LABORATOIRE 0

INTRODUCTION À L'ENVIRONNEMENT DE DÉVELOPPEMENT

VISUAL STUDIO 2013

Table des matières

Section 1 – L'environnement de développement

Dans ce laboratoire, nous apprendrons à utiliser l'environnement de développement ***Visual Studio 2013*** de ***Microsoft®*** (nommé ci-après VS).

Pourquoi un environnement de développement?

Notre outil de travail dans ce cours, VS, est ce qu'on appelle un ***environnement de développement intégré*** (en anglais : "*Integrated Development Environment*", ou **IDE**).

Le but d'un tel outil est, entre autres[1], de permettre :

- l'édition du ***programme*** (on dira souvent du ***code***), qui est le fruit de la traduction d'un algorithme du pseudocode en langage de programmation;
- sa ***compilation***, qui est sa traduction du langage de programmation à un langage qui sera compris directement par l'ordinateur—le langage machine;
- l'***édition des liens*** (une étape qui nous sera un peu transparente dans ce cours); et
- l'***exécution*** du programme.

L'exercice d'aujourd'hui nous amènera à traverser toutes ces étapes dans le but de nous familiariser avec notre IDE en tant que tel, de même qu'avec les principales étapes menant de la traduction d'un algorithme en programme à son exécution et à sa mise au point.

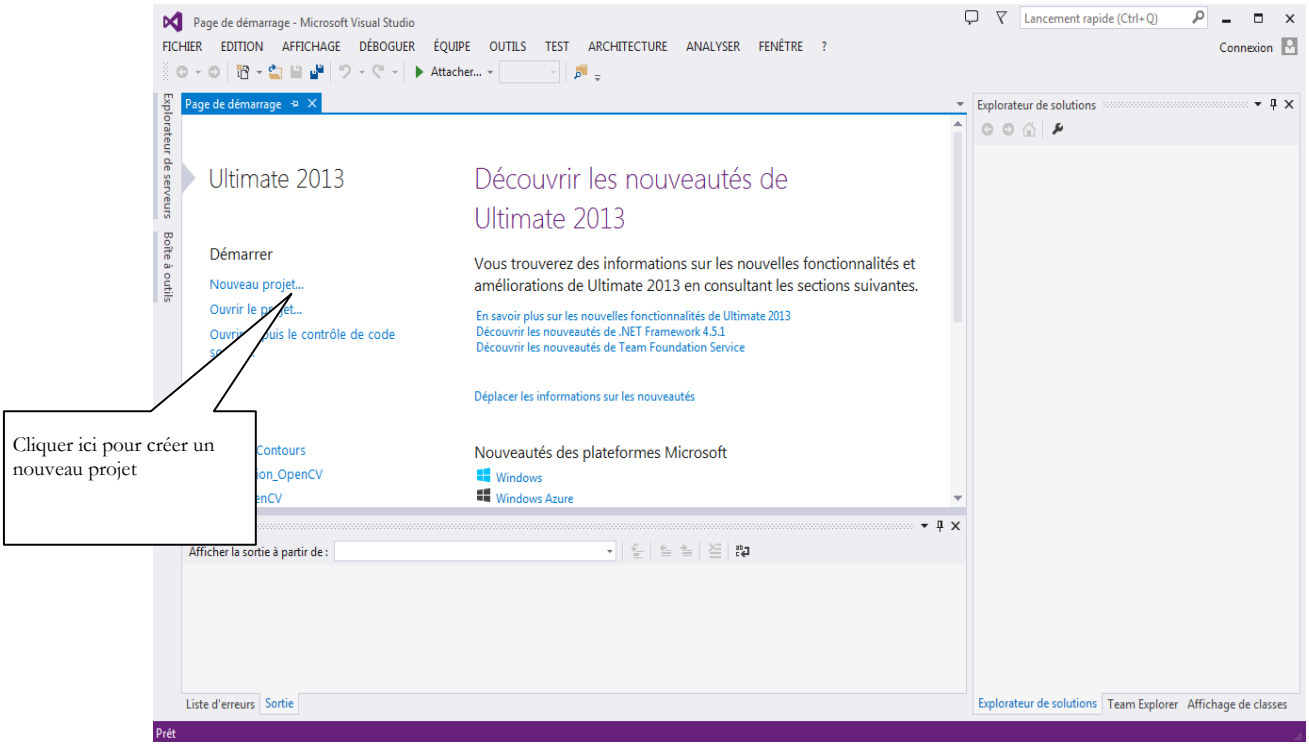
[1] Un `IDE` moderne comme celui de `VS` est un outil extrêmement puissant et polyvalent; nous n'en exploiterons, dans ce cours, que quelques-unes des possibilités.

Démarrer VS

Pour démarrer VS, sélectionnez

`Démarrer➔ Tous les programmes➔ Visual Studio 2013➔ Visual Studio 2013`

Vous devriez obtenir un écran ressemblant à ceci :



Note : La page de démarrage proposée devient rapidement agaçante. Vous pouvez vous en débarrasser aisément par les menus comme montré ci-dessous

```
Outils➔
   Options...➔
      Environnement➔
             Démarrage➔
                    Dans la boîte « Au démarrage: », sélectionnez : « Afficher
                     l'environnement vide »
```

