Programmation C++ (débutant)/Notions de base

Le cours du chapitre 1 : Notions de base

Les langages de programmation

Le C++ est un **langage de programmation** : il sert donc à écrire des applications informatiques. Il s'agit d'ailleurs d'un des langages de programmation les plus utilisés aujourd'hui. Chaque programme en C++ doit être écrit en respectant des règles d'écriture très strictes que nous étudierons petit à petit.

Un langage compilé

Le C++ est un langage compilé : pour écrire un tel programme, il faut commencer par écrire un ou plusieurs fichiers source. Ensuite, il faut compiler ces fichiers source grâce à un programme appelé compilateur afin d'obtenir un programme exécutable. Cette phase s'appelle la compilation. Les fichiers source sont des fichiers texte lisibles dont le nom se termine en général par .c, .cpp ou .h. Les fichiers exécutables portent en général l'extension .exe sous windows et ne portent pas d'extension sous Linux.

Les compilateurs

Il existe de très nombreux compilateurs : on peut citer par exemple Visual C++ (de microsoft), C++ Builder (de Borland), ou encore gcc qui est un excellent compilateur libre.

Les environnements de développement intégrés (EDI)

On programme très souvent en utilisant un environnement de développement intégré : il s'agit d'un ensemble complet d'outils permettant d'éditer et de modifier des fichiers sources, de les compiler, de lancer l'exécutable, de "déboguer" le programme, etc... Visual C++ (version express disponible gratuitement), C++ Builder, Dev-cpp (disponible gratuitement et basé sur gcc) et Code::Blocks (lui aussi gratuit mais plus souvent mis à jour que Dev-cpp) sont des environnements de développement intégrés.

Le C et le C++

Le langage C est un langage de programmation inventé par MM. Kernighan et Ritchie au début des années 70. Au début des années 90, Bjarne Stroustrup fait évoluer le langage vers le langage C++ en lui rajoutant notamment les notions orientées objet. Toutefois, bien que le C++ ait évolué à partir du C, et ait gardé un grand nombre de notions et de syntaxes de son «ancêtre», il s'agit de deux langages différents (le langage C étant étudié dans un livre dédié).

Aspects pédagogiques

Dans ce cours, nous allons d'abord apprendre les notions non orientées objet du C++ (donc nous étudierons principalement les notions du langage C), puis seulement dans un second temps nous étudierons les notions orientées objet. Dans un troisième temps, nous aborderons l'étude des interfaces graphiques. Il existe un débat incessant sur LA bonne manière de faire : quand faut-il aborder la notion d'objet ? Pour des débutants, je conseille plutôt de procéder selon ce plan.
On remarquera que la notion d'interface graphique ne sera abordée que tout à la fin du cours : nous commencerons donc à écrire des programmes en mode texte dont l'interface graphique sera rudimentaire. Ceci est très frustrant pour le débutant mais semble absolument nécessaire.

Un premier exemple

Nous allons maintenant étudier ce premier exemple :

Exemple 1 : un premier exemple

```
#include <iostream>
using namespace std;

int main()
{
   cout << "BONJOUR";
   return 0;
}
```

La directive #include

On place en général au début du programme un certain nombre d'instructions commençant par #include. Cette instruction permet d'inclure dans un programme la définition de certains objets, types ou fonctions. Le nom du fichier peut être soit à l'intérieur des chevrons < et >, soit entre guillemets :

- `#include <nom_fichier>` Inclut le fichier nom_fichier en le cherchant d'abord dans les chemins configurés, puis dans le même répertoire que le fichier source,
- `#include "nom_fichier"` Inclut le fichier nom_fichier en le cherchant d'abord dans le même répertoire que le fichier source, puis dans les chemins configurés.

using namespace std;

Cette ligne est un peu plus difficile à comprendre : en effet, on indique par cette ligne l'utilisation de l'espace de nommage std. Un espace de nommage est un ensemble de classes dont cout fait partie. Etant donné que nous voulons utiliser l'objet cout, nous indiquons que l'on utilisera, par défaut, l'espace de nommage std. Pour simplifier, retenons que, dès que l'on veut utiliser cin ou cout, on doit écrire cette directive.

Il faut également remarquer que les fichiers d'en-tête standard ne sont désormais plus nommés avec une extension .h (comme iostream.h). Si ces fichiers d'en-tête sont inclus sans être suivi de la commande **using namespace std;**, cela ne fonctionnera pas correctement. Dans certaines versions de g++ , si, lors de la compilation, vous spécifiez un fichier d'en-tête standard avec une extension .h (comme iostream.h), le compilateur utilisera le fichier "backward" compatible et vous signifiera un avertissement.

Le fichier iostream

Le fichier iostream contient un certain nombre de définitions d'objets intervenant dans les entrées/sorties du programme, c'est-à-dire dans l'affichage à l'écran ou dans des fichiers. La définition de cout se trouve dans ce fichier; pour utiliser cout dans notre programme, il faut inclure au début du programme la ligne suivante :

1. include <iostream>

Ce fichier est fourni par l'éditeur du compilateur : il s'agit d'un fichier C++ standard.

La fonction main()

Notre programme contient une fonction appelée main : c'est à cet endroit que va commencer l'exécution du programme : exécuter un programme en C++, c'est exécuter la fonction main de ce programme. Tout programme en C++ doit donc comporter une fonction main.

La fonction main apparaît sous la forme suivante :

```
int main()
{
on place ici une liste d'instructions en C++
}
```

La liste d'instructions entre accolades est exécutée séquentiellement : on exécute chaque instruction, dans l'ordre, les unes après les autres.

cout

Il s'agit du flux de sortie du programme (***C**onsole **Out**put* : sortie console). Ce flux de sortie est envoyé par défaut vers l'écran. Il va nous servir à afficher des messages à l'écran en utilisant l'opérateur <<. Cet opérateur à la forme d'une flèche semblant indiquer le sens de transfert des données (écriture vers la console).

Exemple : cout<<"BONJOUR";
Cette instruction affiche BONJOUR à l'écran.
Un autre exemple :
cout<<endl;
Lorsqu'on envoie endl (***End** of **L**ine* : fin de la ligne) vers l'affichage, on passe à la ligne suivante.

Il faut également connaître une écriture plus condensée. Au lieu d'écrire en 3 instructions :

```
cout << "BONJOUR";
cout << endl;
cout << "AU REVOIR";
```

On peut écrire en une seule instruction :

```
cout << "BONJOUR" << endl << "AU REVOIR";
```

Cependant, sur certaines implémentation, cette instruction condensée ne compile pas correctement, car l'implémentation du symbole `endl` ne permet pas d'utiliser l'opérateur `<<` par la suite :

```
cout << "BONJOUR" << endl;
cout << "AU REVOIR";
```

Retour de la fonction

La dernière instruction de notre programme est `return 0;` Elle indique seulement que la fonction main s'est terminée correctement sans erreur particulière.

Exécution du programme

Lorsqu'on édite notre fichier source, puis compile et enfin exécute notre programme il s'affiche alors à l'écran :

```
BONJOUR
```

