



[Compléments C/C++]

Fabrice LE BARS

■ Sommaire

- Passage C/C++
- C
- C++
- Différences Windows/Linux
- Qt Creator
 - Créer un projet (non CMake) C simple avec Qt Creator
 - Opening and debugging a CMake project using Qt Creator
- Visual Studio
- Visual Studio Code
- Utilisation de code existant trouvé sur Internet
- Utilisation de bibliothèques de fonctions externes



Passage C/C++

Passage C/C++

- Le **C** est **inclus** (à 99%, e.g. le C++ est parfois plus strict sur les cast...) **dans** le **C++** : quand on fait du C, on fait aussi du C++ mais l'inverse n'est pas forcément vrai
- Du code **C** ou **C++** peut être écrit dans un fichier **.cpp** mais seul du code **C** peut être écrit dans un fichier **.c**
- Dans un **.h**, on peut écrire du C ou du C++, mais il faut que les **.c** n'incluent que des **.h** avec du C (utiliser **#ifdef __cplusplus, extern C**, etc. si nécessaire...)
- Parfois on trouve aussi des **.hpp**, **.hxx** (**.h** avec du **C++**), **.cxx** (comme **.cpp**)...

Passage C/C++

- Le C++ rajoute des notions de programmation orientée objet (classe, héritage, polymorphisme) ainsi que des facilités d'écriture (surcharge d'opérateurs tels que =, +, -, *, /, >>, [], passage de paramètres par référence, etc.)
- Certains compilateurs (e.g. Visual Studio) en mode C forcent les déclarations en début de bloc, en C++ ils ne le forcent plus...



Autres apports du C++ par rapport au C

- **new**, **delete** et **new[]**, **delete[]** en C++ à la place (ou en plus) de **malloc()** et **free()** en C pour utiliser des pointeurs et tableaux à taille variable
- Mécanisme d'exceptions
- Templates
- Classes **vector**, **deque**, **list**...
- Nouveaux ajouts en C++11, C++17, C++20, etc.
 - A noter que les différents compilateurs ne supportent pas toujours tous les éléments proposés dans les différentes versions du langage
 - Même si des ajouts sont proposés pour C11, etc., en pratique les compilateurs C ne les supportent pas forcément probablement pour maximiser la compatibilité entre les versions de compilateurs C...

Correspondances C - C++

- **Comparer** <https://docs.microsoft.com/en-us/cpp/c-runtime-library/run-time-routines-by-category> et <https://docs.microsoft.com/en-us/cpp/standard-library/cpp-standard-library-header-files...>
- Chaines de caractères
 - En **C**, les **chaines de caractères** sont typiquement définies avec **char str[] = "test"**
Ce sont juste des **tableaux de char** (entiers signés 8 bits) qui, pour pouvoir être manipulés facilement par des fonctions, sont supposés **terminés par un caractère ASCII de code 0** (0 sous forme d'**entier**, ou **'\0'** sous forme de **caractère**). La définition de **str** faite précédemment avec une valeur entre guillemets va faire que **str[0] == 't', str[1] == 'e', str[2] == 's', str[3] == 't', str[4] == '\0' == 0, sizeof(str) == 5, strlen(str) == 4...**

Attention : la notation entre **guillemets double rajoute** automatiquement un **caractère 0** à la fin, mais d'autres manières de définir une chaîne de caractères peuvent ne pas automatiquement l'inclure, il faut donc **surveiller cela**, sinon des fonctions de manipulation comme **strlen()**, **strcpy()**, etc. ne seront pas capables de trouver quand s'arrête la chaîne de caractères et vont donc parcourir toute la mémoire jusqu'à trouver un 0 ou jusqu'à ce qu'il y ait une violation d'accès mémoire...

Correspondances C - C++

- Chaines de caractères
 - C : printf(), scanf(), getchar(), gets(), sprintf(), sscanf(), strlen(), strcpy(), strcat(), strcmp(), atof()
 - C++ : string, cout, cin
- Manipulation de la mémoire
 - C : malloc, calloc(), realloc(), free(), memset(), memcpy()
 - C++ : new, delete, new[], delete[]
- Fichiers
 - C : fopen(), fclose(), fread(), fwrite(), fgets(), fprintf(), feof()
 - C++ : fstream

Pièges divers en C/C++

- **sizeof(d'une struct) != sum(sizeof(des éléments de la struct))** à cause de problèmes de **struct packing**, **alignment**, etc.
- **Attention** à l'utilisation de **sizeof()** pour des tableaux et pointeurs (verifier si fait bien ce qu'on veut)...





