.3.5 L'espace de noms System::Collections 317
 - 2.3.6 L'espace de noms System::Collections::Generic 318
 - 2.3.7 Le portage de la STL pour le C++ CLI 319
- 2.4 Les relations avec les autres langages : C# 319
- 3. Travaux pratiques. 321
 - 3.1 Une application en C++ pour .NET : le tableur InCell 321
 - 3.1.1 Architecture du tableur 321
 - 3.1.2 La feuille de calcul. 323
 - 3.1.3 L'ordonnanceur de calcul 331
 - 3.1.4 Zoom sur l'évaluateur. 336
 - 3.1.5 L'interface graphique 337
 - 3.2 Les pages web dynamiques EZ-Pages 338
 - 3.2.1 Fonctionnement des pages web dynamiques 338
 - 3.2.2 Implémentation de BufferBox 340
 - 3.2.3 Réalisation du questionnaire web EZHandler 340
 - 3.2.4 Tests 342
 - 3.2.5 Pour aller plus loin 343

Chapitre 6
Des programmes C++ efficaces

- 1. Dépasser ses programmes. 345
 - 1.1 Oublier les réflexes du langage C. 345
 - 1.2 Gestion de la mémoire 347
 - 1.3 Concevoir des classes avec soin 348
 - 1.4 Y voir plus clair parmi les possibilités de l'héritage 349
 - 1.5 Analyser l'exécution d'un programme C++ 350

2. La conception orientée objet (COO).....	351
2.1 Relation entre la POO et la COO	351
2.1.1 L'approche initiale de C++	351
2.1.2 UML et C++	352
2.2 Les design patterns	355
 Index	 357

Chapitre 5

Les univers de C++

1. L'environnement Windows

1.1 Les programmes Win32

La plate-forme Win32 a très vite été supportée par C++ car Microsoft a lancé dès 1992 l'un des premiers environnements intégrés pour ce langage ; il s'agissait de Visual C++, et à cette époque, de nombreux architectes étaient convaincus de l'intérêt d'un langage objet proche du langage C pour créer des applications fonctionnant dans des environnements graphiques.

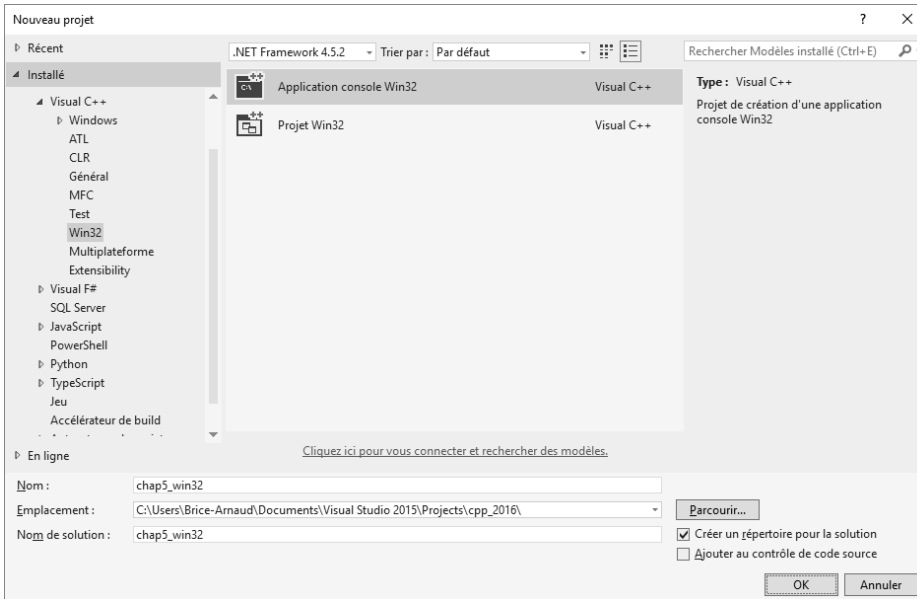
Toutefois, la programmation d'une application fenêtrée, sans framework, n'est pas une tâche aisée. Il n'en demeure pas moins que la plateforme Win32 propose plusieurs formats d'applications : applications "console", services Windows, bibliothèques dynamiques (DLL) et bien entendu les applications graphiques fenêtrées.