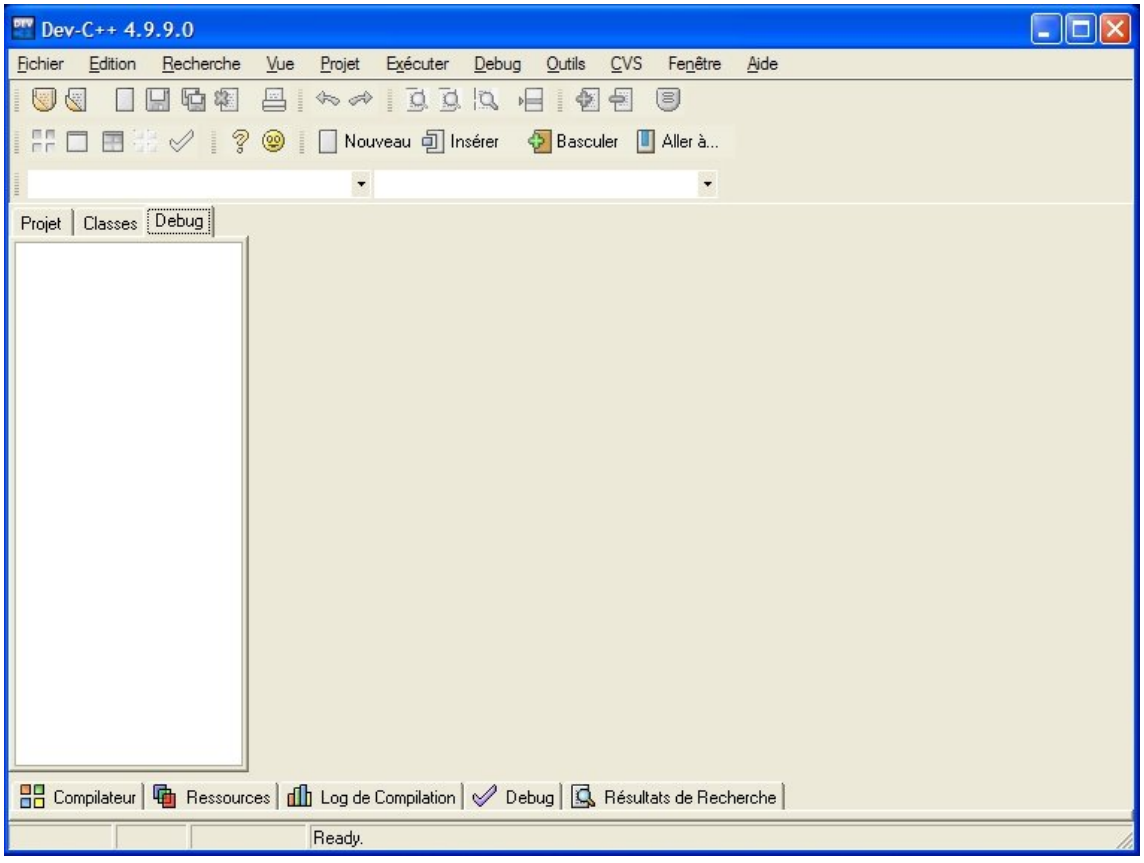
Remarque

L'ajout de l'instruction `system("PAUSE");` sera parfois nécessaire pour que le programme ne s'arrête pas immédiatement après s'être ouvert. Cette instruction doit être ajouté avant l'instruction `return 0;`.

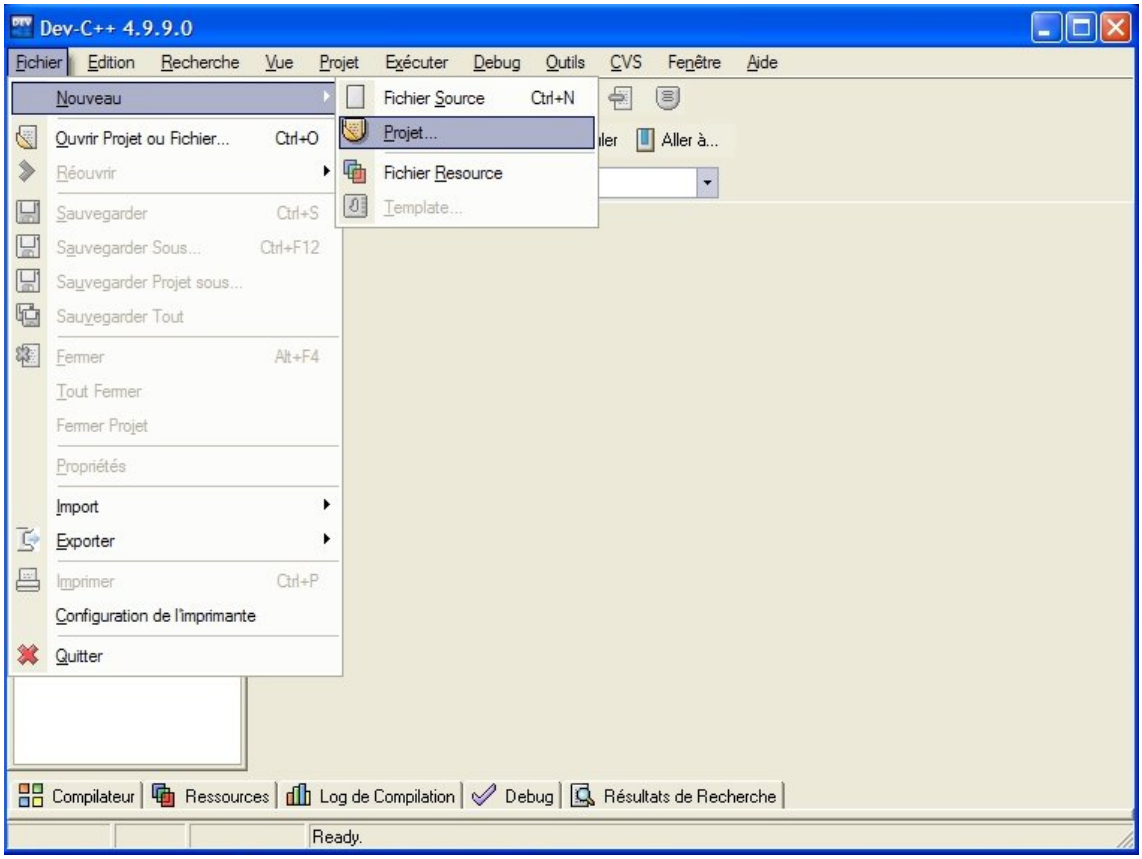
Aspect pratique

Utiliser Dev-Cpp sous Windows

- Lorsque vous lancez Dev C++, vous voyez apparaitre l'écran ci-contre.

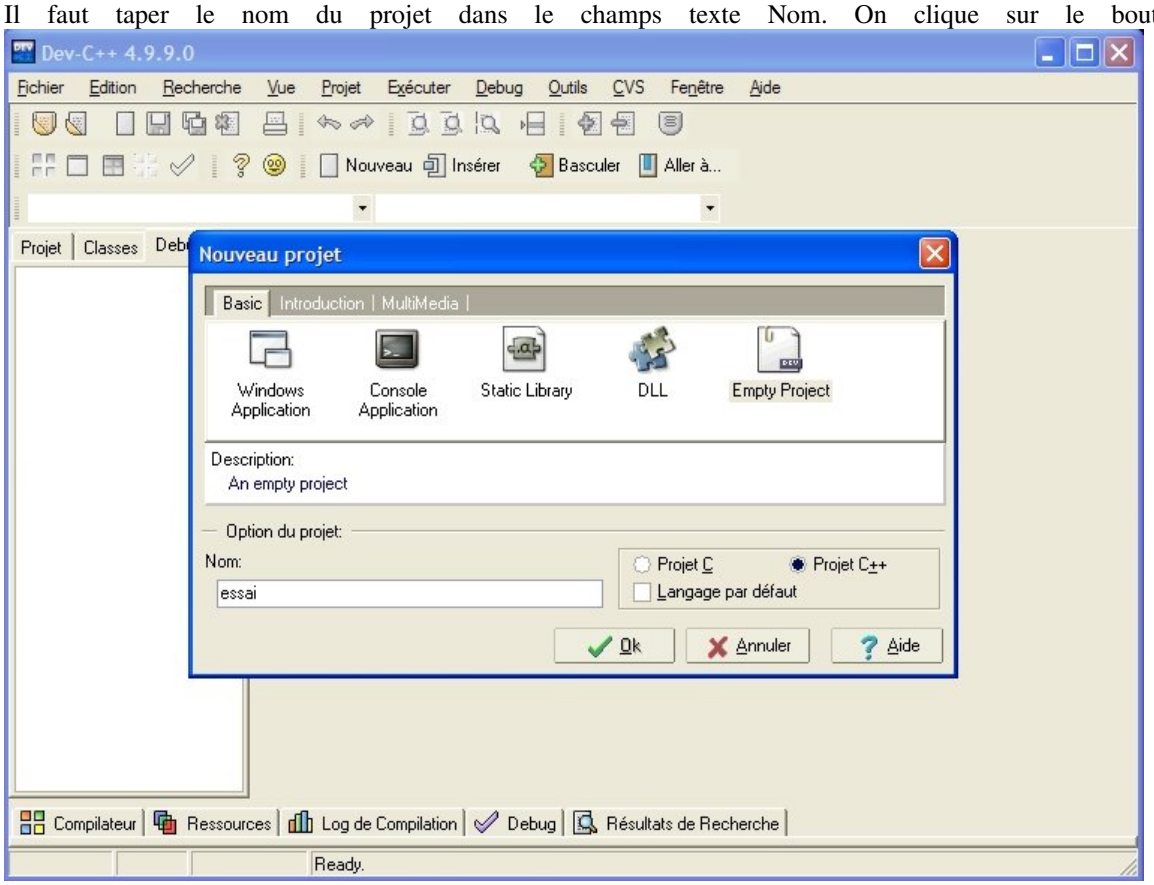


- Pour créer un nouveau projet, il faut choisir dans le menu Fichier puis Nouveau puis Projet.