ENSTA
Bretagne

C



Hello World!

```
#include <stdio.h>

int main()
{
    printf("Hello\n");
    return 0;
}
```



C

- Voir les exemples de code : <http://www.ensta-bretagne.fr/lebars/CTuto.zip>
- C Standard Library : see e.g. <http://www.cplusplus.com/reference/clibrary/> , <https://docs.microsoft.com/en-us/cpp/c-runtime-library/run-time-routines-by-category> , etc.
- Pour trouver rapidement la description officielle de la plupart des fonctions, types, constantes, etc. spécifiques aux OS, rechercher sur Google :
 - Linux : `man function_name`
 - Windows : `MSDN function_name`

C++

Hello World!

Notion de namespace

```
#include <iostream>

int main(int argc, char* argv[])
{
    std::cout << "Hello\n";

    return 0;
}
```

Opérateur

Namespace

Plus besoin d'utiliser std::

```
#include <iostream>

using namespace std;

int main(int argc, char* argv[])
{
    cout << "Hello\n";

    return 0;
}
```



Références

```
#include <iostream>

void functionWithRef(int& value)
{
    value = 2;
}

int main(int argc, char* argv[])
{
    int a = 1;

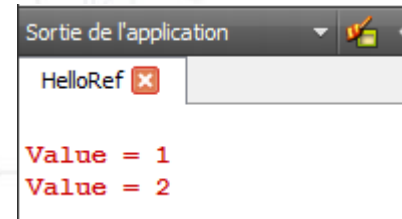
    std::cout << "Value = " << a << "\n";

    functionWithRef(a);

    std::cout << "Value = " << a << "\n";

    return 0;
}
```

Paramètre passé par référence



Classes

```
#ifndef TESTCLASS_H  
#define TESTCLASS_H
```

```
class TestClass  
{  
private:  
    double value;  
public:  
    // Member functions (EX : getters and setters)  
    double getValue() const;  
    void setValue(double);  
};  
#endif // TESTCLASS_H
```

TestClass.h

Compléments C/C++

```
#include "TestClass.h"
```

```
// Member functions (EX : getters and setters)  
double TestClass::getValue() const  
{  
    return value;  
}  
void TestClass::setValue(double value)  
{  
    this->value = value;  
}
```

TestClass.cpp

```
#include <iostream>  
#include "TestClass.h"  
  
using namespace std;  
  
int main(int argc, char* argv[])  
{  
    TestClass tst;  
  
    tst.setValue(2.1);  
  
    cout << tst.getValue() << endl;  
  
    return 0;  
}
```

Main.cpp

Constructeurs, destructeur et surcharge

```
#ifndef TESTCLASS_H
#define TESTCLASS_H

#include <iostream>

class TestClass
{
private:
    double value;
public:
    // Constructors
    TestClass();
    TestClass(const TestClass&);
    TestClass(const double&);

    // Destructor
    ~TestClass();
};

#endif // TESTCLASS_H
```

TestClass.h

```
#include "TestClass.h"

using namespace std;

TestClass::TestClass()
{
    value = 0;
}

TestClass::TestClass(const TestClass& a)
{
    value = a.value;
}

TestClass::TestClass(const double& value)
{
    this->value = value;
}

// Destructor
TestClass::~~TestClass()
{
    value = 0;
}
```

TestClass.cpp

Liste d'initialisation

```
TestClass::TestClass(const TestClass& a) :  
    value(a.value),  
    tmpValue(1)  
    // Initialization list  
{  
  
}
```