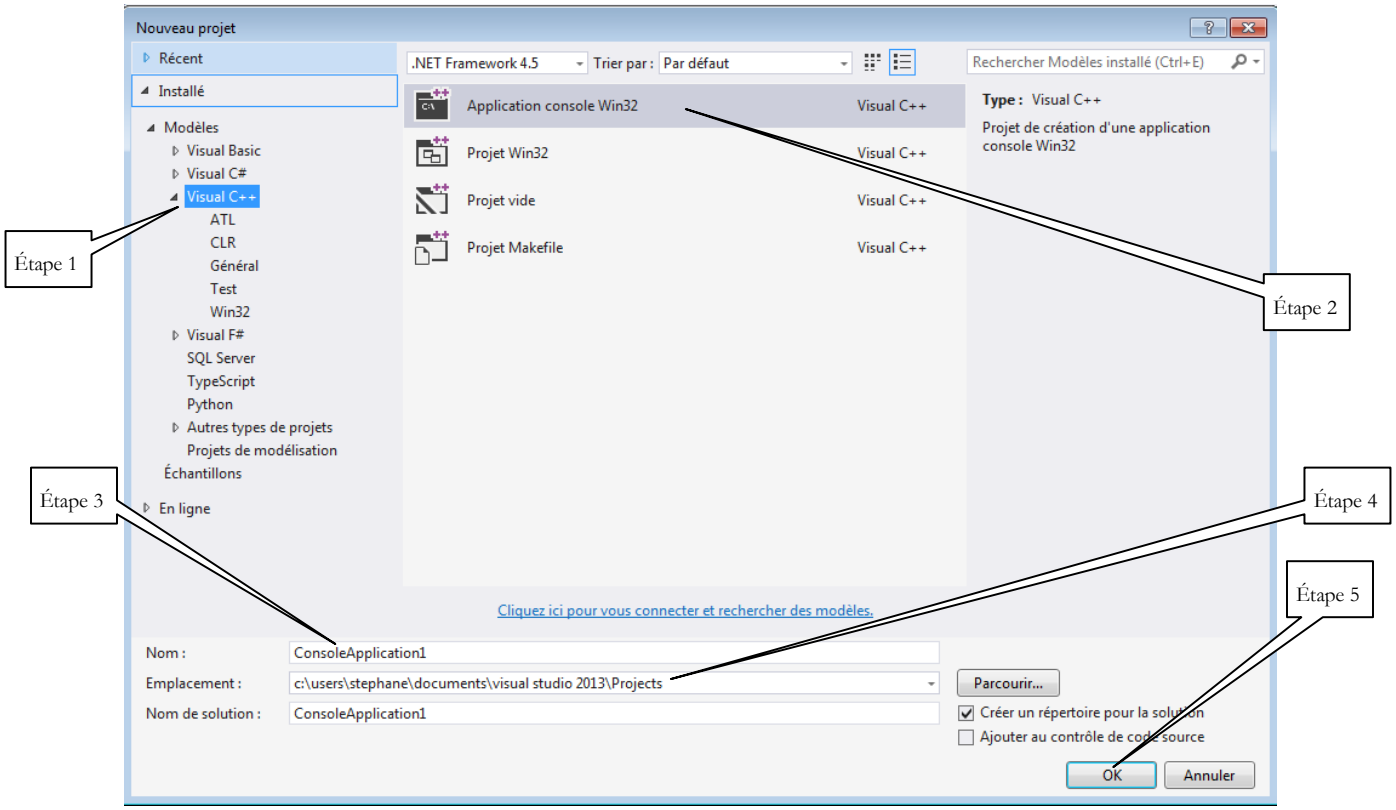
Un projet

Pour utiliser VS et procéder à l'écriture du code, il faut d'abord créer ce qu'on appelle un ***projet***. Un projet nous sera toujours nécessaire pour programmer avec VS; on utilisera un projet distinct par programme développé.

Pour créer un nouveau projet, soit vous cliquez sur le lien de la page de démarrage tel qu'indiqué à la page précédente (voir le schéma), soit vous utilisez le menu

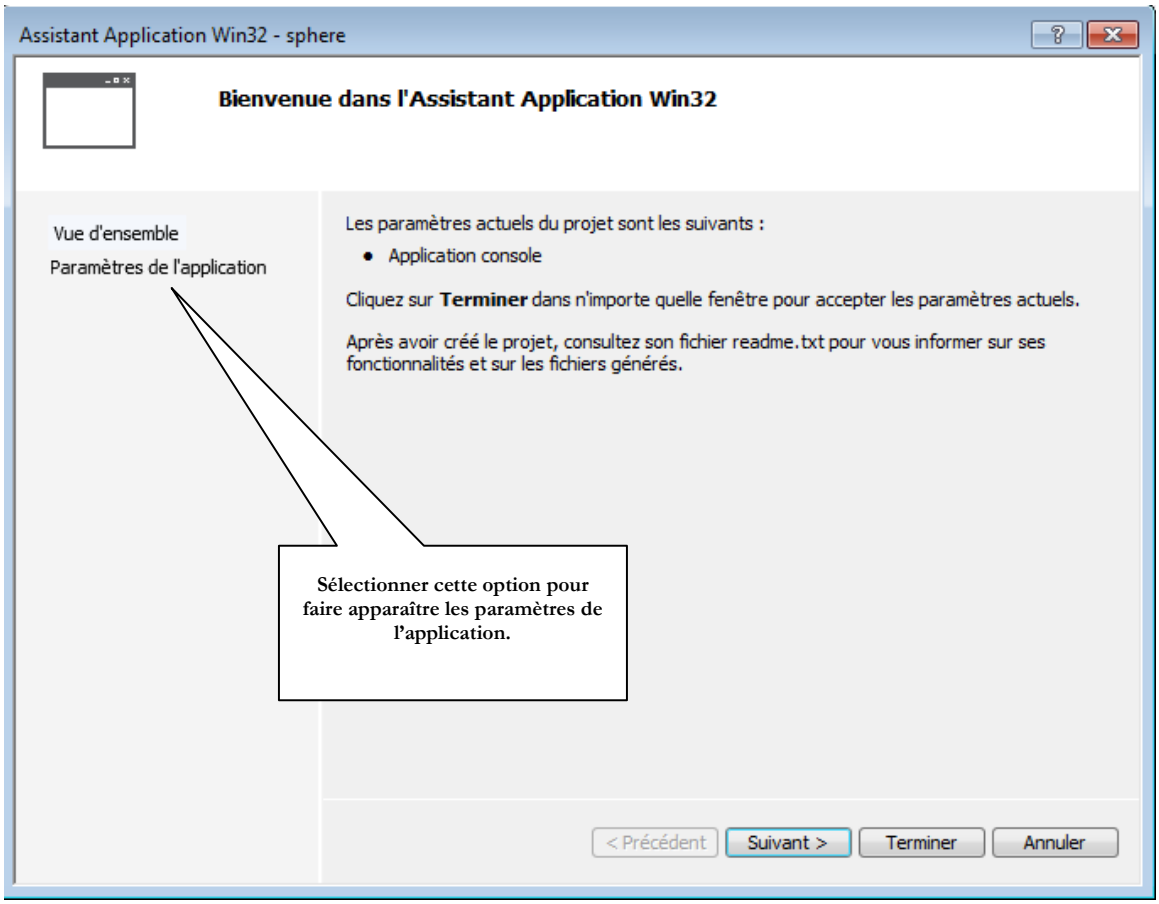
```
Fichier ➔
   Nouveau ➔
      Projet
```

Vous obtenez alors une fenêtre ressemblant à celle-ci:



Étape 1	Nous utiliserons C++. Sélectionnez le dossier **Visual C++**.
Étape 2	Nous développerons du code C++ standard, en évitant les extensions qui ne sont pas portables. Sélectionnez le modèle **Application console Win32**.
Étape 3	Il faut donner un nom à notre projet. Ce petit projet de démonstration se nommera **Sphere**. Un dossier portant le nom du projet sera créé et contiendra l'ensemble des fichiers qui en font partie.
Étape 4	Le dossier dans lequel sera placé le projet doit être placé quelque part. Choisissez un dossier convenable pour travailler.
Étape 5	Lorsque vous aurez terminé, cliquez **OK**

Lorsque vous aurez sélectionné OK, vous verrez apparaître le dialogue suivant :