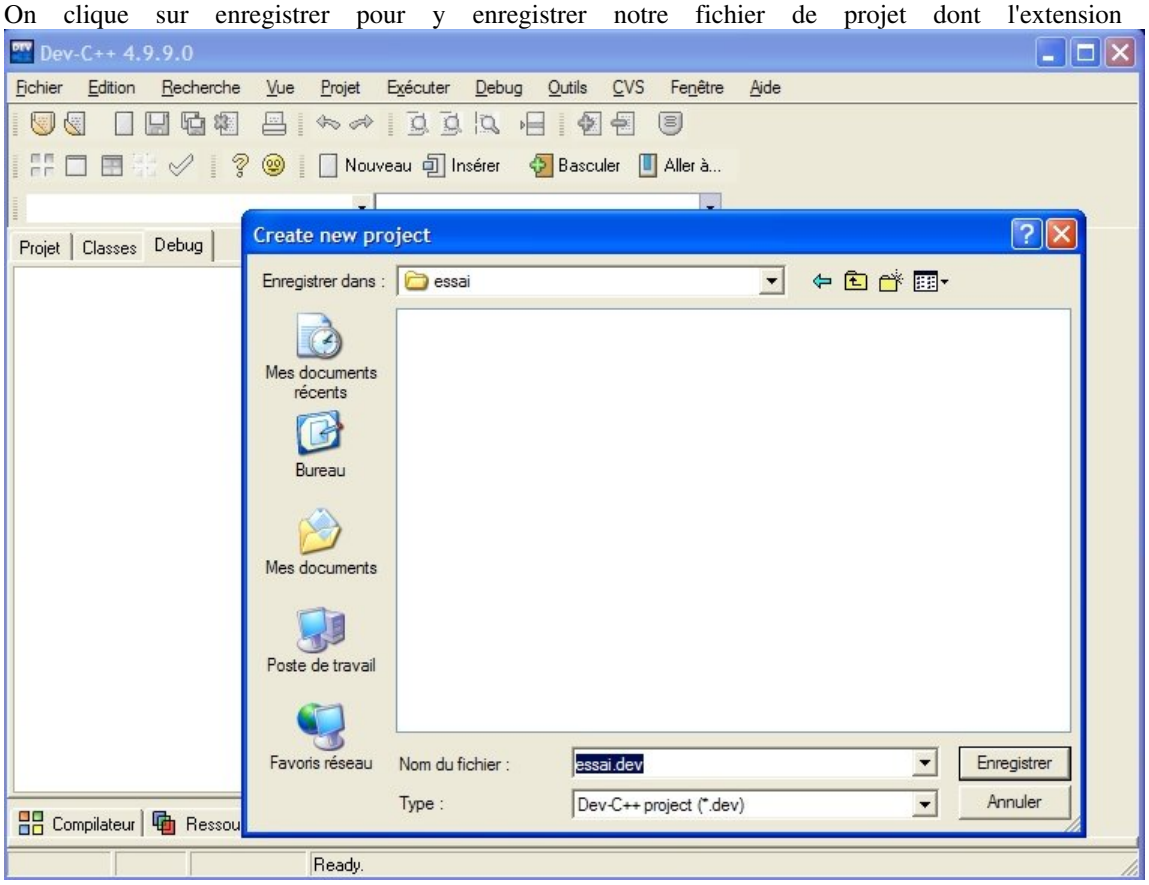
- Il faut faire le choix "Empty project".

Il faut taper le nom du projet dans le champs texte Nom. On clique sur le bouton OK.

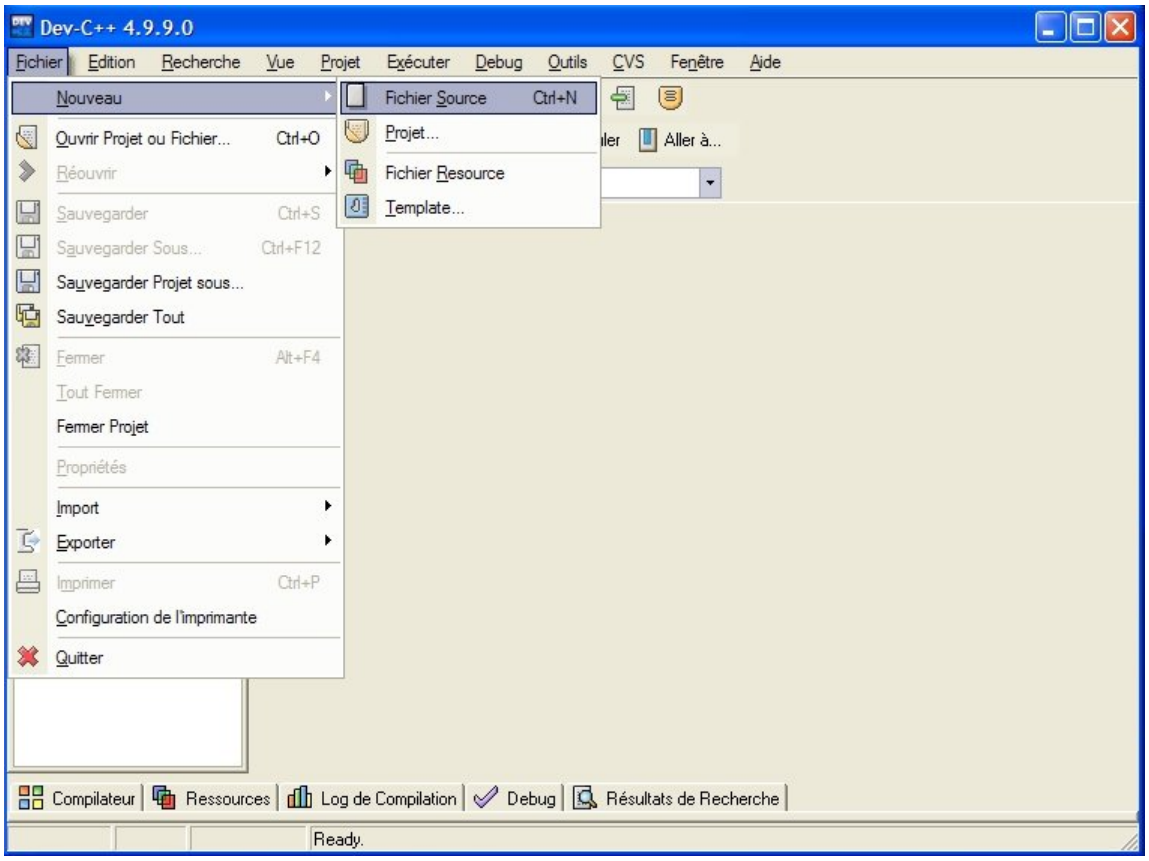


- Il s'ouvre alors une fenêtre qui nous permet de créer le répertoire de notre projet.

