Bien débuter avec Visual C++

Le développement en C++ est plus confortable dans un IDE qu'avec une simple ligne de commande. L'époque ou le développeur était en face à face avec son interpréteur de commandes comme cmd.exe est révolue ou presque.

De nos jours, les environnements de développement intégrés partagent toutes les fonctionnalités requises pour développer confortablement, avec le langage de votre choix. En plus de l'environnement de développement intégré (IDE), le package inclut les différentes entêtes et librairies pour développer sous Windows alias Windows SDK, anciennement nommé Platform SDK. Les IDE existent depuis les années 90.

Exemple d'un IDE Visual Developer Studio 4.2 de 1995 dans lequel on retrouve la liste des classes (et des fichiers) sur la partie gauche de l'écran et il y a des options de recherche dans le texte, les fameux boutons pour compiler l'application et la documentation intégrée.

Voici l'IDE de Visual Studio 2010 Express et plus précisément Visual C++ Express, disponible sur http://www.microsoft.com/vstudio/express. On voit que malgré les années, les IDE gardent le même aspect. Avec Visual Studio 2010, on peut créer différents types de projets.

Pour créer sa première application de VS2010 Express, comment s'y prendre ? Il suffit de faire File -> New -> Project et de choisir un type d'application. Dans notre cas, on va sélectionner application console Win32.

Le wizard propose plusieurs options que l'on peut laisser par défaut. Pour compiler le programme, il suffit de « builder » l'application en utilisant soit le menu Debug -> Build Solution, ou bien en faisant clic droit sur le projet dans le Solution Explorer et de faire Build. La dernière option consiste à utiliser la toolbar et d'appuyer sur l'icône Build. Pour pouvoir builder l'application, l'IDE a besoin d'un compilateur, d'un éditeur de lien et aussi de fichiers d'entêtes que l'on va trouver dans le Windows SDK. Visual Studio distribue tout cela.

> Windows SDK

Le Windows SDK fournit les outils, les compilateurs, les entêtes et les librairies nécessaires à la création d'application sur les plateformes Windows. Il existe aujourd'hui deux versions majeures de kits de développement « Windows SDK » : Windows SDK pour Windows 7 et NET Framework 3.5 SP1 ou Windows SDK pour Windows 7 et NET Framework 4.0. Les deux versions de Windows SDK sont disponibles en téléchargement gratuit. La première version redistribue le compilateur disponible dans Visual C++ 2008 SP1 (VC9) et la deuxième distribue le compilateur de Visual C++ 2010 SP1 (VC10).

Pour lancer une compilation en mode de commande, il faut positionner quelques variables d'environnement comme PATH, INCLUDE et LIB. La définition exacte de l'environnement est décrite dans le fichier vcvars32.bat disponible sous %ProgramFiles%\Microsoft Visual Studio 10.0\VC\bin. Le kit Windows SDK fournit aussi un fichier qui permet de créer son environnement de build en utilisant SetEnv.cmd disponible sous %ProgramFiles%\Microsoft SDKs\Windows\v7.1\Bin et dont l'utilisation est la suivante :

```
Usage : Setenv [/Debug | /Release][/x86 | /x64 | /ia64 ]
[/vista | /xp | /2003 | /2008 | /win7][-h | /?]
```

Bien qu'il soit possible de compiler directement depuis la ligne de commande, l'utilisation d'un IDE apporte un plus grand confort de travail. Le plus simple est d'utiliser Visual C++ 2010 Express qui est gratuit et qui fournit le même environnement que Visual Studio 2010. VC++ Express distribue la runtime C, la librairie standard C++ (STL) et d'autres librairies comme PPL et les fichiers du Windows SDK.