L'initialisation des membres dans le constructeur peut être faite de cette façon



Opérateurs

```
#ifndef TESTCLASS_H
#define TESTCLASS_H

class TestClass
{
private:
    double value;
public:
    // Member functions (EX : getters and setters)
    double getValue() const;
    void setValue(double);

    // Member operators (EX : unary operator =)
    TestClass& operator=(const TestClass&);
};

#endif // TESTCLASS_H
```

```
#include "TestClass.h"

// Member functions (EX : getters and setters)
double TestClass::getValue() const
{
    return value;
}

void TestClass::setValue(double value)
{
    this->value = value;
}

// Member operators (EX : unary operator =)
TestClass& TestClass::operator=(const TestClass& a)
{
    value = a.value+10;

    return *this;
}
```



TestClass.h

TestClass.cpp

Opérateurs

```
#include <iostream>
#include "TestClass.h"

using namespace std;

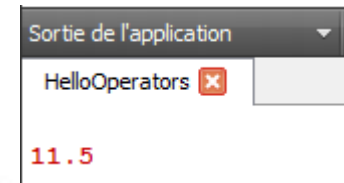
int main(int argc, char* argv[])
{
    TestClass tst1;
    TestClass tst2;

    tst2.setValue(1.5);

    tst1 = tst2;

    cout << tst1.getValue() << endl;

    return 0;
}
```



Résultat

Main.cpp

Fonctions amies

```
#ifndef TESTCLASS_H  
#define TESTCLASS_H
```

```
#include <iostream>
```

```
class TestClass
```

```
{
```

```
private:
```

```
    double value;
```

```
public:
```

```
    // Member functions (EX : getters and setters)
```

```
    double getValue() const;
```

```
    void setValue(double);
```

```
    // Friend functions
```

```
    friend TestClass compute(const TestClass&, const TestClass&);
```

```
};
```

```
#endif // TESTCLASS_H
```

TestClass.h

Fonctions amies

```
#include "TestClass.h"

// Member functions (EX : getters and setters)
double TestClass::getValue() const
{
    return value;
}

void TestClass::setValue(double value)
{
    this->value = value;
}

// Friend functions
TestClass compute(const TestClass& a, const TestClass& b)
{
    TestClass c;

    c.setValue(a.value + 2* b.value);

    return c;
}
```

TestClass.cpp

Fonctions amies

```
#include "TestClass.h"

using namespace std;

int main(int argc, char* argv[])
{
    TestClass tst1;
    TestClass tst2;

    tst1.setValue(1);
    tst2.setValue(-2);

    TestClass tst = compute(tst1, tst2);

    cout << tst.getValue() << endl;

    return 0;
}
```

Main.cpp



Héritage

```
// constructors and derived classes
#include <iostream>
using namespace std;

class mother {
public:
    mother ()
        { cout << "mother: no parameters\n"; }
    mother (int a)
        { cout << "mother: int parameter\n"; }
};

class daughter : public mother {
public:
    daughter (int a)
        { cout << "daughter: int parameter\n\n"; }
};

class son : public mother {
public:
    son (int a) : mother (a)
        { cout << "son: int parameter\n\n"; }
};

int main () {
    daughter cynthia (0);
    son daniel(0);

    return 0;
}
```

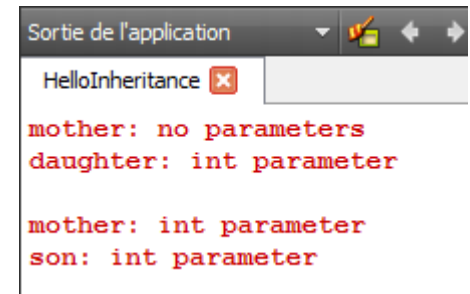
Classe de base

Classe dérivées

Appel à un constructeur spécifique de la classe de base

Attention : un appel d'un constructeur de la classe de base ailleurs que dans la liste d'initialisation ne crée qu'une nouvelle instance de ce type, indépendante de la classe dérivée

Destructeur : celui de la classe de base sera toujours appelé automatiquement après celui de la classe dérivée



```
Sortie de l'application
HelloInheritance
mother: no parameters
daughter: int parameter
mother: int parameter
son: int parameter
```

Classes abstraites

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

class CPolygon {
protected:
    int area;
public:
    virtual int computeArea(void) = 0;
};

class CRectangle : virtual public CPolygon {
private:
    int width, height;
public:
    CRectangle(int w, int h) : width(w), height(h) {}
    int computeArea(void) {
        area = width * height;
        return area;
    }
};

class CSquare : virtual public CPolygon {
private:
    int width;
public:
    CSquare(int w) : width(w) {}
    int computeArea(void) {
        area = width * width;
        return area;
    }
};
```

Classe de base abstraite

Classes dérivées

Classes abstraites