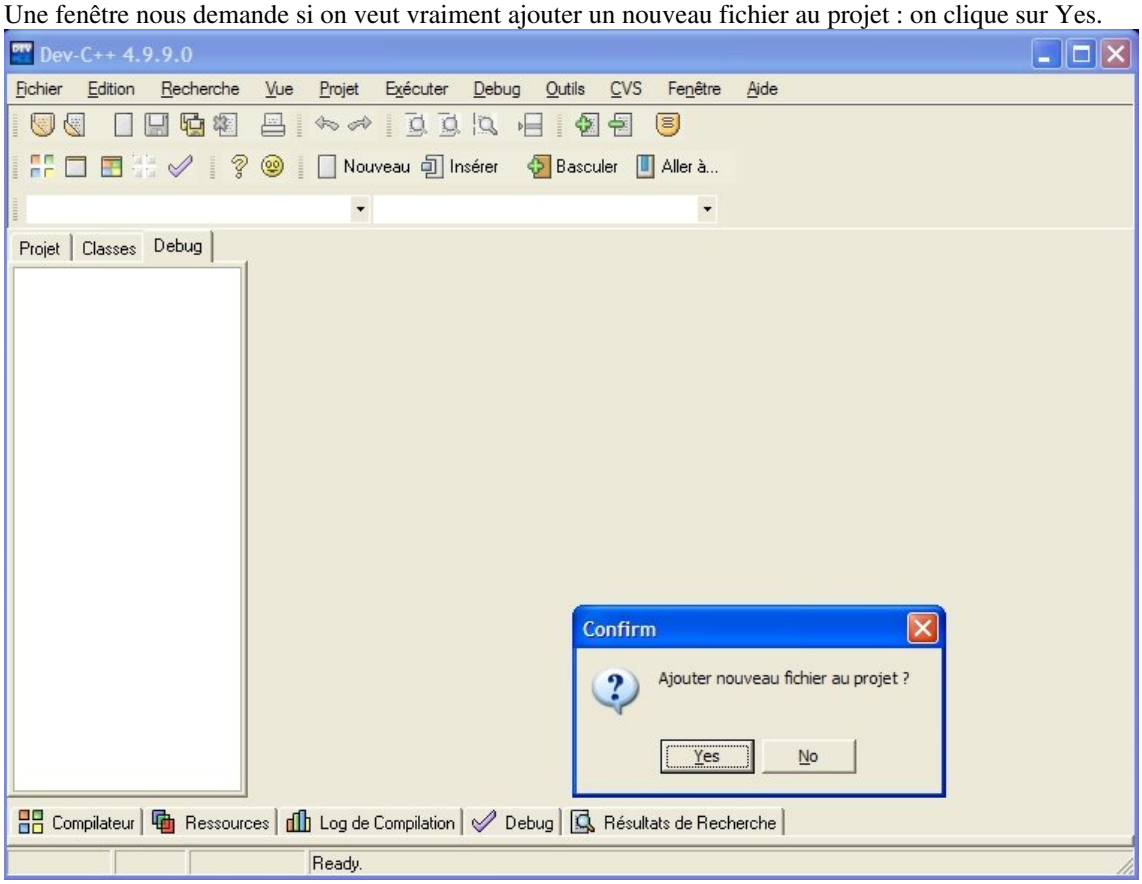
On clique sur enregistrer pour y enregistrer notre fichier de projet dont l'extension est .dev.



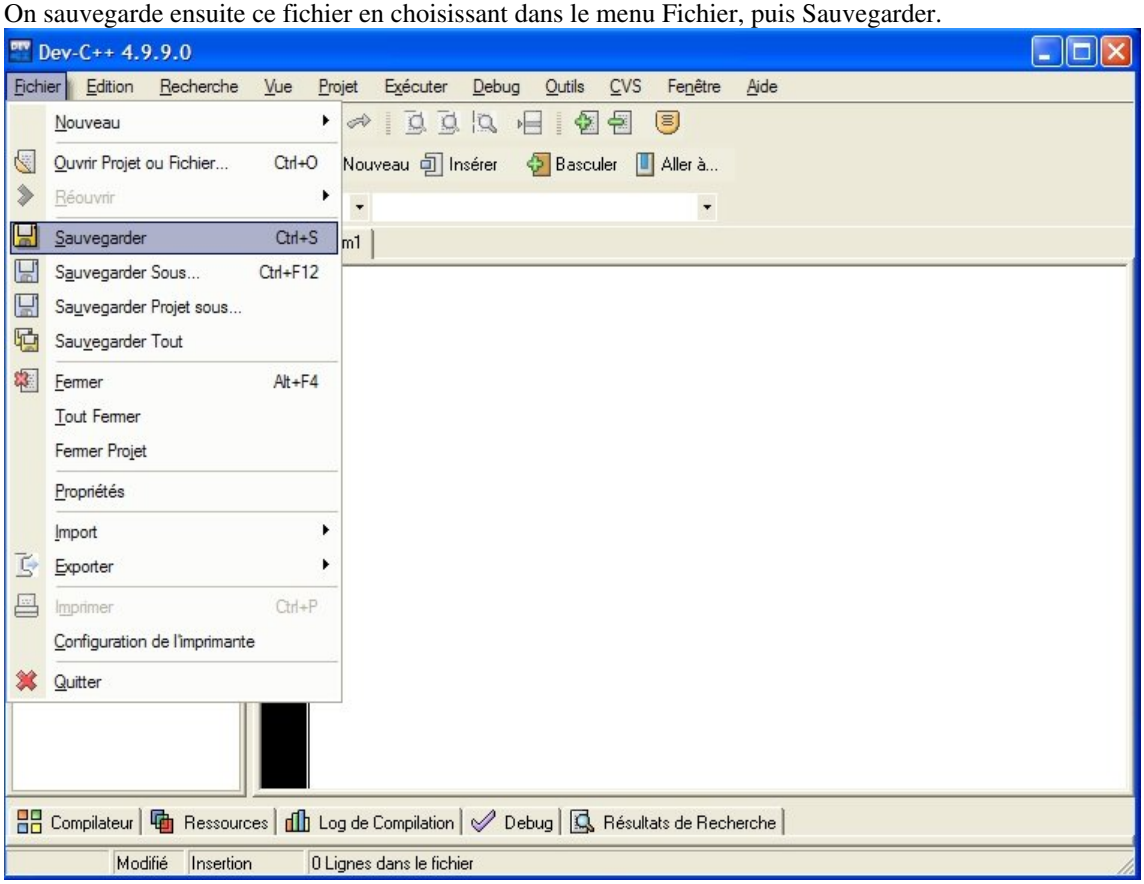
- Pour rajouter un nouveau fichier source dans notre projet, il faut choisir dans le menu Fichier, puis Nouveau puis Fichier Source.



- Une fenêtre nous demande si on veut vraiment ajouter un nouveau fichier au projet : on clique sur Yes.

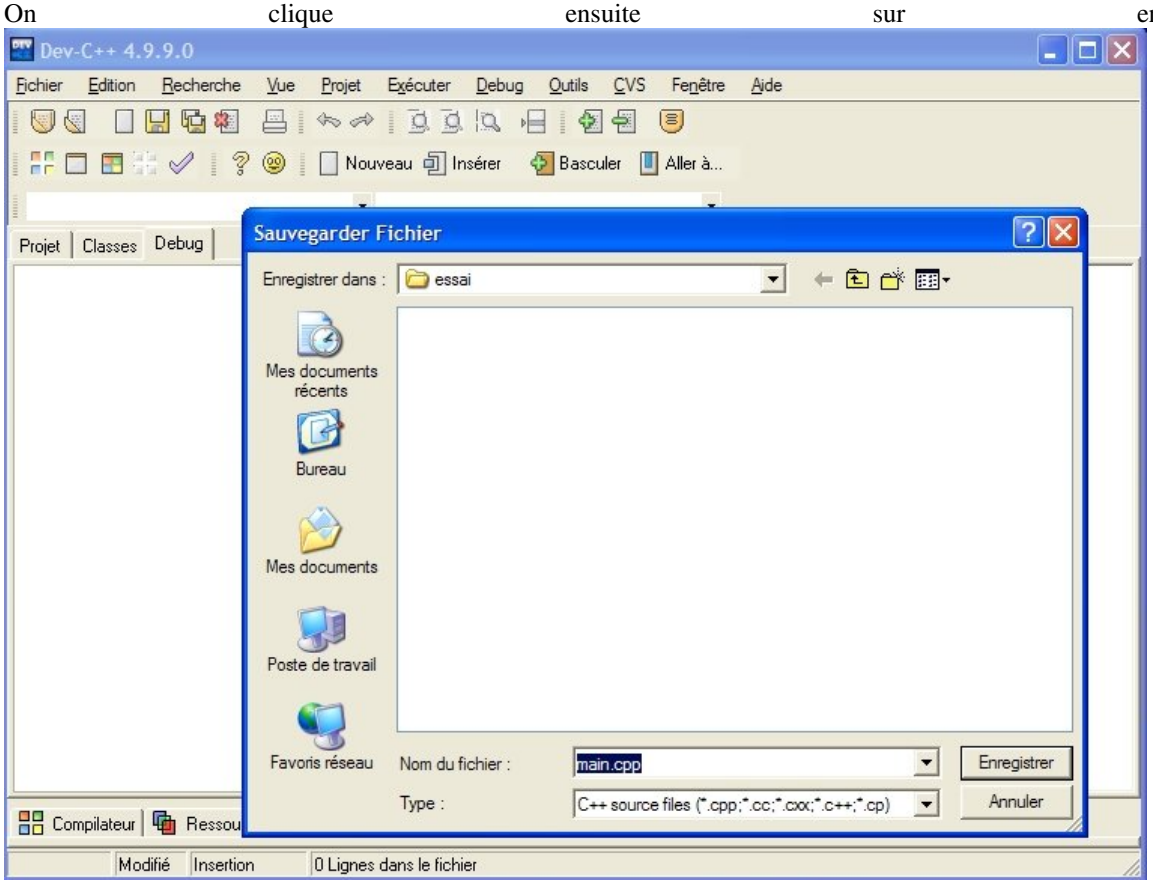


- On sauvegarde ensuite ce fichier en choisissant dans le menu Fichier, puis Sauvegarder.