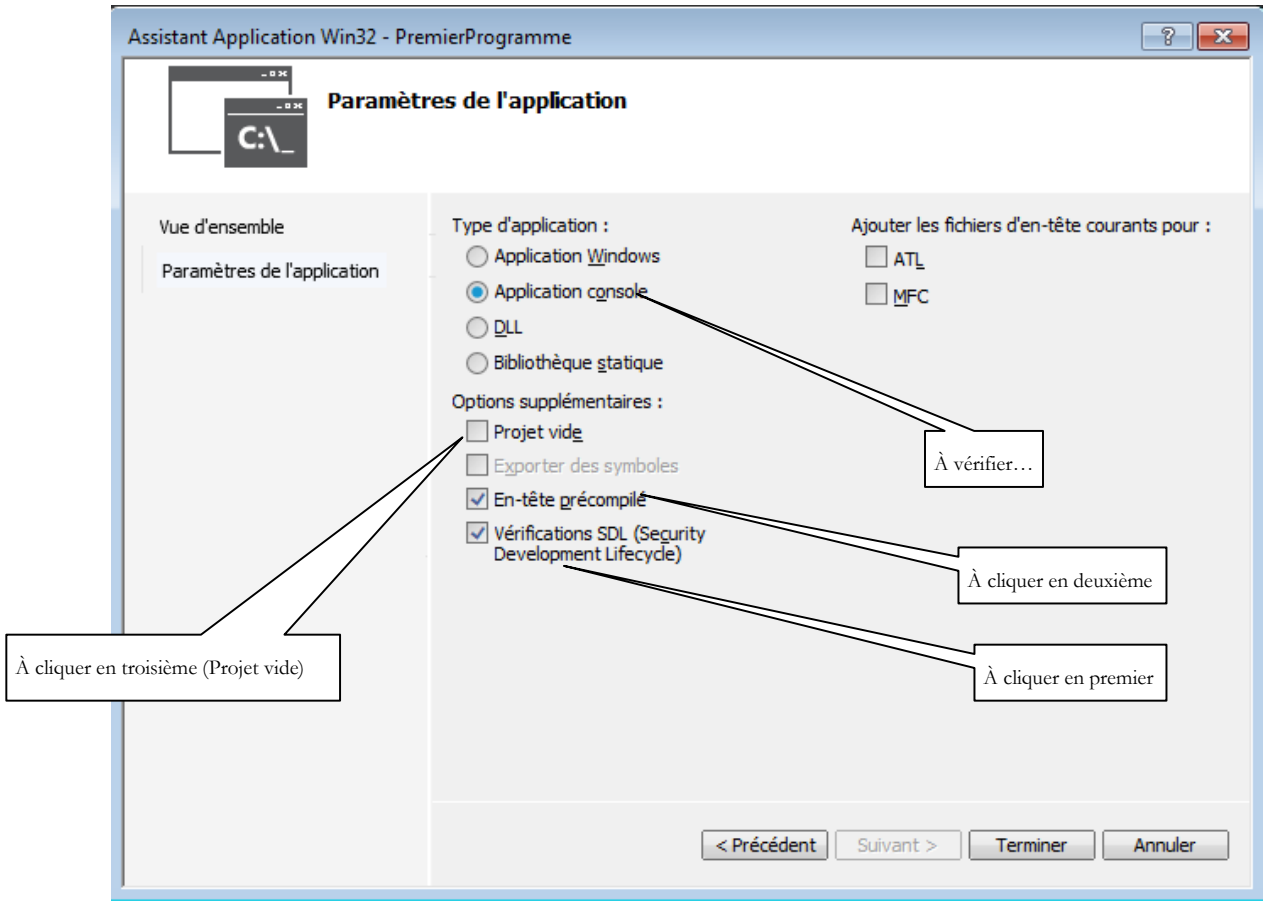
Par défaut, le dialogue nous propose un modèle de projet « *Application console* ».

Sélectionnez donc, à la gauche du dialogue, l'élément « *Paramètres de l'application* ». Ceci nous permettra de faire un autre choix.

Note : dans le cours, nous privilégierons les programmes de type *console*, et nous n'utiliserons que du code C++ standard.

De même, nous utiliserons à chaque fois un projet a priori vide, pour être en mesure de bien saisir toutes les étapes du développement, et pour nous concentrer sur le sujet du cours.

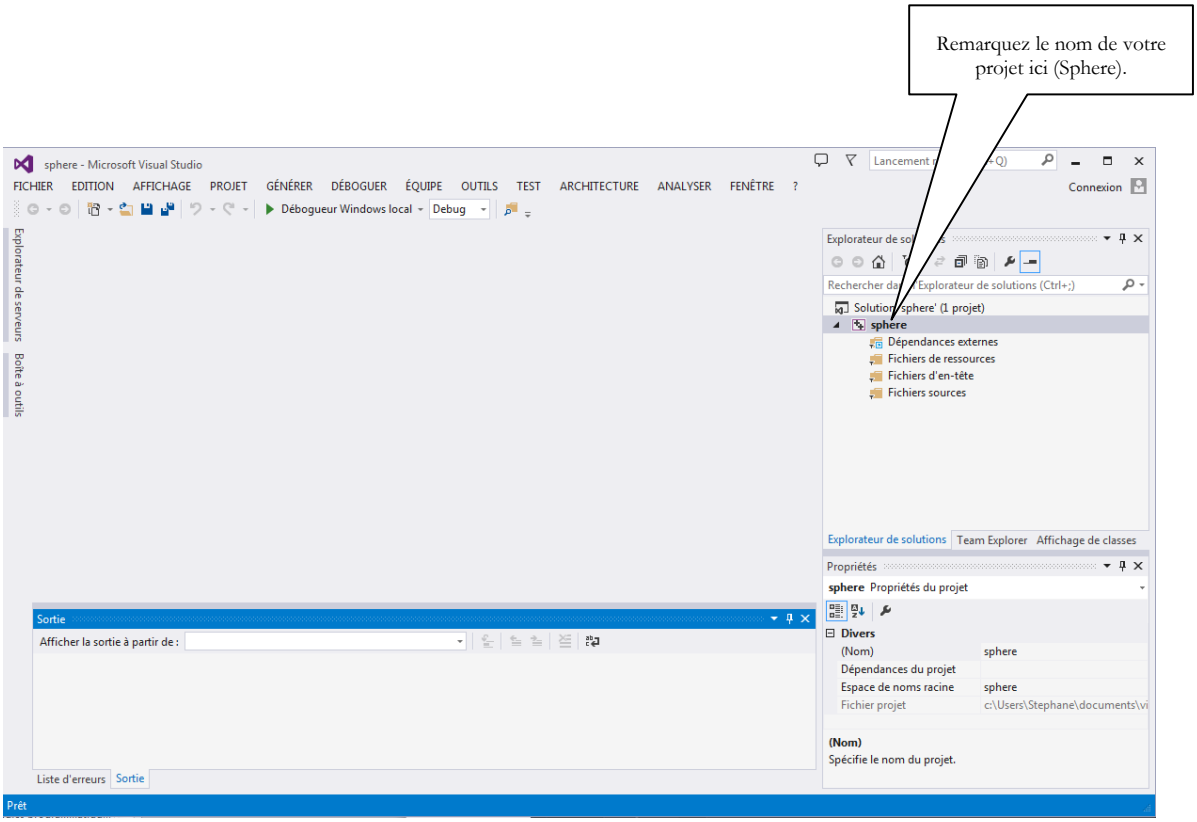
Le dialogue se présentera maintenant comme suit :



Lorsque vous aurez terminé la sélection des options indiquées ci-dessus, sélectionnez Terminer et votre IDE générera le squelette de base du projet. Nous pourrons ensuite commencer.