```
int main() {  
    CRectangle rect = CRectangle(3,4);  
    CSquare square = CSquare(2);  
    CPolygon * pPoly1 = &rect;  
    CPolygon * pPoly2 = &square;  
    cout << pPoly1->computeArea() << endl;  
    cout << pPoly2->computeArea() << endl;  
    return 0;  
}
```



- Voir les exemples de code : <http://www.ensta-bretagne.fr/lebars/C++Tuto.zip>
- C++ Standard Library : voir e.g. <https://docs.microsoft.com/en-us/cpp/cpp/cpp-language-reference> , <https://gcc.gnu.org/onlinedocs/gcc-11.2.0/libstdc++/manual/>
- <https://github.com/isocpp/CppCoreGuidelines/>





Différences Windows/Linux

Différences Windows/Linux

- Différences entre les compilateurs Windows et Linux

- **Linux**

Le compilateur **C** le plus utilisé est **GCC** (commande **gcc**)

Son équivalent **C++** est **G++** (commande **g++**)

- **Windows**

GCC/G++ existent avec **MinGW**, Cygwin, Windows Subsystem for Linux (ces 2 derniers sont un peu particuliers...)

Différents **IDE** (Integrated Development Environment) existent et fournissent souvent leurs **propres compilateurs** :

Microsoft Visual Studio avec **CL**

Borland C++ Builder / Turbo C++ / Borland Developer
Studio avec **BCC32**

Qt Creator, Eclipse, Code Blocks / Dev-C++ avec **MinGW**

Différences Windows/Linux

- Différences entre les compilateurs Windows et Linux
 - Certains **IDE** peuvent aussi être configurés pour utiliser **plusieurs compilateurs différents**, e.g. sous Windows **Qt Creator** supporte GCC/G++ de MinGW et CL de Visual Studio. Ceci est aussi souvent possible à travers leur support de CMake
 - Chaque **compilateur** a sa **propre version** de la **C et C++ Standard Library**, par contre la plupart des autres fonctions systèmes ou d'autres bibliothèques dépendent a priori de l'OS (même si en fait il y a aussi souvent quelques petites adaptations selon le compilateur détecté via directives de précompilation)...
 - e.g. sous Windows, si on utilise le GCC 13.2.0 de MinGW la C++ Standard Library est celle de <https://gcc.gnu.org/onlinedocs/gcc-13.2.0/libstdc++/manual/>, pour le CL de Visual Studio 2022 c'est plutôt celle de <https://docs.microsoft.com/en-us/cpp/standard-library/cpp-standard-library-header-files?view=vs-2022>, par contre les fonctions systèmes de Windows sont pour les 2 celles de <https://docs.microsoft.com/en-us/windows/win32/apiindex/windows-api-list> (en fait dans les détails ce n'est pas forcément non plus exactement la même version, mais c'est la documentation principale...)

Différences Windows/Linux

- Différences entre les compilateurs Windows et Linux
 - Dans un **IDE** il y a souvent :
 - Un concept de **projet** (enregistré parfois dans un format de fichier spécifique indépendant du compilateur mais propre à l'IDE)
 - Un concept de **configuration** (e.g. **Debug**, **Release**, chacune utilisant en général des options de compilation différentes)
 - On a en général le choix entre une exécution du programme compilé à travers un **debugger** ou sans debugger que le projet soit compilé en configuration **Debug** ou **Release**
 - Le bouton d'**exécution du programme** est souvent un **triangle vert** (avec des options autour permettant de choisir si on passe par un debugger, la configuration, etc.), voir les captures d'écran des différents IDE dans la suite...

- Différences entre les compilateurs Windows et Linux

Equivalences Linux / Windows		
	Linux/GCC	Windows/Visual C++
Fichiers objets	.o	.obj
Bibliothèque statique	.a	.lib
Bibliothèque dynamique	.so	.dll
Exécutable		.exe

Différences Windows/Linux

■ Outils de compilation de projets C/C++ sous Linux

- **Makefile** : langage dédié aux lancements de commandes de compilation de fichiers en prenant en compte notamment la date de modification des fichiers sources pour éviter de recompiler les fichiers qui n'auraient pas été modifiés (e.g. joue le même rôle qu'un fichier projet **.vcxproj** de Visual Studio)
- **autotools** : outils pour vérifier la disponibilité d'options spécifiques du compilateur, de certaines bibliothèques ou fonctions pas toujours disponibles, génère typiquement un fichier **config.h** avec des **#define HAVE_XXX** pour indiquer la présence ou non de quelque chose et génère un fichier **Makefile**.

Utilisation classique :