Important : Il faut installer Visual Studio 2010 SP1 après l'installation de Visual C++ 2010 Express. La version Express possède quelques limitations comme l'absence des librairies MFC et ATL, pas de compilation 64 bit, pas de gestionnaire de fichiers de ressources (RC editor) pour désigner les interfaces graphiques, pas de support pour OpenMP ni pour les add-ins Visual Studio. Cependant, ces restrictions ne sont pas trop importantes pour quelqu'un qui veut démarrer dans le développement Visual C++. L'utilisation d'une bonne documentation est fondamentale. Il est possible de télécharger la documentation ou de l'utiliser en ligne. Je conseille, pour ceux qui débutent, une autre option qui est la documentation "MSDN Library pour Visual Studio 2008 SP1". Disponible gratuitement, cette image ISO de 2.2 GB qui contient la fameuse MSDN Library qui est la véritable bible pour les programmeurs Windows. Pour bien démarrer, il suffit parfois de trouver un programme existant et de l'adapter. Les programmes d'exemples sont disponibles gratuitement.

> Introduction à C++

Le langage C++ standard ISO/IEC (C++11) vient d'être publié par Herb Sutter et son équipe au mois de mars 2011 après de nombreuses années d'effort sur la standardisation. C'est le langage C++ standard, celui qui est disponible sur les environnements Windows, Linux et que l'on utilise avec un compilateur comme GCC ou Visual C++. C++ est un langage type-safe, qui fournit des mécanismes d'abstractions évolués (classe, template) et qui ne sacrifie par le pouvoir et les performances. C++ a toujours été pour Microsoft comme l'électricité, quelque chose de naturel à utiliser. Les équipes Windows, Office, SQL Server, SharePoint sont autant d'équipes qui utilisent C++. A ce titre, je vous renvoie sur les derniers webcasts Channel9 là où les vidéos sur « C++ Renaissance », « Going Native » et « Advanced STL » sont populaires et cela traduit le retour à la programmation native.

> Commencer par le C

Historiquement C a été construit pour faire Unix. Puis, C a été enrichi pour devenir « C avec classes » et s'est progressivement appelé C++ au fur et à mesure de l'effort de standardisation. Au début, on apprend les types de base et les rudiments du langage: void, signed/unsigned char, int, long, float, double. Ensuite, on apprend les tableaux, les pointeurs, les énumérations, les structures et les unions et les divers éléments d'expression comme les opérateurs, les opérations de conversion de types puis les séquences de code comme if, else, else-if, do-while, while, for, switch, break, goto, puis viennent ensuite la programmation des fonctions et le point d'entrée main(). Ensuite on utilise la Runtime C appelée CRT et elle va nous

servir de boîte à outils pour développer. En 1998, j'achetais pour 350 F, un livre qui se nommait « L'essentiel du C++, 2e édition » de Stanley B. Lippman. Ce livre est toujours d'actualité en 2012. Je vais suivre la trame et les chapitres du livre pour vous expliquer l'essentiel de l'essentiel du C++.

> Mon premier programme

Le premier programme qui doit être écrit (sans faute) et qui donne du cœur à l'ouvrage. Créer un fichier nommé totor.cpp et écrire cela dedans :

```
#include <stdio.h>

int _tmain(int argc, _TCHAR* argv[])
{
        printf(«Hello Programmez\n»);
        return 0;
}
```

Ensuite, essayer de compiler ce programme. La commande va se traduire par <nom du compilateur> <nom du fichier>.cpp ou bien « Build » dans l'IDE. Cela devrait marcher. Apprendre un langage, c'est savoir comment déclarer des variables. Une variable possède toujours un type que nous détaillons ici. Il existe plusieurs types en C++ : short, int, char, double, float. La variante, un peu pénible pour ceux qui débutent, est de pouvoir ajouter le petit mot (mot clé ou keyword) unsigned ou signed. Voici l'essentiel :

```
20
024
0x14            // hexa
128u            // unsigned
1024UL  // unsigned long
1L              // long
8Lu             // long unsigned
3.14159F// float
0.1f            // float
12.345L // long float haute précision
0.0             // double
3E1             // exposant
1.0E-3
2.
1.0L
'a'
'2'
','
' '             // blank
""              // null string
"aa"            // chaine aa
"A \n B \n"     // A B avec un saut de ligne (\n alias newline)
entre chaque lettre
```

Maintenant qu'on possède des types, il faut créer des variables. Il suffit juste de nommer la variable en y insérant avant le type :

```
        int i;
        float j;
        double k;
        char c;
```

Par contre, il serait plus intelligent de nommer des variables avec un vrai nom appelé identificateur. Il existe des mots réservés en C++ et il est interdit de nommer une variable comme un mot réservé.