Faîtes bien attention de « *décocher* » l'option « En-tête précompilé » avant de cocher l'option « Projet Vide ».

L'environnement de travail qui se présente à vous devrait ressembler à ceci :

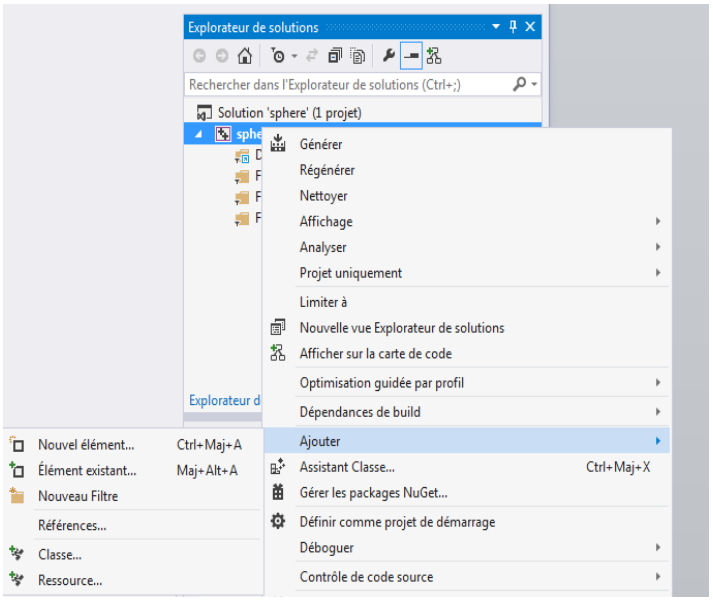


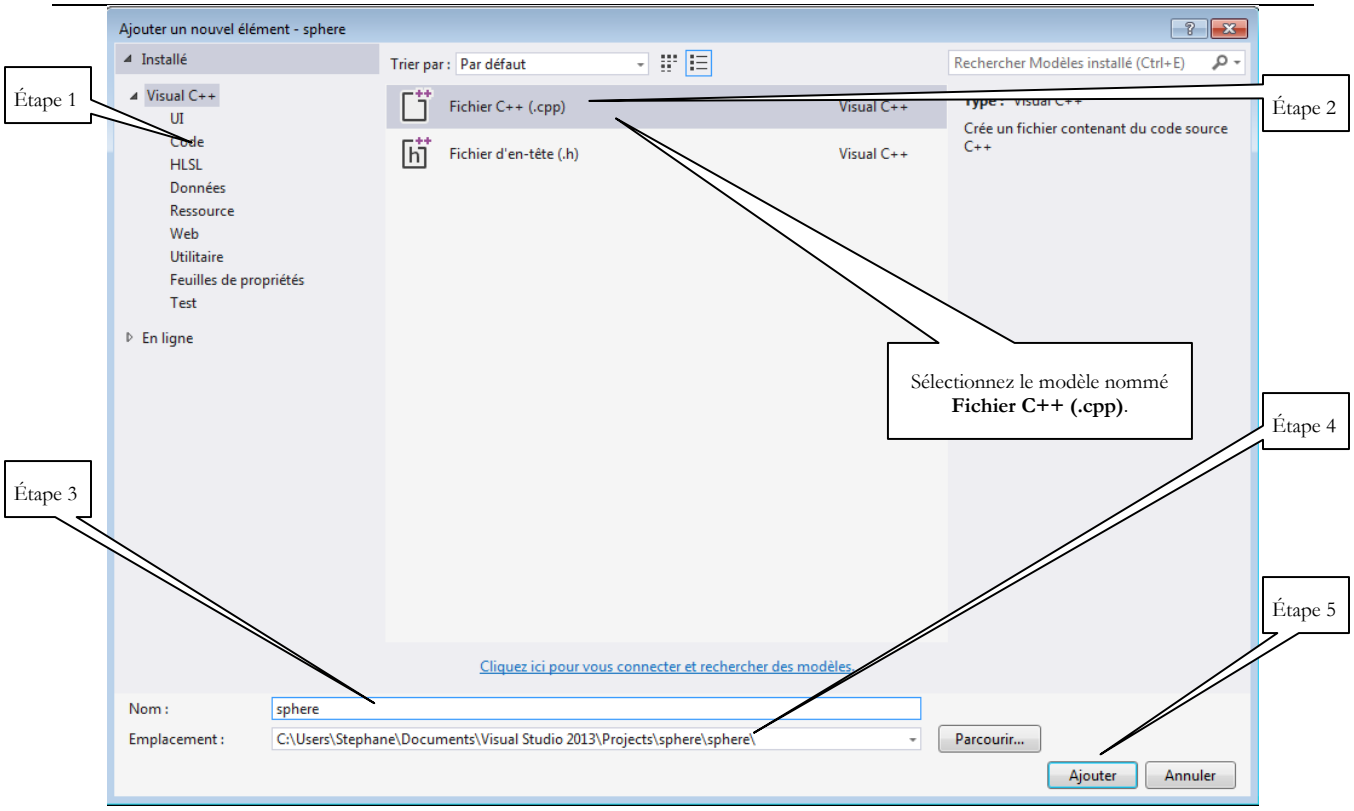
Ajouter un fichier source au projet

Nous avons donc créé un projet. Le projet peut être vu comme notre cadre de travail; mais le projet, ce n'est pas le programme. Et ce programme, il nous faut l'écrire.

Le texte d'un programme C++ tel que nous les rédigerons ici doit être écrit dans ce qu'on nomme un ***fichier source***. Et pour que ce fichier source soit utile au projet, il doit en faire partie.

Vous pouvez ajouter un fichier source en cliquant avec le bouton droit de la souris sur l'élément « Fichiers source », ensuite « Ajouter » et « Nouvel Élément... » comme montré dans la figure à droite.