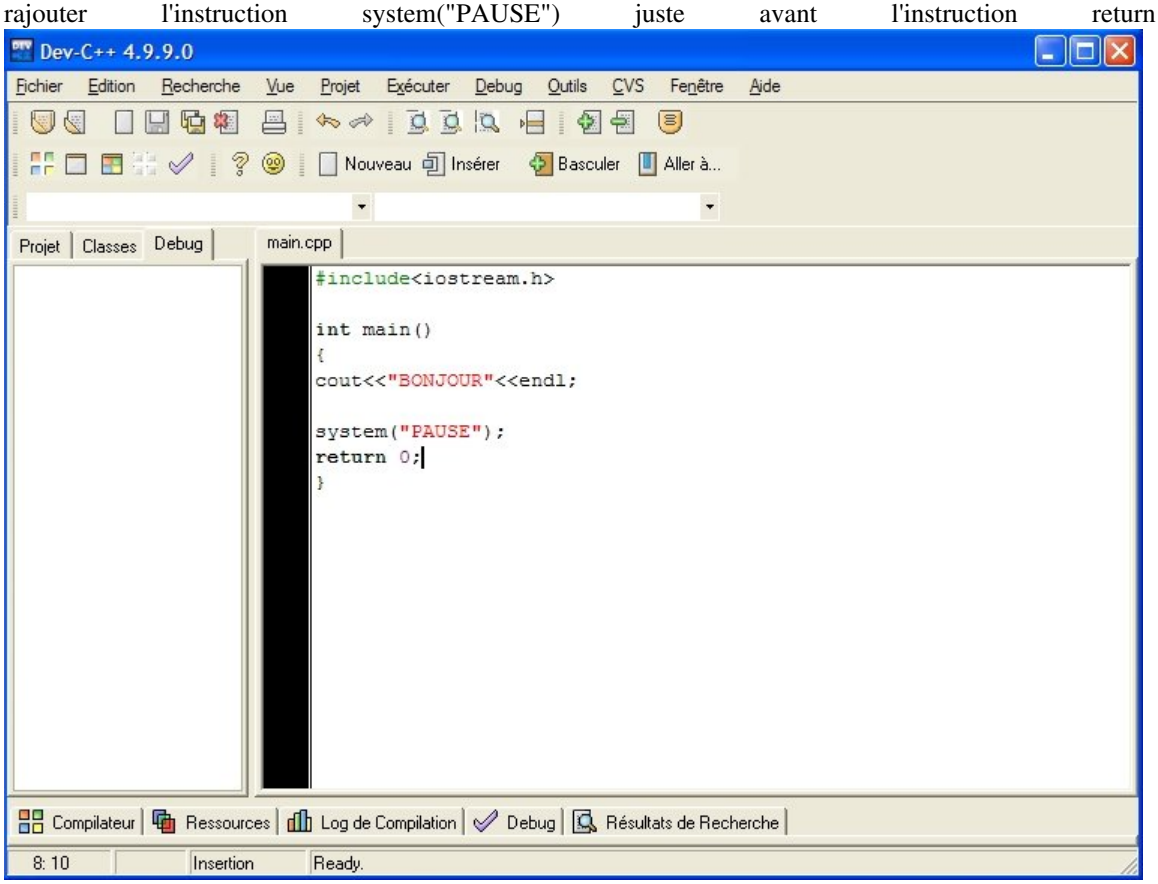
- On nous demande alors le nom du fichier à ajouter: ici on l'a appelé main.cpp.

On clique ensuite sur enregistrer.

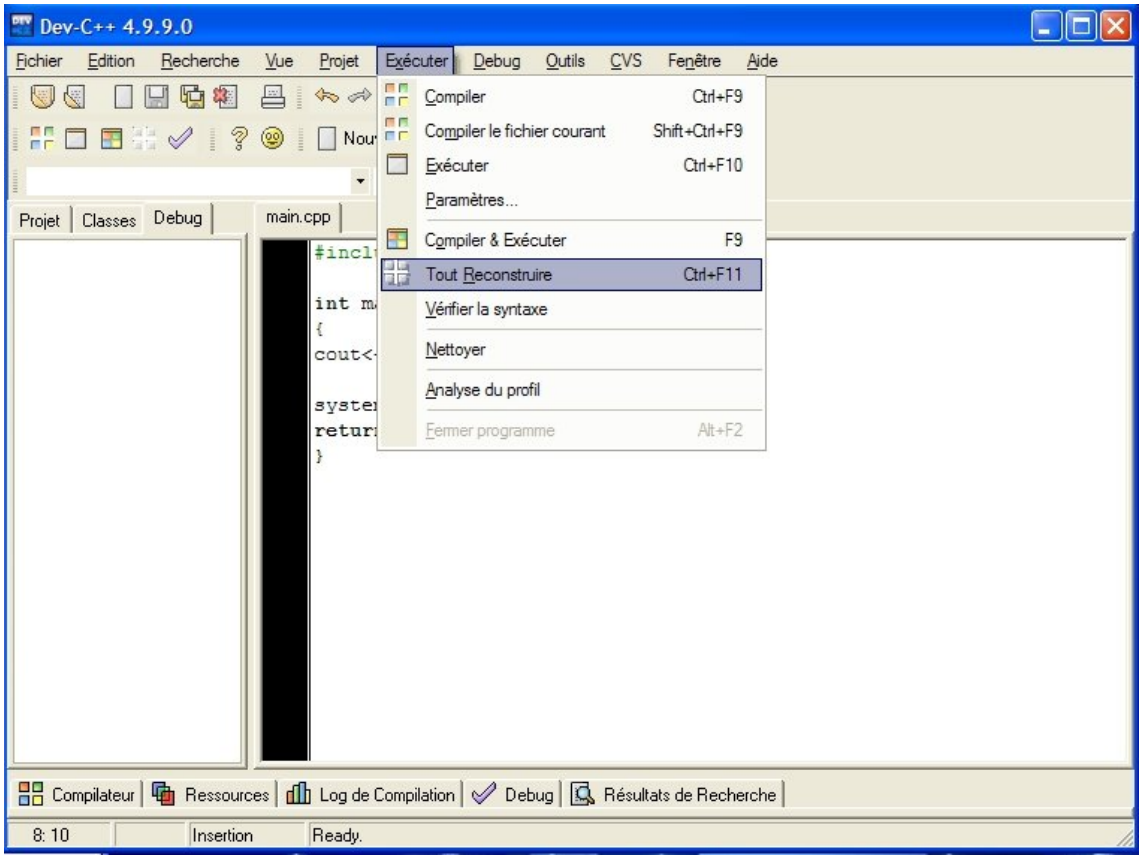


- On tape alors notre ficher source.