> Les pointeurs

Les pointeurs sont des types particuliers qu'il est possible de noter ainsi. Un pointeur est une adresse mémoire. C'est à partir de cet emplacement que l'on trouve des données. Exemple de pointeur qui contient un int :

```
int i = 10;
int * pointeurSurI = &i ;    // je pointe sur i ;
i=11 ;          // je modifie i
*pointeurSurI = 12 ;     // je modifie i car je pointe sur i
```

Les chaînes de caractères sont des ensembles de caractères qu'il est possible de déclarer explicitement en indiquant une taille ou non. Les chaînes se terminent avec un zéro terminal :

```
char sz[255] = «Edith» ;
char *sz2 = «Lisa» ;
char sz3[] = «Maggie» ;
```

Dans notre exemple, sz2 est un pointeur et sz3 est un tableau. QuickWatch différencie bien les types différents, un char* et char[7]. Les chaînes sont terminées par un zéro.
Une constante est déclarée avec le mot const devant :

```
int a = 10;
const int b = 20;
a++;// incrémente a
b++; // b ne peut pas être modifié -> error C2105: '++' needs l-value
```

Le compilateur C++ n'est pas très cool lorsqu'il s'agit de dire ce qu'il se passe en cas de problème. L'erreur veut dire, ++, c'est-à-dire l'incrémentation ne peut se faire que sur un left value, une valeur de gauche. Je vous laisse chercher sur bing ! (ou sur Google). Un tableau se définit avec des crochets. Les éléments commencent à la zero-ième place :

```
int tab[20];
tab[0] = 25;  // le premier élément vaut 25
tab[10] = 100;// le 11eme élément vaut 100
float tab2[10][10];
tab2[0][5] = 0.5f;
tab2[2][7] = 2.7f;
```

> La Runtime C

Le langage C dispose d'une librairie nommée Runtime C qui contient les routines de base pour réaliser des applications. Cette runtime est implémentée en langage C et fournit les fonctions communément définies dans le C ANSI, qui fait l'objet du livre K&R « C ANSI ». Avec Visual C++, le code source de la runtime est fourni dans le répertoire %ProgramFiles%\Microsoft Visual Studio 10.0\VC\crt\ src. Dans Visual C++ 2010, la CRT contient 1300 fichiers pour 15 MB de source code. L'avantage d'apprendre à utiliser la runtime C permet de comprendre le bonheur induit dans l'utilisation de C++ et de sa librairie standard ou STL. Cette librairie propose des types comme string, vector, map qui sont élégants à manipuler. Il n'y a plus à allouer de la mémoire, à indiquer de longueur de chaînes… Tout devient simple et facile à utiliser.

> Les classes

La notion de classe est l'essence même du C++. Techniquement parlant, une classe est une structure qui contient des membres avec une certaine visibilité : public, private ou protected. En C++, une classe est un type de données qu'il est possible de créer. Votre classe contient ce que vous y mettez. Il n'y a aucun interdit, aucune restriction. Une classe est-elle une construction orientée objet ? La réponse est : ça dépend de vous car la notion de classe technique (une structure) et les concepts de l'orienté objet (O.O.) sont deux choses différentes. Il est possible de faire de l'orienté objet avec du C et il est possible de faire du procédural avec du C++. Nous allons voir comment faire de l'orienté objet avec du C++ de manière simple et élégante. On va partir très simplement pour que vous ne soyez perdu au premier type venu. Les classes sont des types particuliers. Une classe est un conteneur d'éléments : des membres (variables), des fonctions (membres nommés méthodes), des opérateurs et aussi des énumérations ou des structures. Et puis il y a un constructeur (ctor) et un destructeur (dtor). Bref, une classe c'est un cadre dans lequel on définit des éléments cohérents. Pour comprendre les classes, il faut se souvenir que C++ a été créé parce que C ne contenait que les structures. C++ introduit les classes qui sont des structures avec des fonctions. Une classe est une structure qui contient des fonctions. On introduit des classes dans un programme pour créer un nouveau type. Une classe peut hériter d'une classe existante.

```
class CRenault
{
public:
 string brand;

public:
 CRenault()
 {
   brand = «quality made»;
 }
};

class CLaguna : public CRenault
{
public:
 string immatriculation;
 string owner;

public:
 CLaguna()
 {
   immatriculation = «BF-008-CZ»;
   owner = «Christophe Pichaud»;
 }
};
```