```
./configure && make && sudo make install
```

autotools est de plus en plus remplacé par **CMake**

Différences Windows/Linux

■ CMake

- Pour la compilation de projets impliquant plusieurs fichiers C/C++ idéalement compatibles Windows, Linux, macOS
- CMake ne fournit pas de compilateur, il faut en installer un par ailleurs, son but est de permettre la compilation d'un projet C/C++ avec idéalement les mêmes commandes quel que soit le compilateur et l'OS
- Un fichier **CMakeLists.txt** doit être préparé, voir e.g. http://simon-rohou.fr/cours/c++/doc/05/cpp_05_cm_developpement.pdf et <https://www.ensta-bretagne.fr/lebars/C++Tuto.zip>
- Utilisation la plus simple et portable :

cmake .

cmake --build .

Il reste alors à exécuter le programme compilé...

Différences Windows/Linux

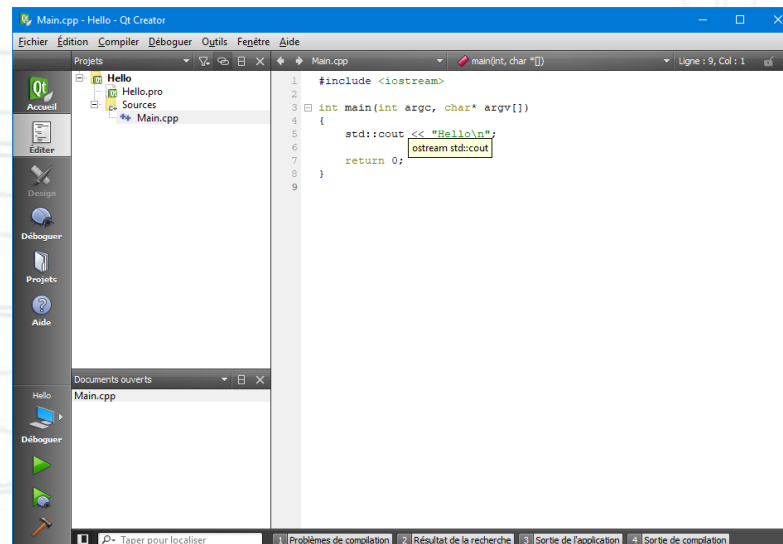
■ CMake

- En cas de problème, supprimer les fichiers générés par CMake, notamment **CMakeCache.txt**
- Problèmes courants : CMake peut ne pas trouver le compilateur qu'on souhaite utiliser, utiliser l'option **-G** pour spécifier explicitement le « Generator » souhaité, e.g. **cmake -G "MinGW Makefiles"** . pour Windows si MinGW est installé, essayer de supprimer le CMakeCache.txt et en tester d'autres si ne marche pas (utiliser **cmake-gui** peut être pratique pour voir les différentes options), vérifier aussi l'installation du compilateur souhaité (idéalement l'installer dans les emplacements par défaut), le rajouter dans le PATH si besoin, etc.