■ Remarque

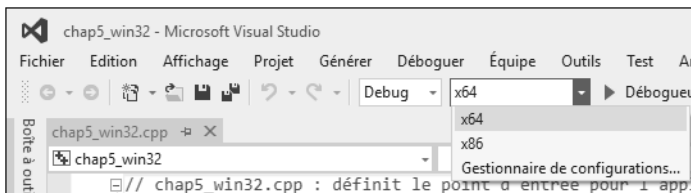
Il n'existe pas de modèle de projet "Win64" car Windows reste un système très ancré dans le mode 32 bits. Toutefois, Visual Studio sait compiler en mode 64 bits depuis un projet Win32.

1.2 Choix du mode de compilation

La création d'un projet s'effectue avec la commande **Fichier - Nouveau Projet** :

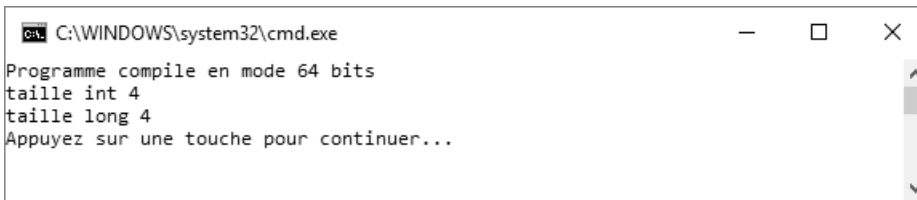


Depuis le gestionnaire de configuration, le mode de compilation 64 bits est activé :



Le programme qui suit est intéressant ; il indique que la taille des types `int` et `long` est identique à un mode 32 bits.

```
printf("Programme compilé en mode 64 bits\n");
printf("taille int %d\n", sizeof(int));
printf("taille long %d\n", sizeof(long));
```

A screenshot of a Windows command prompt window. The title bar shows the path 'C:\WINDOWS\system32\cmd.exe'. The window contains the following text:

```
Programme compile en mode 64 bits
taille int 4
taille long 4
Appuyez sur une touche pour continuer...
```

Le mode 64 bits recommande au compilateur de produire des instructions microprocesseur 64 bits, mais cela ne veut pas dire que les types sont ajustés en conséquence. Autrement dit, un calcul sur un type `__int64` pourra prendre plus de temps en 32 bits qu'en 64, mais `__int64` reste toujours 64 bits et `int` reste sur 4 octets quel que soit le mode de compilation. Il s'agit là d'une particularité du compilateur de Microsoft ; d'autres compilateurs peuvent adopter un comportement différent.

2. L'environnement .NET

2.1 Le code managé et la machine virtuelle CLR

Comme Java, .NET fait partie de la famille des environnements virtualisés. Le compilateur C++ managé ne produit pas du code assembleur directement exécuté par le microprocesseur mais un code intermédiaire, lui-même exécuté par une machine "virtuelle", la CLR (*Common Language Runtime*). Cette couche logicielle reproduit tous les composants d'un ordinateur – mémoire, processeur, entrées-sorties... Cette machine doit s'adapter au système d'exploitation qui l'héberge, et surtout optimiser l'exécution du code.

■ Remarque

Il y a là une différence importante entre .NET et Java. Alors que Microsoft a évidemment fait le choix de se concentrer sur la plate-forme Windows, d'autres éditeurs ont porté Java sur leurs OS respectifs – Unix, AIX, Mac OS, Linux... La société Novell a cependant réussi le portage de .NET sur Linux, c'est le projet Mono.

Cet environnement virtualisé a une conséquence très importante sur les langages supportés ; les types de données et les mécanismes objets sont ceux de la CLR. Comme Microsoft était dans une démarche d'unification de ses environnements, plusieurs langages se sont retrouvés éligibles à la CLR, quitte à adapter un peu leur définition. Le projet de Sun était plutôt une création ex nihilo, et de ce fait seul le langage Java a réellement été promu sur la machine virtuelle JVM.

2.2 Les adaptations du langage C++ CLI

Le terme CLI signifie *Common Language Infrastructure* ; c'est l'ensemble des adaptations nécessaires au fonctionnement de C++ pour la plate-forme .NET / CLR.