What ? Talking to me ???? kesako. Comment lire ce programme? La classe CLaguna hérite de la classe CRenault. Un objet CLaguna est un objet CRenault avec des éléments en plus. Un objet CRenault possède un attribut « brand » mais un objet CLaguna possède les attributs immatriculation et propriétaire. Dans mon exemple, j'y ai fait figurer dans le constructeur des valeurs par défaut. Rien ne m'y oblige. Une classe est un ensemble de membres qui ont une visibilité (public, private ou protected).

> Un autre exemple de classes

Prenons un exemple du monde réel. Un programme de dessin avec des formes plus ou moins simples. Les classes vont être les formes. CCarre, CRond, CPolygone, etc. Voici à quoi cela peut ressembler : on va définir les classes dans les points .h (headers ou en-têtes) et puis le code dans les points .cpp (body ou corps).
On va définir les types de formes dans une énumération :

```
enum Shape
{
 carre,
 rond,
 ellipse,
 image
};
```

L'énumération permet de faire des constantes « propres » et claires à utiliser avec `Shape::carre` ou `Shape::rond`. Maintenant, il faut définir la classe de forme (Shape en anglais). On va faire simple, une classe ne contiendra que la méthode pour dessiner une forme.

```
class CShape
{
public:
 CShape(void);
 virtual ~CShape(void);

 virtual void Draw(int x, int y);
};

class CShapeCarre : public CShape
{
public:
 CShapeCarre(void);
 virtual ~CShapeCarre(void);

 virtual void Draw(int x, int y);
};

class CShapeRond : public CShape
{
public:
 CShapeRond(void);
 virtual ~CShapeRond(void);

 virtual void Draw(int x, int y);
};
```

Et maintenant le code d'implémentation des méthodes. On fait simple, la fonction virtuelle (que l'on peut redéfinir) ne fait qu'afficher le texte du dessin, à savoir le nom de la forme.

```
void CShape::Draw(int x, int y)
{
 cout << «CShape::Draw x:» << x << « y:» << y << endl;
}

void CShapeCarre::Draw(int x, int y)
{
 cout << «CShapeCarre::Draw x:» << x << « y:» << y << endl;
```

```
}

void CShapeRond::Draw(int x, int y)
{
 cout << «CShapeRond::Draw x:» << x << « y:» << y << endl;
}
```

J'ai volontairement oublié les constructeurs et destructeurs (qui sont vides). Il n'y a rien dedans. Vous trouverez bien le moyen d'y mettre quelque chose, nom par exemple, la taille de l'objet par défaut. Et maintenant, essayons de créer des objets au travers du programme main :

```
#include «stdafx.h»
#include «ClassFactory.h»
#include «Shape.h»

int _tmain(int argc, _TCHAR* argv[])
{
 shared_ptr<CShape> pShape1 = CClassFactory::CreateShape(Shape:
:carre);
 pShape1->Draw(15, 20);
 shared_ptr<CShape> pShape2 = CClassFactory::CreateShape(Shape:
:rond);
 pShape2->Draw(50, 50);
 return 0;
}
```

Je suis en C++ 11, donc plus besoin de faire des new/delete, on passe par des shared_ptr et cela donne cela au travers d'une classe factory :

```
class CClassFactory
{
public:
 CClassFactory(void);
 virtual ~CClassFactory(void);

 static shared_ptr<CShape> CreateShape(Shape type);
};

shared_ptr<CShape> CClassFactory::CreateShape(Shape type)
{
 shared_ptr<CShape> pNewElement = nullptr;
 switch( type )
 {
 case carre:
   pNewElement = make_shared<CShapeCarre>();
   break;
 case rond:
   pNewElement = make_shared<CShapeRond>();
   break;
 case ellipse:
   pNewElement = make_shared<CShapeEllipse>();
   break;
 case image:
   pNewElement = make_shared<CShapeImage>();
   break;
 }
 return pNewElement;
}
```

Avec le make_shared, je crée un objet shared_ptr qui saura se libérer tout seul malgré le passage en retour de fonction. Le résultat du programme est le suivant :

```
CShapeCarre::Draw x:15 y:20
CShapeRond::Draw x:50 y:50
```

En effet, dans le main, on crée un carré puis un rond. Les deux objets sont dessinés via l'appel de méthode Draw(). Et vous remarquerez qu'il n'est jamais nécessaire de détruire les objets créés.