- Avancé : guide de bonnes pratiques : <https://cliutils.gitlab.io/modern-cmake/>

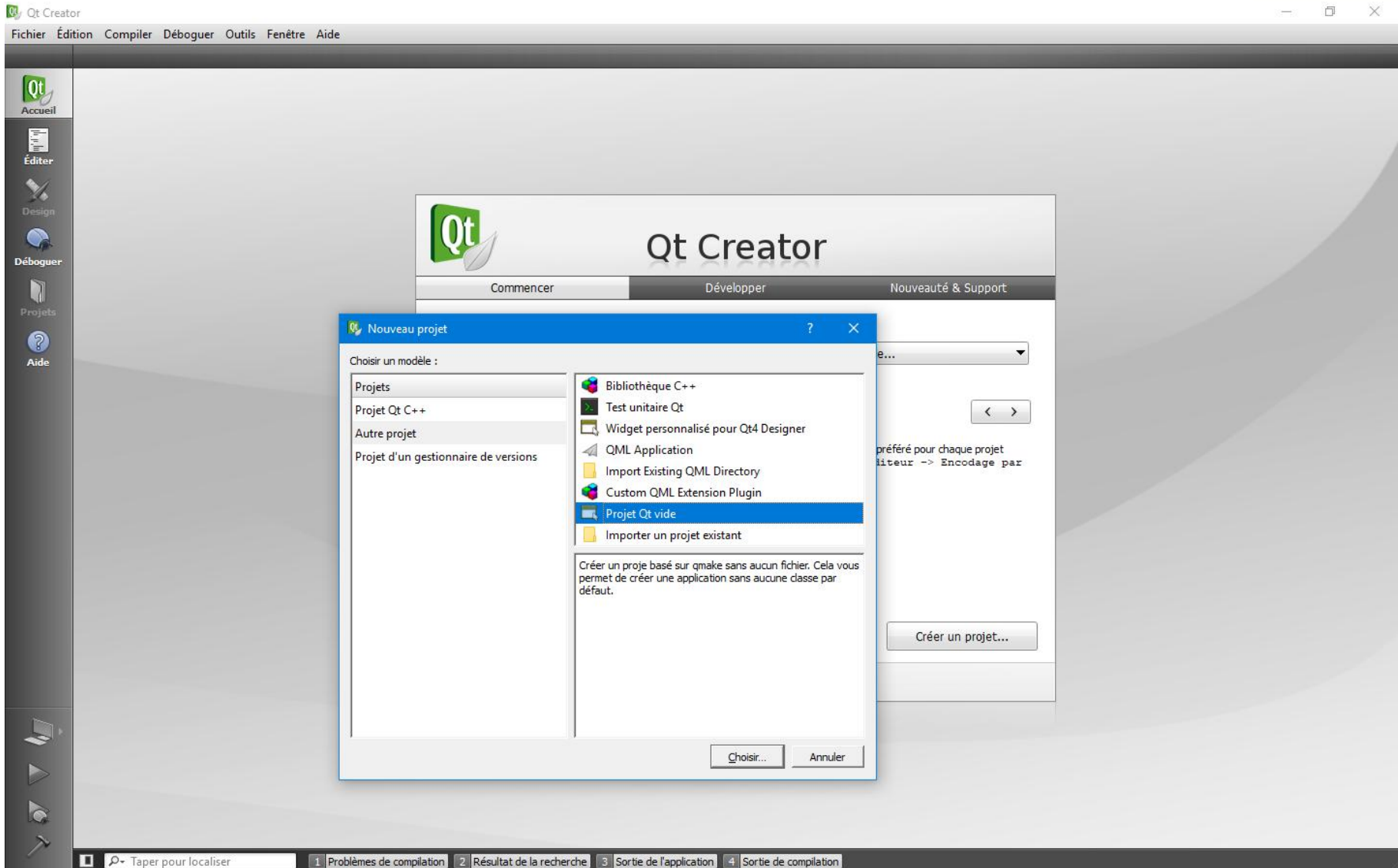


Qt Creator

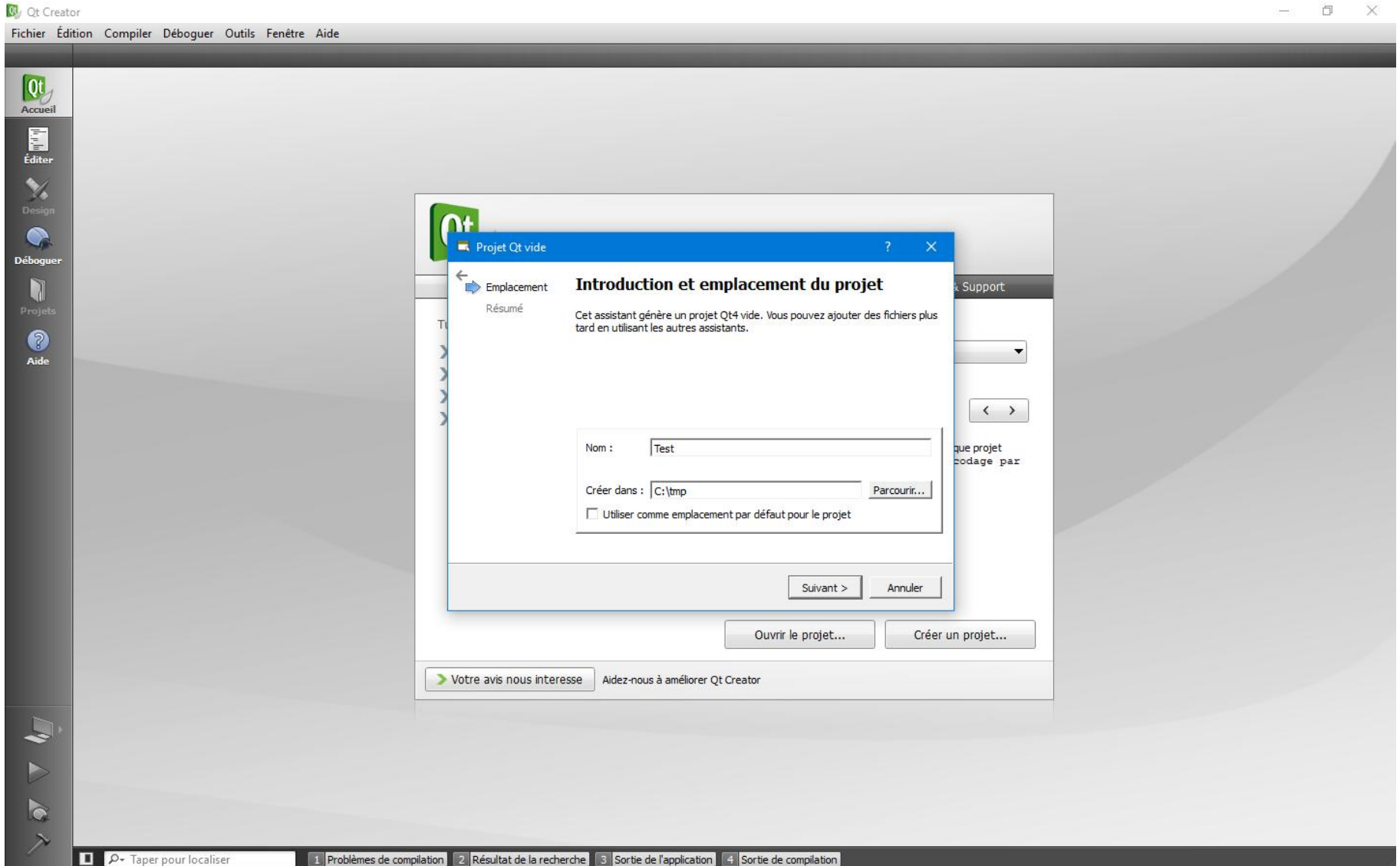
- IDE (Integrated Development Environment) portable :
 - Windows, Linux, macOS
 - Peut utiliser le compilateur de Visual Studio ou le GCC de MinGW sous Windows
 - Fichier **projet** (regroupement de .c, .cpp, .h...) : **.pro**
 - Fichier texte **éditable** manuellement (e.g. pour pouvoir rajouter bibliothèques et options spécifiques)
 - Un fichier **CMakeLists.txt** peut aussi être utilisé en tant que projet **CMake**
 - Installation : voir e.g. https://www.ensta-bretagne.fr/lebars/Share/setup_qt_opencv.pdf