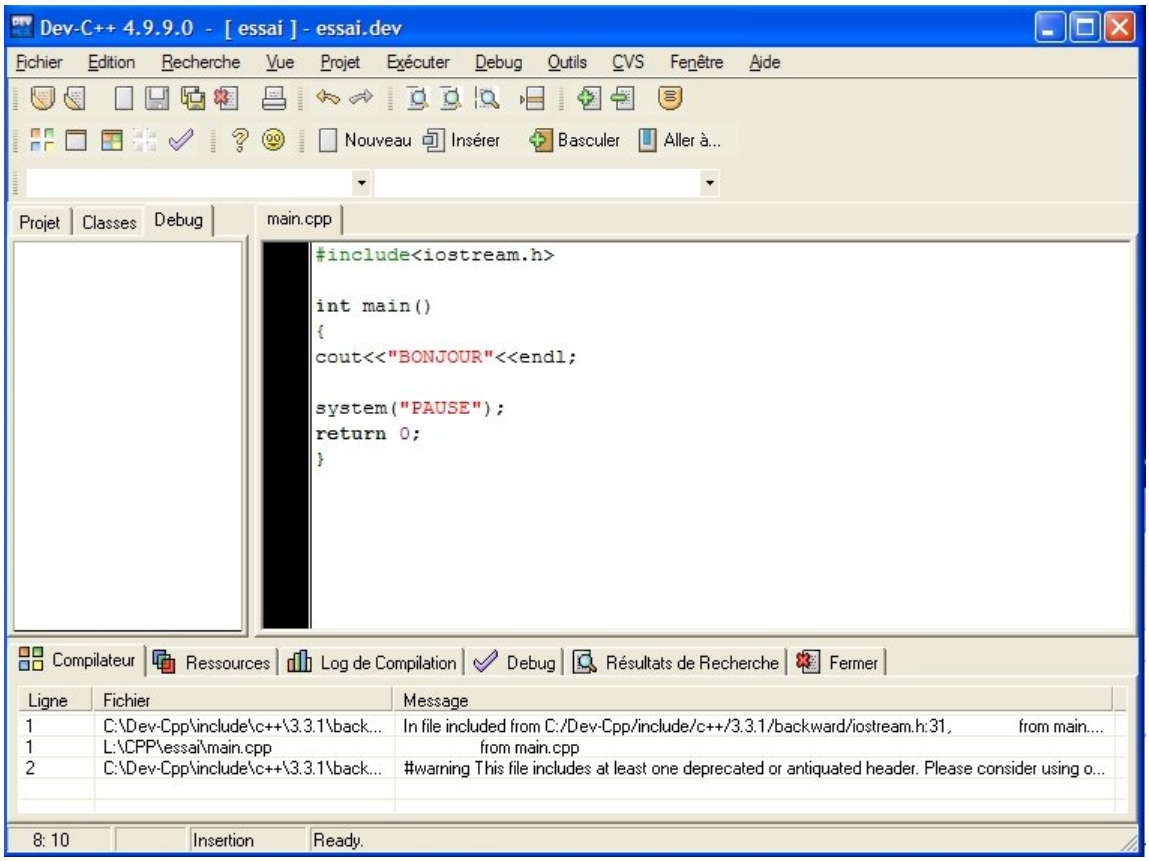
On l'enregistre en cliquant sur Fichier puis Enregistrer. Pour une application en mode texte, il est recommandé de rajouter l'instruction system("PAUSE") juste avant l'instruction return 0;.



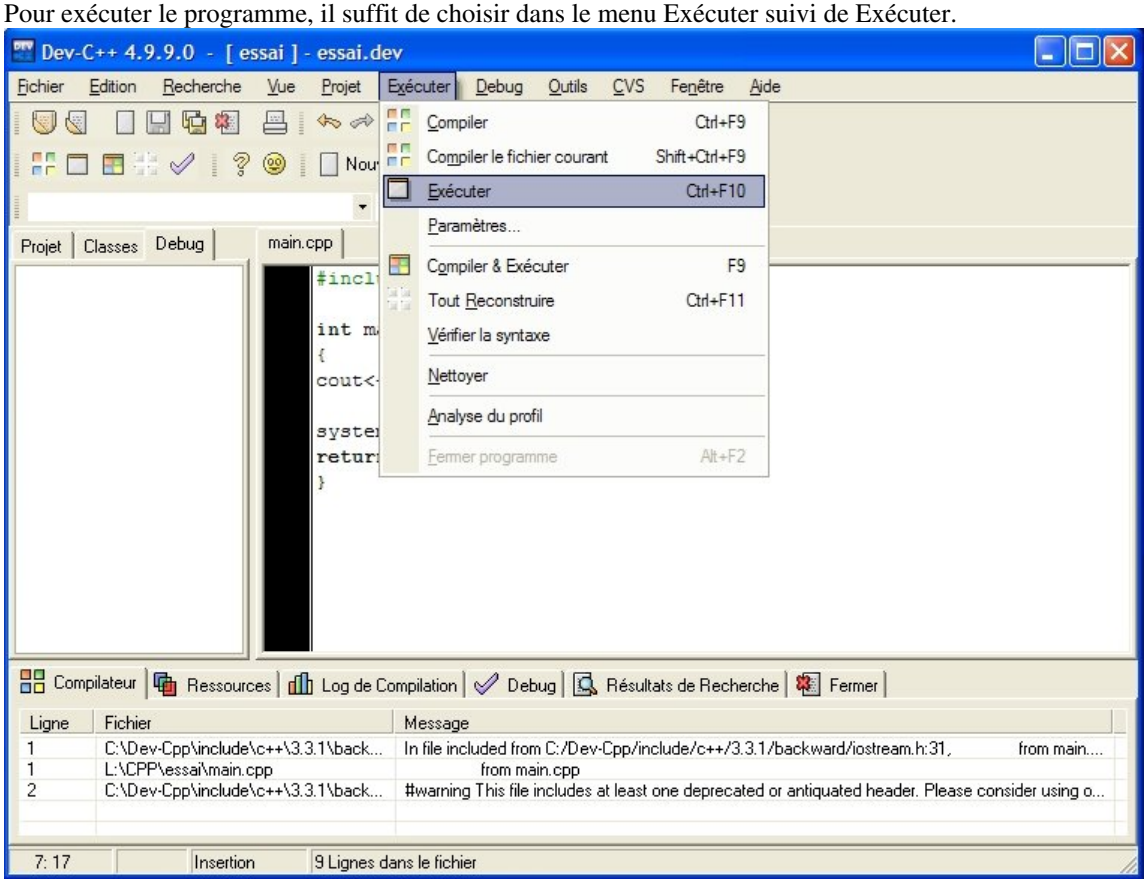
- Pour compiler notre programme, on choisit dans le menu Exécuter puis Tout reconstruire.



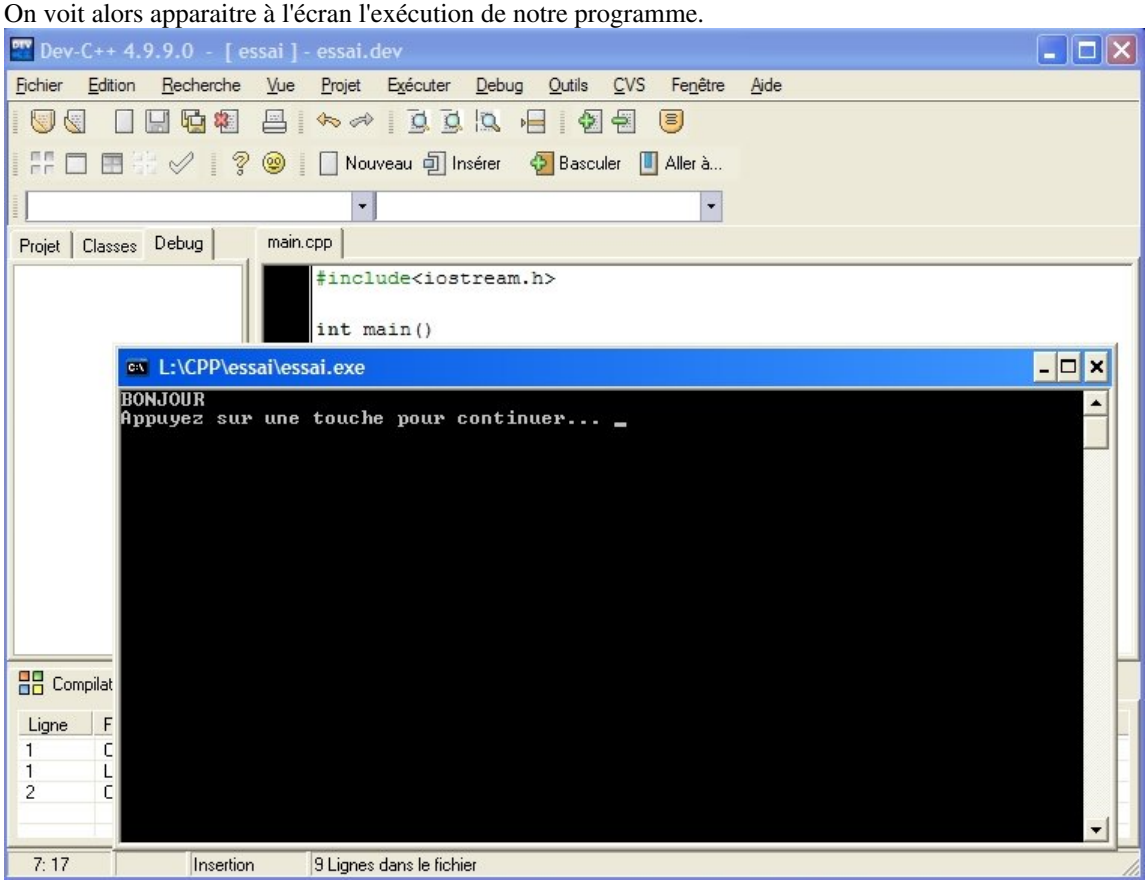
- Les éventuelles erreurs de compilation apparaissent dans la fenêtre du bas.