> La librairie standard C++ (STL)

La libraire STL est constituée de classes et de templates définis en 6 parties : containers, container adapters, iterators, algorithms, function objects et function adapters. Les éléments de la famille « containers » sont équivalents aux collections. Ces classes permettent de stocker, supprimer ou d'accéder à certains éléments. Voici la liste des containers :

Container	Fichier d'entêtes	Description
vector	<vector>	vector<T> est un tableau qui grossit automatquement. C'est le container le plus utilisé
deque	<deque>	deque<T> est une double-ended queue. C'est comme un vector mais plus rapide pour l'insertion en début ou en fin
list	<list>	list<T> permet l'ajout et la suppression rapide d'éléments
map	<map>	map<K,T> stoque des couples key/types uniques
multimap	<map>	multimap<K,T> permet que les couples soient identiques
set	<set>	set<T> est un ensemble d'éléments uniques
multiset	<set>	multiset<T> permet que les éléments soient identiques
hash_map	<hash_map>	hash_map<K, T> est un map rapide en accès
hash_multimap	<hash_map>	hash_multimap<K, T> est un multimap rapide en accès
hash_set	<hash_set>	hash_set<T> est un set rapide en accès
hash_multiset	<hash_set>	hash_multiset<T> est un multiset rapide en accès
stack	<stack>	stack<T> implémente LIFO (last in, first out) donc dernier entré, premier sorti
queue	<queue>	queue<T> implémente FIFO(first in, first out) donc premier entré, premier sorti
priority_queue	<queue>	priority_queue<T> est queue dans laquelle l'élement le plus haut est le premier dans la queue

Les iterators ou itérateurs en français permettent l'accès aux containers. Ils fonctionnement comme des pointeurs et peuvent être de différents styles : input, output, forward, bidirectional, random access. Chaque container utilise un type d'itérateur spécifique dont la notation est de type prefixe type puis suffixe type d'iterator. La documentation associée à chaque classe indique lequel utiliser. Un exemple illustre comment utiliser les containers et les itérateurs.

```
#include <vector>
#include <string>
#include <iostream>
using namespace std;
…
void routine2()
{
 vector<string> vector1;
 // Ajout d'éléments
 vector1.push_back(«C++ Renaissance»);
 vector1.push_back(«Going Native»);
 vector1.push_back(«Why C++ ?»);
 // Obtention de l'itérateur -> vector<string>::iterator it =
vector1.begin()
 // syntaxe C++ 11 -> auto it = vector1.begin();
 for(vector<string>::iterator it = vector1.begin() ; it!=vector1
.end() ; it++)
 {
   cout << «value: « << *it << endl;
```

```
 }
 // Itération dans une boucle for classique
 for(size_t i=0 ; i<vector1.size() ; i++ )
 {
   cout << «value: « << vector1[i] << endl;
 }
}
```