Étape 1	Il faut sélectionnez la catégorie **Code**.
Étape 2	Nous voulons ajouter un fichier source C++. Un tel fichier porte l'extension **cpp**.
Étape 3	Il faut donner un nom au fichier, et on voudra que ce nom soit significatif. Nous nommerons notre fichier source **Sphere**, ce qui créera un fichier nommé **Sphere.cpp** dans le projet.
Étape 4	Le dossier dans lequel sera enregistré le fichier. Par défaut, l'environnement vous propose le répertoire du projet.
Étape 5	Lorsque vous aurez terminé, cliquez **Ajouter**

Section 2 – La mise en oeuvre

Présumons que nous avons écrit le pseudocode d'un algorithme permettant de trouver l'aire d'une sphère, et que cet algorithme soit celui proposé à droite.

```
Pi ← 3,14159
Lire Rayon
Volume ← 4/3 * Pi * Rayon ^ 3
Écrire Volume
```

Nous avons identifié que cet algorithme requiert :

- du côté des ***données***, une ***variable*** (Rayon), et une ***constante*** (Pi);
- du côté des ***opérations***, une opération de ***lecture*** (Lire Rayon), une opération d'***affectation*** (Volume ← 4/3 * Pi * Rayon * Rayon * Rayon), et une opération d'***écriture*** (Écrire Volume);
- du côté des ***expressions***, une expression arithmétique (4/3 * Pi * Rayon ^ 3).

Nous allons maintenant procéder à sa traduction (on dit aussi son implantation) en langage C++.

Du pseudocode au langage C++

Le langage C++, comme tout autre langage de programmation, offre un ensemble d'instructions permettant d'accomplir les opérations élémentaires qu'on retrouve en pseudocode.

Lecture

L'opération de lecture se représente comme dans le petit tableau visible à droite :

En pseudocode…	***… en langage C++***
Lire X	`cin >> X;`

On peut remplacer, en C++, l'opération Lire par `cin >>`.

Ne confondez pas >>, fait des symboles successifs et collés > et >, avec les guillemets français », *qui n'ont pas de sens en C++.*

Le mot `cin` est généralement pris au sens de *console input*, ou *entrée à la console*[2].

La console est l'écran noir qui apparaît lorsqu'on exécute un programme console.

L'opération `cin >> X` est donc une lecture au clavier.

Remarquez qu'une instruction C++ se termine toujours par un point virgule `;`.

[2] Bien que ce ne soit pas exactement le sens de ce mot.

Écriture

L'opération d'écriture se représente comme dans le petit tableau visible à droite :

En pseudocode…	*… en langage C++*
Écrire X	`cout << X;`

On peut remplacer, en C++, l'opération Écrire par `cout <<`. *Il est important de se souvenir du sens des symboles* << *et* >> *pour utiliser* `cout` *et* `cin` *correctement.*

Le mot `cout` est généralement pris au sens de *console output*, ou *sortie à la console*[3].

L'opération `cout << X` est donc une écriture à l'écran.

Affectation

L'opération d'affectation se représente comme dans le petit tableau visible à droite :

En pseudocode…	*… en langage C++*
X ← 3	`X = 3;`

Remarquez que l'opérateur = en C++ signifie *affecter le résultat de ce qui se trouve à droite dans la variable qui se trouve à gauche*, et rien d'autre.

Expressions arithmétiques

Les expressions arithmétiques les plus élémentaires s'écrivent comme dans le petit tableau visible à droite :

Remarquez qu'en langage C++, il n'existe pas d'opérateur pour représenter l'élévation à une puissance (pas d'exposant).

Heureusement, il existe des solutions pour contourner cette situation, mais nous y reviendrons plus tard cette session.

En pseudocode…	*… en langage C++*
X ← Y + 3	`X = Y + 3;`
X ← Y – 3	`X = Y - 3;`
X ← Y * 3	`X = Y * 3;`
X ← Y / 3	`X = Y / 3;`
X ← Y ^ 3	`X = Y * Y * Y;`

[3] Tout comme pour `cin`, cette interprétation est raisonnable, mais inexacte.

L'enrobage cadeau

Écrire un programme en pseudocode est simple, au sens où pour y arriver, on peut se limiter à l'essentiel, aux idées. Lorsque vient le temps de traduire un programme dans un langage de programmation comme le C++, il faut s'attarder à certains détails plus techniques.

Dans cette section, nous allons écrire en C++ le programme calculant le volume d'une sphère, et nous allons en profiter pour présenter ces détails qu'il vous faudra assimiler—mais n'ayez crainte : vous allez pratiquer suffisamment pour que cela devienne naturel.

L'en-tête du programme

Au début de chaque programme, vous devrez insérer quelques ***commentaires*** pertinents décrivant le programme et son auteur(e).

⇒Un **commentaire** est un bout de texte informatif inséré dans un programme pour les lecteurs humains, et que le compilateur négligera lorsqu'il analysera le programme pour le traduire en langage machine.

En langage C++, un commentaire commence par les symboles ***// et se termine à la fin de la ligne***[4]***.***

L'en-tête du programme devra contenir au minimum le *nom de votre fichier source*, *votre propre nom*, la *date de création* du fichier et une brève description du *but du programme*.

```
// Sphere.cpp
// Fait par Yprogramme Enmasse
// le 21 Août 2014
// But : lit un rayon et calcule le volume de la sphère correspondante
```

Inclusion des bibliothèques

Vous devrez régulièrement inclure ce qu'on appelle des ***bibliothèques*** (en anglais : *libraries*) au début de votre programme.

4 Il existe également une autre syntaxe qui elle permet de commenter plusieurs lignes de code contigües.

⇒Une **bibliothèque** est un regroupement d'outils mis à la disposition des programmes.

Systématiquement, veillez à inclure les deux lignes suivantes au début de chaque programme du cours. Elles feront en sorte que vous pourrez lire (`cin >>`) et écrire (`cout <<`) des données.

```
// Sphere.cpp
// Fait par YProgramme Enmasse
// le 21 août 2014
// But : lit un rayon et calcule l'aire de la sphère correspondante
#include <iostream>
using namespace std;
```

La directive `#include <iostream>` inclut les outils de la bibliothèque d'entrée/ sortie standard (permettant `cin >>` et `cout <<`) de C++.

L'instruction `using namespace std;` est plus technique; prenez-la en tant que nécessité pour le moment, et utilisez-la systématiquement.

Le programme

Un programme C++ suit la forme suivante :

```
int main ()
{
   // le programme va ici...
 return  0;
}
```

On peut donc procéder à l'écriture du programme selon les règles examinées plus haut :

```
// Sphere.cpp
// Fait par Yprogramme Enmasse
// le 21 août 2014
// But : lit un rayon et calcule le volume de la sphère
// correspondante
#include <iostream>
using namespace std;

int main ()
{
   Pi = 3.14159;  // Pi <-- 3,14159
   cin >> Rayon; // Lire Rayon
   Volume = 4/3 * Pi * Rayon * Rayon * Rayon; // Calcule du volume
   cout << Volume; // Écrire Volume
   return 0;
} // fin du programme
```

L'essentiel d'un programme traduit en C++ s'y trouve, et il est facile de voir la correspondance directe entre pseudocode et code C++, ligne à ligne.

Remarquons tout de suite qu'on a écrit, dans le programme C++, la valeur de `Pi` comme étant `3.14159` plutôt que 3**,**14159 (nous avons utilisé le point plutôt que la virgule). *Le langage C++ utilise le point pour marquer le début des décimales*.