- Pour exécuter le programme, il suffit de choisir dans le menu Exécuter suivi de Exécuter.

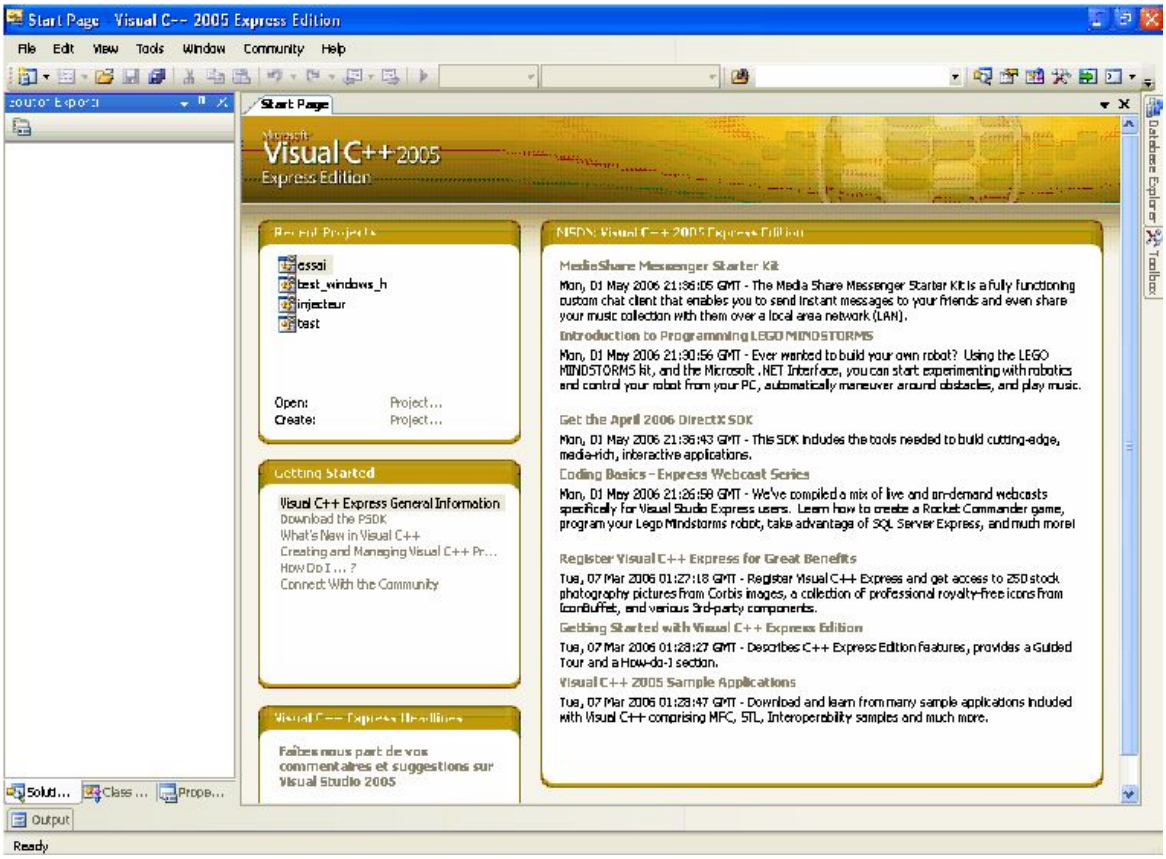


- On voit alors apparaitre à l'écran l'exécution de notre programme.