Pour ajouter des éléments dans un container vector<T>, on utilise la méthode push_back(T). Pour itérer dans le container, on obtient l'itérateur propre au container puis on itère en utilisant l'opérateur ++. L'accès à l'élément se fait en utilisant l'opérateur *.
Une itération peut aussi se faire via l'utilisation du vector<T> comme un tableau en utilisant l'opérateur []. La STL fournit plusieurs moyens pour réaliser les opérations et ainsi permet plusieurs styles d'écriture de code.
Il existe un container très sympa à utiliser qui permet d'associer une clé avec une valeur : map<K,T>. L'ajout d'éléments se fait soit en utilisant l'opérateur [] ou bien la méthode insert qui prend un type pair<first, second>.
L'itération dans le container se fait en utilisant les membres first et second. La recherche se fait en utilisant la méthode count(). Exemple de code :

```
#include <map>
…
void routine3()
{
 map<string, string> dico;
 dico[«VB»] = «Visual Basic»;
 dico[«C#»] = «Visual C# -> Productivity»;
 dico[«C++»] = «Visual C++ -> Power and Performance !»;
 // Ajout en utilisant la fonction make_pair
 dico.insert(make_pair(«C++11», «Fast and Fluid ! See you later
 in Windows 8»));
 // Obtention de l'itérateur via container<T>::iterator
 for( map<string, string>::iterator it = dico.begin() ; it!=
dico.end() ; it++)
 {
   cout << «key: « << it->first << « value:» << it->second << endl;
 }
 // count détermine si une clé existe -> retourne 0 ou 1
 if(dico.count(«C++») != 0)
 {
   cout << «C++ value: « << dico[«C++»] << endl;
 }
}
```

Les containers de la STL sont génériques. Ils fonctionnent sur des types que nous pouvons créer, exemple avec le type LangageProp qui contient deux strings.

```
class LangageProp
{
private:
 string desc, core_value;
public:
 LangageProp(string d, string cv ) : desc(d), core_value(cv) {}
 string Display() { return desc + « is « + core_value; }
};
```

La STL fournit aussi un ensemble de fonctions templates appelées algorithmes, disponibles dans le fichier d'entêtes <algorithm>. Ce fichier contient plusieurs dizaines de fonctions génériques qui savent traverser les containers et y opérer un certain traitement. Voici une liste non exhaustive des fonctions :

Fichier d'entêtes	Description
<algorithm>	binary_search, copy, count, count_if, equal, equal_range, fill, find, find_if, for_each, generate, lower_bound, max, merge, mismatch, partial_sort, remove, remove_if, replace, replace_if, reverse, rotate, search, sort, swap, transform ...

La fonction la plus connue est for_each qui permet de faire un même traitement pour l'ensemble des éléments d'un container. Nous reprenons comme base la collection ci-dessus. L'appel à for_each se fait sur la base d'un itérateur de début, d'un itérateur de fin et d'une routine, OnItem, qui va être appelée sur chaque élément du container.

```
void OnItem(pair<string, LangageProp> value)
{
 // retrieve the LangageProp object reference
 LangageProp &prop = value.second;
 cout << «OnItem...» << prop.Display() << endl;
}

void routine5()
{
 map<string, LangageProp> dico;
 dico.insert(make_pair(«VB», LangageProp(«Visual Basic», «for the
 kids ?»)));
 dico.insert(make_pair(«C#», LangageProp(«Visual C#», «for
productivity»)));
 dico.insert(make_pair(«C++», LangageProp(«Visual C++», «for Power
and Performance !»)));
 for_each(dico.begin(), dico.end(), OnItem);
}
```

> Programmation moderne avec C++11

Les classes et templates disponibles dans le nouveau C++ standardisé qu'est C++ 11 sont fantastiques et en grande quantité. Tous les algorithmes, par exemple, rendent de grands services. Écrire du C++ moderne ça veut dire quoi ?
Par exemple, on ne va plus explicitement faire de simples new/delete sur nos objets. On va construire et manipuler les objets avec des « smart pointers » via shared_ptr<T>. Ce template implémente un mécanisme de compteur de références qui permet de partager des objets entre plusieurs clients, et de stocker des pointeurs dans des containers en toute sécurité. L'abstraction weak_ptr<T> est le meilleur compagnon de shared_ptr<T>. Cela permet de casser les références circulaires. Le template weak_ptr<T> implémente le pattern Observer pour les shared_ptr<T>. Avec un weak_ptr<T>, dès qu'un shared_ptr<T> libère ses ressources, l'information est propagée à tous les weak_ptr<T> afin qu'il ne puisse plus utiliser un pointeur invalide. C++11 fera l'objet d'articles dans le futur !

Christophe **Pichaud** / *.NET Rangers by Sogeti*
Consultant sur les technologies Microsoft
christophepichaud@hotmail.com - www.windowscpp.net