Nous avons maintenant une ébauche de programme qui est jolie, mais incomplète. Si nous essayons de traduire ce programme en un format qui soit compréhensible par l'ordinateur, nous obtiendrions plusieurs erreurs.

Vous pouvez vérifier ceci en l'essayant vous-mêmes. Traduire un programme C++ en un format compréhensible par l'ordinateur, c'est ***compiler*** ce programme (menu : Générer➔Générer la solution). Vous trouverez les erreurs générées dans la section Output, en bas et à gauche de votre IDE.

Que manque-t-il à ce programme pour être complet?

Les données

Type d'une donnée

En pseudocode, nous faisons abstraction de ce qu'on appelle le ***type*** d'une donnée. Par exemple, écrire Pi ← 3,14159 nous amène à l'intuition que Pi sera un nombre (réel, en plus), et cela nous suffit.

La plupart des langages de programmation, incluant C++, sont dits *typés*. Il est obligatoire, en C++, de ***déclarer*** chaque variable et chaque constante avant de l'utiliser.

⇒**Déclarer** une variable ou une constante signifie spécifier son **nom** et son ***type***.

⇒ Le **type** d'une donnée nous indique, entre autres choses, l'étendue des valeurs que cette donnée peut prendre.

Pour le moment, nous restreindrons nos types de données aux types suivants : **`int`** pour les nombres entiers, et **`float`** pour les nombres réels.

Les constantes (symboliques)
Les **constantes** sont des données dont la valeur est fixe pour toute la durée de l'exécution du programme. On précédera leur déclaration du mot `const` pour spécifier que ce sont bel et bien des constantes.

Rapprochons notre programme d'un état opérationnel en y déclarant les variables et les constantes qui y sont utilisées :

```
// Sphere.cpp
// Fait par Yprogramme enmasse
// le 21 août 2014
// But : lit un rayon et calcule le volume de la sphère
// correspondante
#include <iostream>
using namespace std;

int main ()
{
   // déclaration de la constante Pi
   const float Pi = 3.14159;  // Pi <-- 3,14159
   float Rayon;  // déclaration de la variable Rayon
   float Volume;   // déclaration de la variable Volume

   cin >> Rayon; // Lire Rayon
   Volume= 4/3 * Pi * Rayon * Rayon * Rayon; // ...
   cout << Volume; // Écrire Volume
   return 0;
} // fin du programme
```

Toujours parce que nous faisons habituellement abstraction du type des données, nous avons en tête que 4/3 vaut 1,333333…

Rien n'est plus faux. ☺

4 et 3 sont des *entiers*. Bien que vous l'ayez déjà vu à l'école élémentaire, vous avez sûrement oublié que la division de 4 en 3 entiers égaux donne *1, reste 1*. Tout comme la division de 14 par 3 donne 4 reste 2, puisque 3 * 4 donne 12, et que 12 + 2 donne 14).

Comment faire alors pour obtenir la valeur 1,3333333… que nous avions en tête, et pour nous assurer que les calculs effectués par notre programme donnent les bons résultats?

Il suffit de remplacer, dans le programme, l'instruction

```
Volume = 4/3 * Pi * Rayon * Rayon * Rayon; // ...
```

par celle-ci :

```
Volume = 4.0/3.0 * Pi * Rayon * Rayon * Rayon; // ...
```

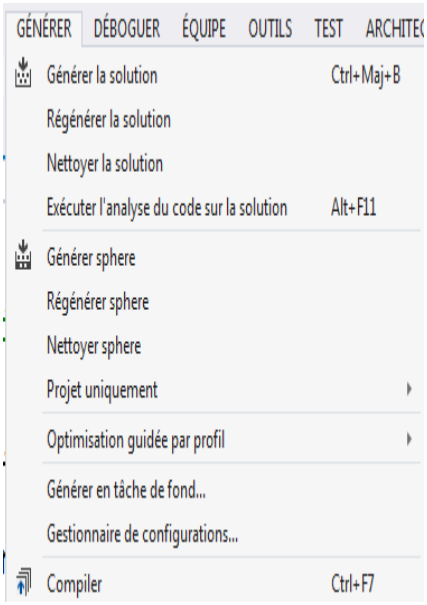
La valeur 4.0 est le nombre réel ayant 4 comme partie entière et à peu près 0 comme partie décimale (la même forme s'applique à 3.0). La division de deux réels donnant un réel, cette ligne nous donnera les bons résultats.

Le programme complet sera donc :

```
// Sphere.cpp
// Fait par Yprogramme Enmasse
// le 21 août 2014
// But : lit un rayon et calcule le volume de la sphère
// correspondante
#include <iostream>
using namespace std;
int main ()
{
   // déclaration de la constante Pi
   const float Pi = 3.14159;  // Pi ← 3,14159
   float Rayon,  // déclaration de la variable Rayon
         Volume;   // déclaration de la variable Volume
   cin >> Rayon; // Lire Rayon
   Volume = 4.0/3 * Pi * Rayon * Rayon * Rayon; // ...
   cout << Volume; // Écrire Volume
   return 0;
} // fin du programme
```

Note : avez-vous enregistré le tout? (Fichier ➔ Enregistrer tout, ou ).

Compiler



Un programme, une fois qu'il est écrit, n'est pas encore prêt à être exécuté. Il doit d'abord être traduit en code machine, pour que l'ordinateur le comprenne. On nommera cette étape la ***compilation***.

⇒**Compiler** un programme signifie le traduire en code machine.

Avec VS, on compilera un programme à l'aide du menu Générer ➔ Compiler. (ou la séquence des touches CTRL-F7)

Portez attention à la fenêtre Sortie (ou Output), en bas et à gauche de votre IDE. Vous y verrez apparaître des messages indiquant le déroulement de la compilation.

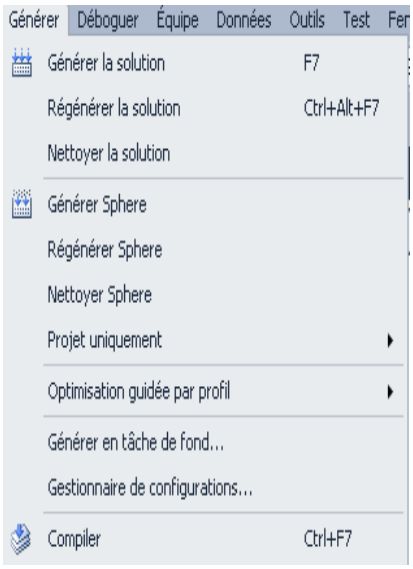
S'il y a des erreurs, essayez de les trouver et de les corriger. Tant qu'il y aura des erreurs dans le programme, il sera impossible de l'exécuter.

