Ressourcesinformatiques

C++

Les fondamentaux du langage

2e édition

Brice Arnaud GUÉRIN

Fichiers complémentaires à télécharger

eni Editions

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

Chapitre 1
Introduction

Chapitre 2
De C à C++

Chapitre 3
Programmation orientée objet

Chapitre 4
La bibliothèque Standard Template Library

Chapitre 5
Les univers de C++

Chapitre 6
Des programmes C++ efficaces

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.

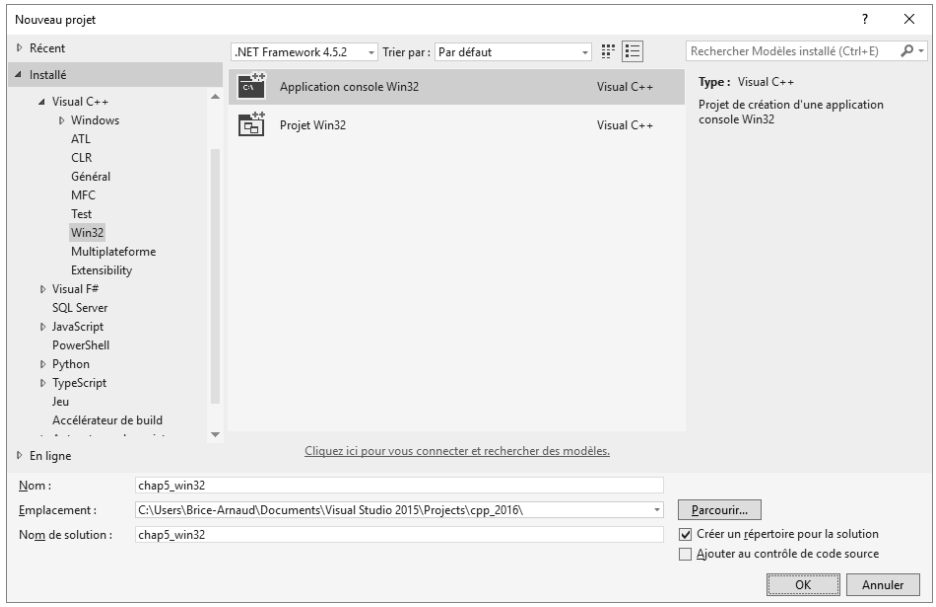
Toutefois, la programmation d'une application fenêtrée, sans framework, n'est pas une tâche aisée. Il n'en demeure pas moins que la plateforme Win32 propose plusieurs formats d'applications : applications "console", services Windows, bibliothèques dynamiques (DLL) et bien entendu les applications graphiques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

Remarque

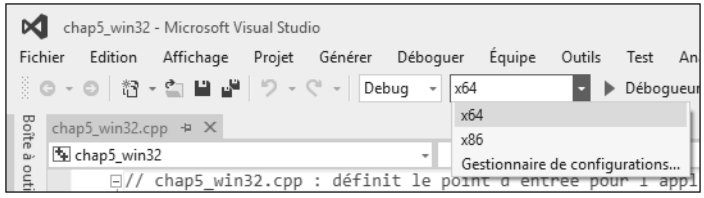
Il n'existe pas de modèle de projet "Win64" car Windows reste un système très ancré dans le mode 32 bits. Toutefois, Visual Studio sait compiler en mode 64 bits depuis un projet Win32.

1.2 Choix du mode de compilation

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

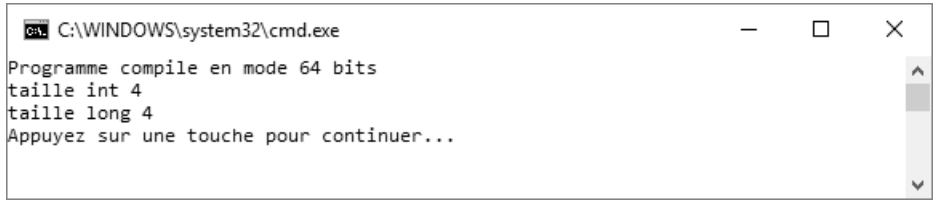


Depuis le gestionnaire de configuration, le mode de compilation 64 bits est activé :



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));
```



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`** :

`wchar_t`	`System::Char`	Caractère unicode
`signed char`	`System::SByte`	Octet signé
`unsigned char`	`System::Byte`	Octet non signé
`double`	`System::Double`	Décimal double précision
`float`	`System::Float`	Décimal simple précision
`int`	`System::Int32`	Entier 32 bits
`long`	`System::Int64`	Entier 64 bits
`unsigned int`	`System::UInt32`	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 gcnew
    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 **`gcnew`**.

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

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

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

    return 0;
}
```

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

Les langages .NET sont fortement typés, en évitant autant que possible la généricité "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 = gcnew 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);
```