2.2.1 La norme CTS

Le premier changement concerne les types de données. En successeur de C, le C++ standard a basé sa typologie sur le mot machine (`int`) et le caractère de 8 bits (`char`). Confronté à des problèmes de standardisation et d'interopérabilité, Microsoft a choisi d'unifier les types de données entre ses différents langages.

Les types primitifs, destinés à être employés comme variables locales (boucles, algorithmes...), sont des types "valeurs". Leur zone naturelle de stockage est la pile (*stack*), et leur définition appartient à l'espace de noms **System** :

<code>wchar_t</code>	<code>System::Char</code>	Caractère unicode
<code>signed char</code>	<code>System::SByte</code>	Octet signé
<code>unsigned char</code>	<code>System::Byte</code>	Octet non signé
<code>double</code>	<code>System::Double</code>	Décimal double précision
<code>float</code>	<code>System::Float</code>	Décimal simple précision
<code>int</code>	<code>System::Int32</code>	Entier 32 bits
<code>long</code>	<code>System::Int64</code>	Entier 64 bits
<code>unsigned int</code>	<code>System::UInt32</code>	Entier 32 bits non signé

unsigned long	System::UInt64	Entier 64 bits non signé
short	System::Int16	Entier court 16 bits
bool	System::Boolean	Booléen
void	System::Void	Procédure

Voici un exemple utilisant à la fois la syntaxe habituelle de C++ et la version "longue" pour définir des nombres entiers. Cette dernière ne sera utilisée qu'en cas d'ambiguïté, mais il s'agit en fait de la même chose.

```
int entier = 4; // alias de System::Int32
System::Int32 nombre = entier;
```

Les structures managées (**ref struct**) sont composées de champs reprenant les types ci-dessus. Elles sont créées sur la pile et ne sont pas héritables.

```
ref struct Point
{
public:
    int x,y;
};

int main(array<System::String ^> ^args)
{
    Point p; // initialisation automatique sur la pile : pas de gnew
    p.x = 2;
    p.y = 4;

    return 0;
}
```

Comme tous les types valeurs, les structures managées n'autorisent pas l'effet de bord à moins qu'une référence explicite n'ait été passée (voir ci-dessous les références suivies).

Par opposition les classes managées (**ref class**) sont créées sur le tas et correspondent davantage au fonctionnement des classes (structure) du C/C++ standard. Évidemment, elles sont héritables, manipulées par références, et instanciées par l'opérateur **gnew**.

```
ref class Personne
{
public:
```



```
        String^ nom;
        int age;
    } ;

int main(array<System::String ^> ^args)
{
    Personne^ pers = gnew Personne();
    pers->nom = "Marc";
    pers->age = 35;

    return 0;
}
```

Naturellement, les structures et les classes gérées supportent la définition de méthodes, C++ est bien un langage objet !

Les langages .NET sont fortement typés, en évitant autant que possible la généralité "fourre-tout" du `void*` issu du C, ou du `Variant` de VB. En opposition, la norme CTS prévoit que l'ensemble des types de données (valeurs ou référence) héritent d'un comportement commun `System::Object`. Nous découvrirons un peu plus loin comment cette caractéristique est exploitée dans les classes de collections d'objets.

2.2.2 La classe `System::String`

Parmi les types intrinsèques à .NET on trouve la classe `System::String` pour représenter les chaînes. Elle n'est pas aussi intégrée au langage C++ CLI que dans C#, mais cependant elle offre exactement les mêmes services. Microsoft l'a de plus aménagée pour faciliter la manipulation et la conversion avec `char*` :

Voici un premier exemple de construction de chaînes. On remarquera l'emploi facultatif du symbole `L` devant les littérales de chaîne, ainsi que le symbole `^` dont la signification sera expliquée un peu plus tard.

```
char* c = "Bonjour C++";
String ^ chaine1 = gnew String(c); // construction à partir d'un char*
String ^ chaine2 = L"C++ CLI c'est formidable"; // littérale de chaîne .NET
String ^ chaine3 = "Un univers à découvrir"; // idem

Console::WriteLine(chaine1);
Console::WriteLine(chaine2 + ". " + chaine3);
```