Si vous avez exactement le programme proposé dans ce document, la fenêtre Output devrait contenir à peu près le texte suivant (suit à la compilation de Sphere.cpp) :

```
------ Début de la génération : Projet : sphere, Configuration : Debug Win32 ------
  sphere.cpp
c:\users\stephane\documents\visual studio 2013\projects\sphere\sphere\sphere.cpp(11):
warning C4305: 'initialisation' : troncation de 'double' à 'const float'
c:\users\stephane\documents\visual studio 2013\projects\sphere\sphere\sphere.cpp(15):
warning C4244: '=' : conversion de 'double' en 'float', perte possible de données
========== Génération : 1 a réussi, 0 a échoué, 0 mis à jour, 0 a été ignoré ==========
```

Note :	remarquez la présence de deux avertissements (*warning*) mais d'aucune erreur. Nous ne porterons pas attention à ces avertissements pour le moment; un avertissement indique un problème dans le programme qui n'est pas assez grave pour en empêcher l'exécution. Nous reviendrons sur ces avertissements un peu plus tard.

Éditer les liens

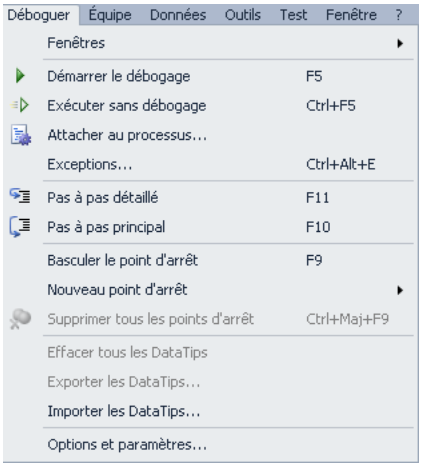


La compilation seule n'est pas suffisante pour nous donner quelque chose qu'on puisse exécuter (un *exécutable*). Une fois le programme compilé correctement, il faut en éditer les liens (faire ce qu'on appelle souvent le *link* ou le *build*).

Avec VS, on transformera le programme en exécutable à l'aide du menu Générer ➔ Générer la Solution (ou, dans notre cas, simplement Générer ➔ Générer Sphere).

Encore une fois, si des erreurs se produisent, vous les verrez listées dans la fenêtre Sortie (ou Output). Si aucune erreur ne s'est produite, vous venez tout juste de générer un exécutable nommé Sphère.exe.

Exécuter le tout

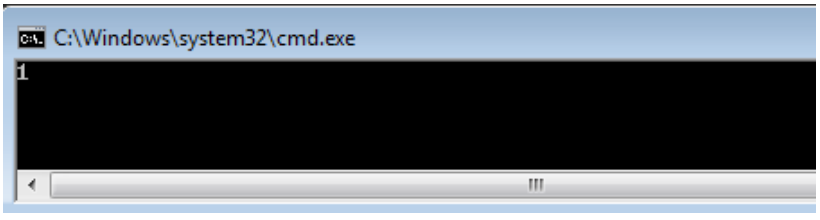


Si l'édition des liens a été résolue sans erreurs, vous pouvez exécuter votre programme.

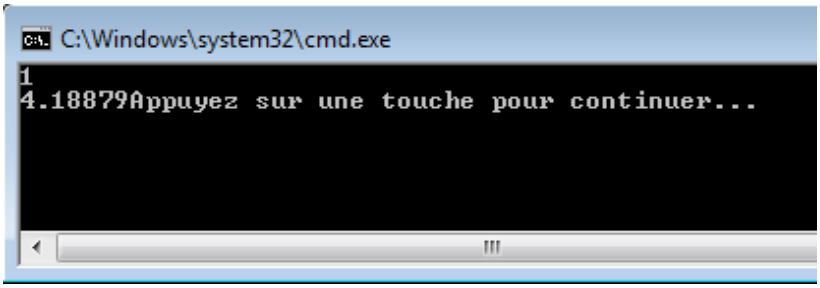
Avec VS, on exécutera le programme à l'aide du menu Deboguer➔Exécuter sans débogage ou CTRL-F5.

Ceci créera une fenêtre de type console. Puisque la première chose que votre programme fait est Lire Rayon, vous aurez à entrer une valeur réelle.

Supposons qu'on entre la valeur 1. On peut voir (à droite) de quoi aurait alors l'air l'écran console.



Une fois que vous aurez appuyé la touche Entrée (return), le programme devrait procéder à ses calculs et écrire le résultat à l'écran (à droite).

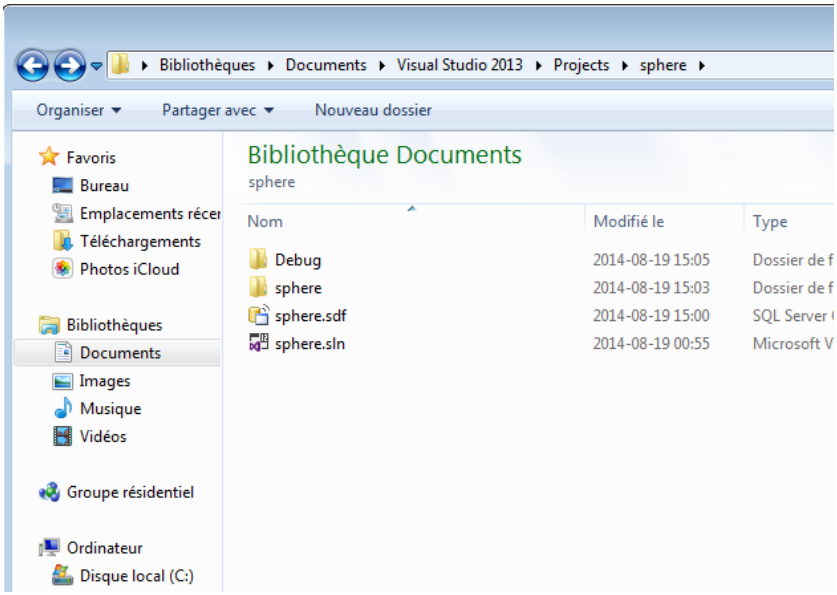


Les fichiers contenus dans notre projet

Un projet créé avec VS se nomme une *solution*. Lorsque nous avons créé un projet C++ console vide (*Empty project*) nommé ***Sphere***, VS a généré pour nous :

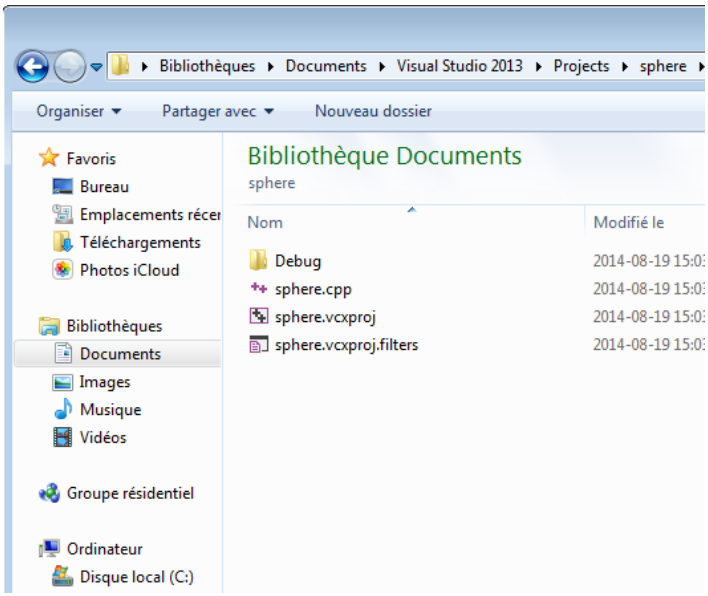
un dossier nommé Sphere; et
parmi les fichiers du dossier, vous retrouverez respectivement Sphere.sdf, Sphere.sln.