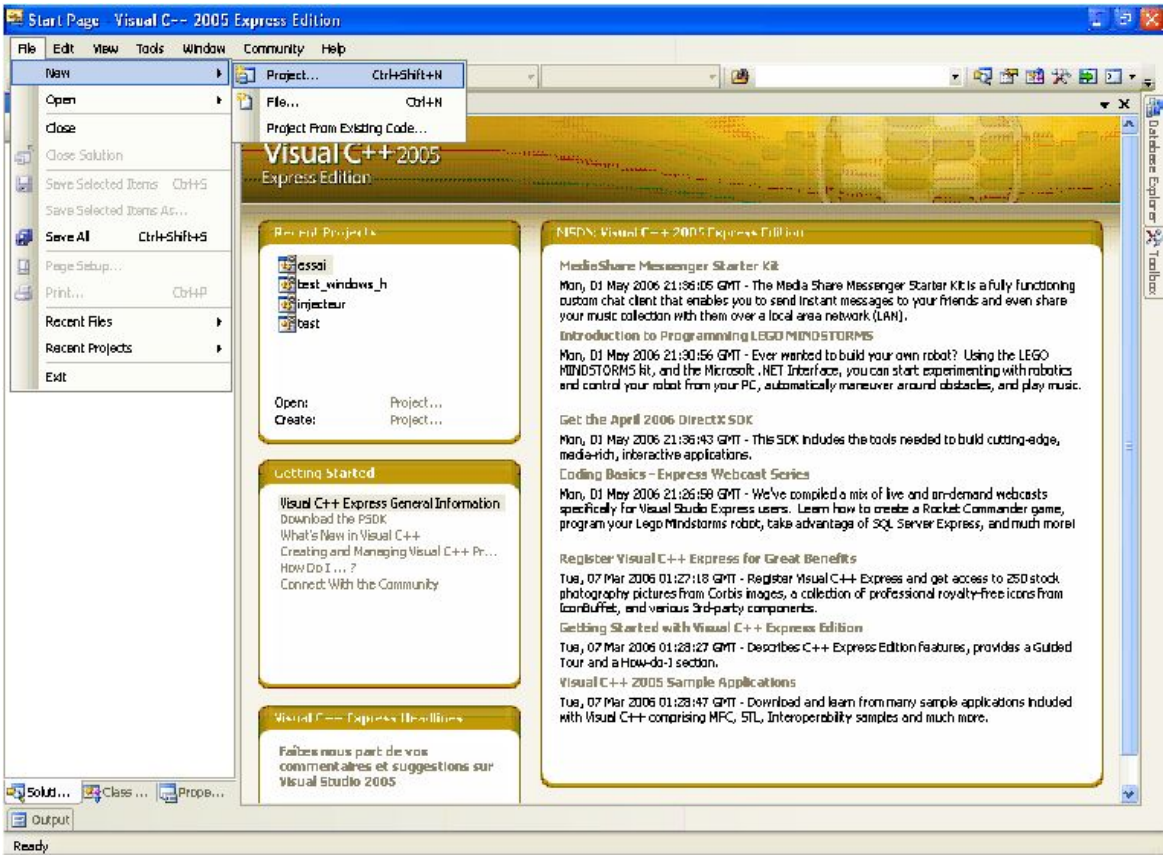


Utiliser Visual C++ sous Windows

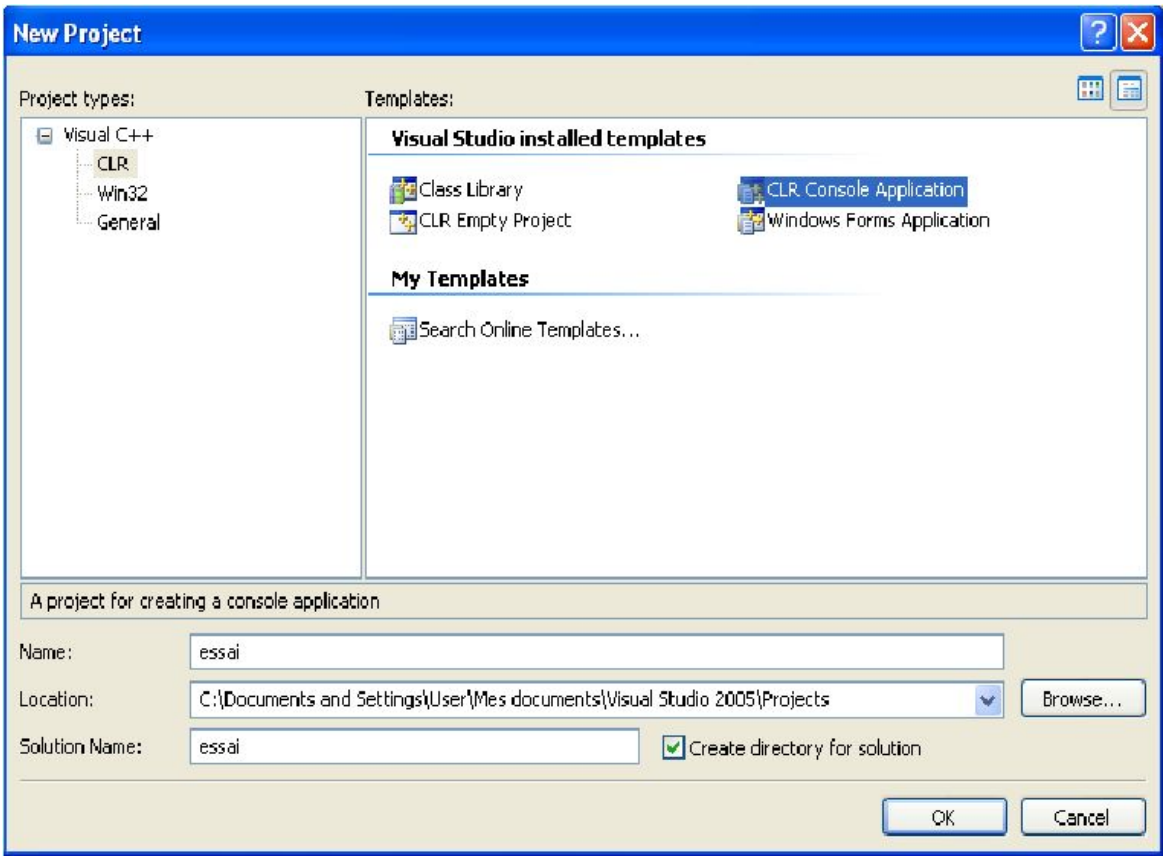
- Lorsque vous lancez Visual c++,vous obtenez cet écran



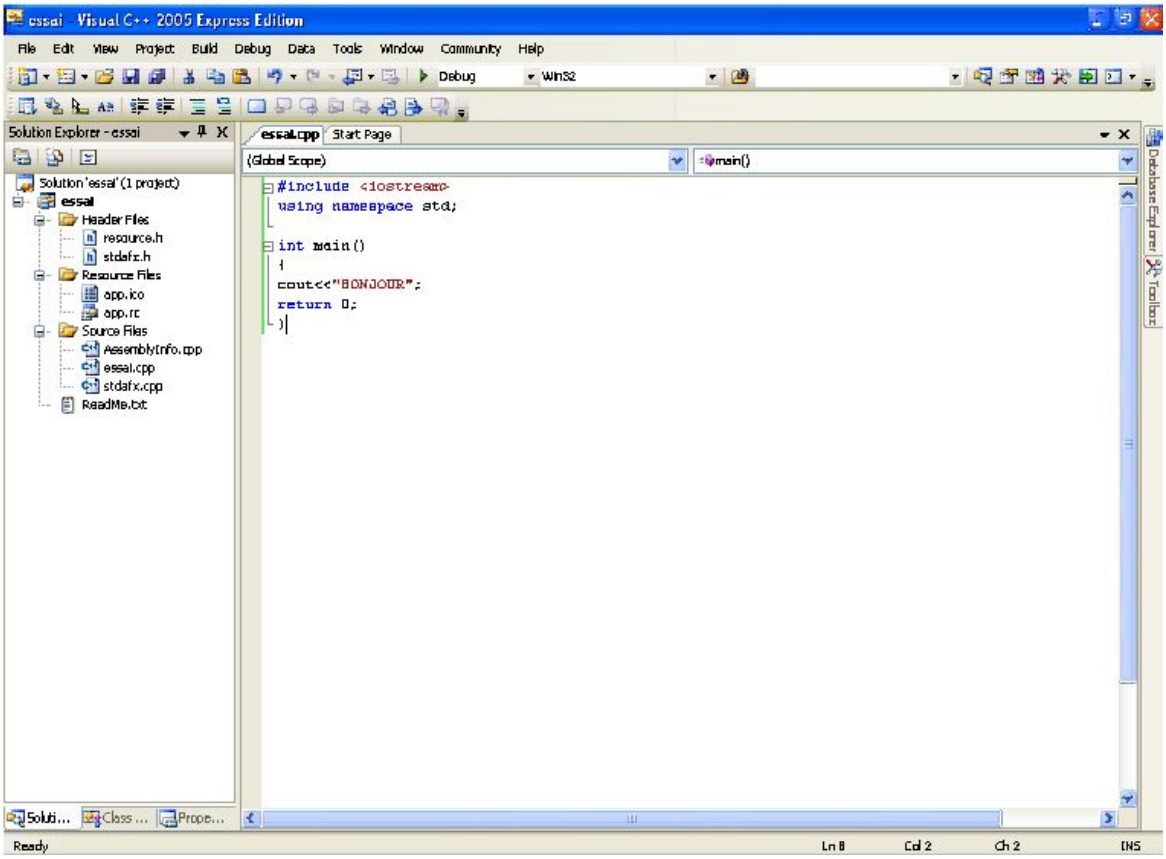
- Pour créer un nouveau projet il faut cliquer dans fichier->nouveau



- Puis sur Clr Console application,ensuite donnez un nom à votre projet dans le champ name



- Tapez votre code



- Et appuyez sur F5 ,votre application se lance

'À compléter'

Utiliser MinGW/MSys/Cmake sous windows

Si vous projetez de programmer à partir de divers libraires c++ issus de projets tiers, il est fortement déconseillé d'utiliser dev-cpp ou visual c++. Les outils MinGW, MSys et Cmake sont plus efficaces et souples.

Installer g++

g++ est généralement présent dans les paquets de votre distribution. Cependant, il n'est pas évident qu'il soit installé par défaut.

Sous Debian: **apt-get install g++**

Sous d'autres : utilisez votre gestionnaire de paquets (yum, urpmi, rpmdrake, synaptic...)

Éditer le code

L'édition d'un fichier se fait avec un éditeur de texte : on pourra citer emacs, vi et ses variantes, et plein d'autres.

Il faut sauvegarder le fichier par exemple sous le nom "main.cpp".

Compilation

Depuis un terminal, on se place dans le répertoire contenant notre fichier à compiler.

cd ./monjolicode

puis la commande la plus simple pour compiler est:

g++ main.cpp

qui génèrera un fichier nommé ./a.out si tout se passe bien (ie si votre code ne génère pas d'erreurs de compilation).

Pour générer un fichier portant un nom bien précis, on peut utiliser les commandes suivantes :

- sous Unix/Linux : **g++ main.cpp -o main**
- sous Windows : **g++ main.cpp -o main.exe**

La première commande génèrera un fichier nommé "main", et la deuxième un fichier "main.exe".

Il est vivement conseillé de regarder le manuel de g++. On pourra notamment être intéressé par les options de compilation comme l'optimisation (gain généralement important d'efficacité).

à compléter

Exercices du chapitre 1

1) En utilisant l'environnement de développement de votre choix, écrire, compiler et exécuter un programme qui affiche bonjour à l'écran.