En tout temps, si vous désirez ouvrir votre projet tel qu'il était lors de sa dernière utilisation, un double clic sur le fichier portant l'extension **.sln** de ce projet (ici, Sphere.sln) l'ouvrira tel quel.

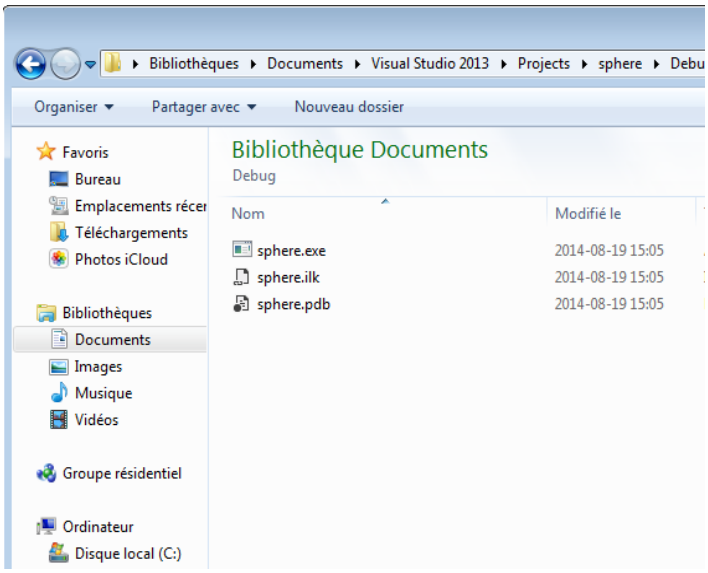


Notre IDE sera en mesure, à partir du fichier .sln, de retrouver correctement tous les autres fichiers du projet.

Quand on ajoutera des fichiers à notre projet, par exemple des fichiers sources, on les trouvera dans le répertoire « Sphere ». Ces fichiers sont tous relativement petits pour un projet simple comme le projet Sphere. Remarquez le fichier « Sphere.cpp » qui contient votre code source.



Lorsqu'on procédera à au moins une compilation ou un Build, VS créera un dossier nommé Debug dans le dossier du projet, et y insérera les fichiers qu'il aura générés en procédant à la traduction. Remarquez le fichier « Sphere.exe » qui est l'exécutable de votre projet.



Note :

Lorsque vous faites une copie de sauvegarde de votre projet, règle générale, vous pouvez détruire le répertoire Debug du projet et tout son contenu. Cela limitera beaucoup vos besoins en fait d'espace. Vous pouvez aussi, avec VS, faire Générer➔ Nettoyer la solution. Ce qui fera précisément la même chose.

Section 3 — Règles et normes de programmation

Note :	ce qui suit constitue un sous-ensemble des règles à appliquer dans la rédaction de vos programmes.

Règles du langage

Dans un programme C++…

- les majuscules et les minuscules sont importantes. Les mots `int` et `Int` sont différents l'un de l'autre, et il en va de même pour `main` et `MAIN` ;
- à part les commentaires, les inclusions de bibliothèques et les accolades ouvrantes et fermantes (comme celles qui accompagnent `int main ()` par exemple), chaque instruction C++ se termine par un point virgule **`;`** ;
- il est important de ne pas inverser le sens des symboles **`>>`** et **`<<`** auprès de `cin` et `cout` ;
- pour chaque parenthèse ouvrante **`(`**, il y a une parenthèse fermante **`)`** ;
- pour chaque accolade ouvrante **`{`**, il y a une accolade fermante **`}`** ;
- il est important de ne pas se tromper de type de projet ou de type de fichier lorsqu'on crée un projet et un fichier source lors de la rédaction du code.

Règles programmatiques

Dans un programme C++, selon les normes de ce cours :

- on ne met jamais plus d'une instruction par ligne ;
- on ne déclare pas plus d'une variable et pas plus d'une constante par ligne ;
- on doit identifier clairement chaque programme ;
- on doit s'assurer que toutes les instructions du programme sont facilement compréhensibles, notamment en ajoutant des commentaires judicieux ;
- on doit utiliser en tout temps des noms significatifs pour les variables et les constantes—une variable servant à contenir la valeur du rayon de la sphère se nommera **`Rayon`**, pas `R` ou `r` ou `xyz123`. On mettra normalement la première lettre du nom en majuscule, le reste en minuscules. Si votre variable a un nom composé de plusieurs mots (par souci de clarté), on les collera tous ensemble, en mettant une majuscule au début de chaque mot. Par exemple : **`RayonSphere`**. Un nom de variable ne peut pas contenir d'espaces.

Règles programmatiques propres à l'indentation

L'indentation est un terme utilisé pour dénoter les règles de présentation générale du code. Pour le moment, gardez en tête que (a) le code ne doit pas être tassé à gauche ou à droite sans raisons, et que (b) lorsque du code se trouve entre deux accolades, il doit être poussé à droite d'un nombre fixe d'espaces par rapport à ces accolades.

- on doit toujours indenter notre code : toute ligne d'instructions se trouvant entre une ligne où se trouve une accolade ouvrante et celle où se trouve l'accolade fermante correspondante doit se trouver en retrait à droite de 3 ou 5 espaces précisément par rapport à ces accolades ;

Correct	*Incorrect (1)*	*Incorrect (2)*

`int main ()`
`{`
`  int X;`
`}`
	`int main ()`
	`  {`
	`  int X;`
	`  }`
		`int main ()`
		`  {`
		`int X;`
		`}`

Incorrect (3)	***Incorrect (4)***	***Incorrect (5)***

`int main ()`
`{`
`int X;`
`}`
	`int main () {`
	`  int X; }`
		`int main ()`
		`{ int X;`
		`}`

- une accolade ouvrante «{» doit se trouver au même niveau d'indentation que la ligne qui la précède, et à laquelle elle est rattachée (exemples incorrects 1, 2 et 4) ;

- on ne doit trouver aucune instruction sur la même ligne qu'une accolade ouvrante **{** (exemples incorrects 4 et 5) ;

- on ne doit trouver aucune instruction sur la même ligne qu'une accolade fermante} (exemple incorrect 4) ;

- une accolade fermante doit se trouver au même niveau d'indentation que l'accolade ouvrante à laquelle elle correspond (exemples incorrects 2 et 4).