Créer un projet (non CMake) C simple avec Qt Creator



Créer un projet (non CMake) C simple avec Qt Creator



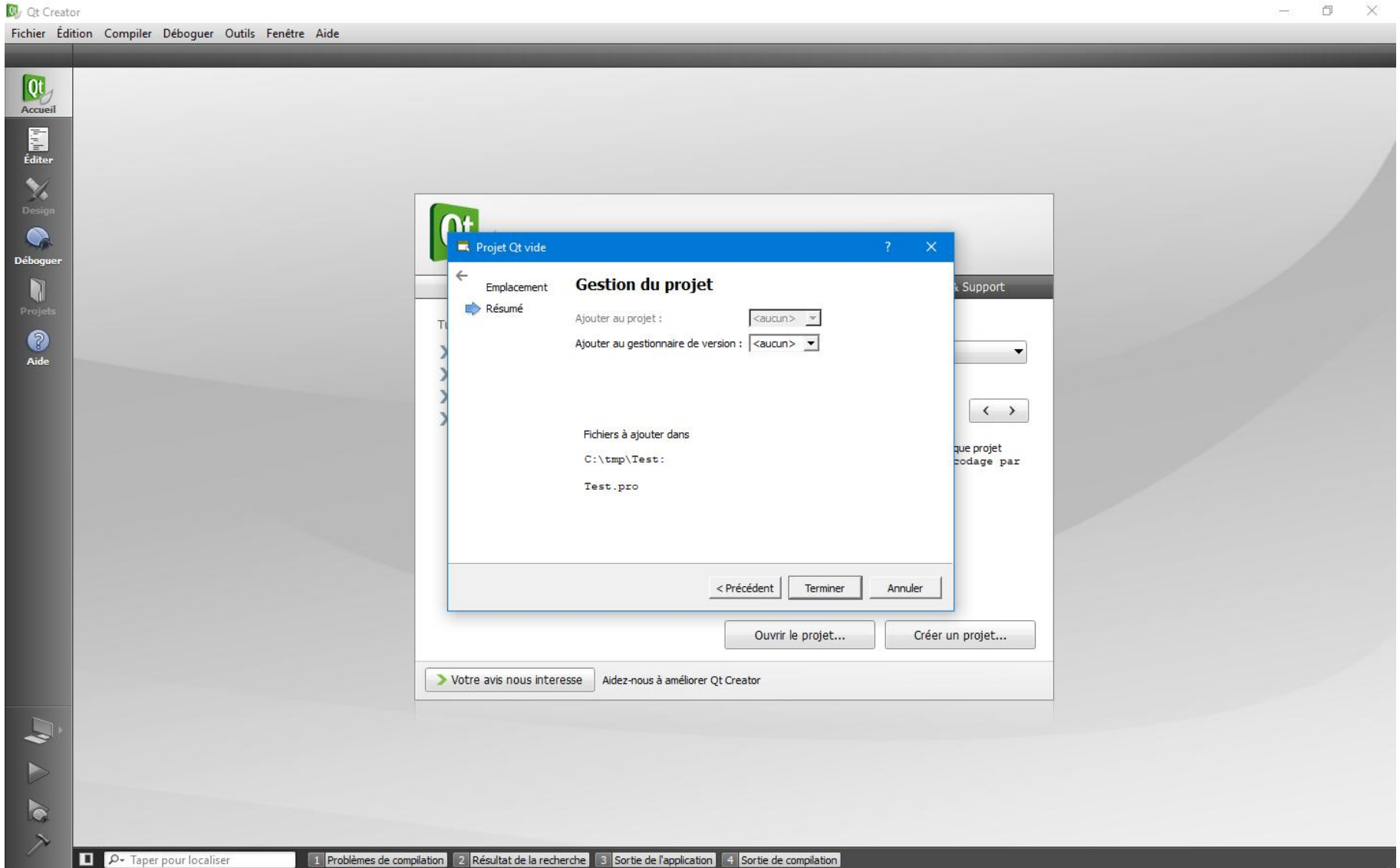
The screenshot shows the Qt Creator interface with a dialog box titled "Projet Qt vide" (Empty Qt Project) open. The dialog has a "Résumé" (Summary) tab selected. The text in the dialog reads: "Cet assistant génère un projet Qt4 vide. Vous pouvez ajouter des fichiers plus tard en utilisant les autres assistants." (This wizard generates an empty Qt4 project. You can add files later using the other wizards.)

The dialog contains the following fields and options:

- Nom : Test
- Créer dans : C:\tmp (with a "Parcourir..." button)
- Utiliser comme emplacement par défaut pour le projet

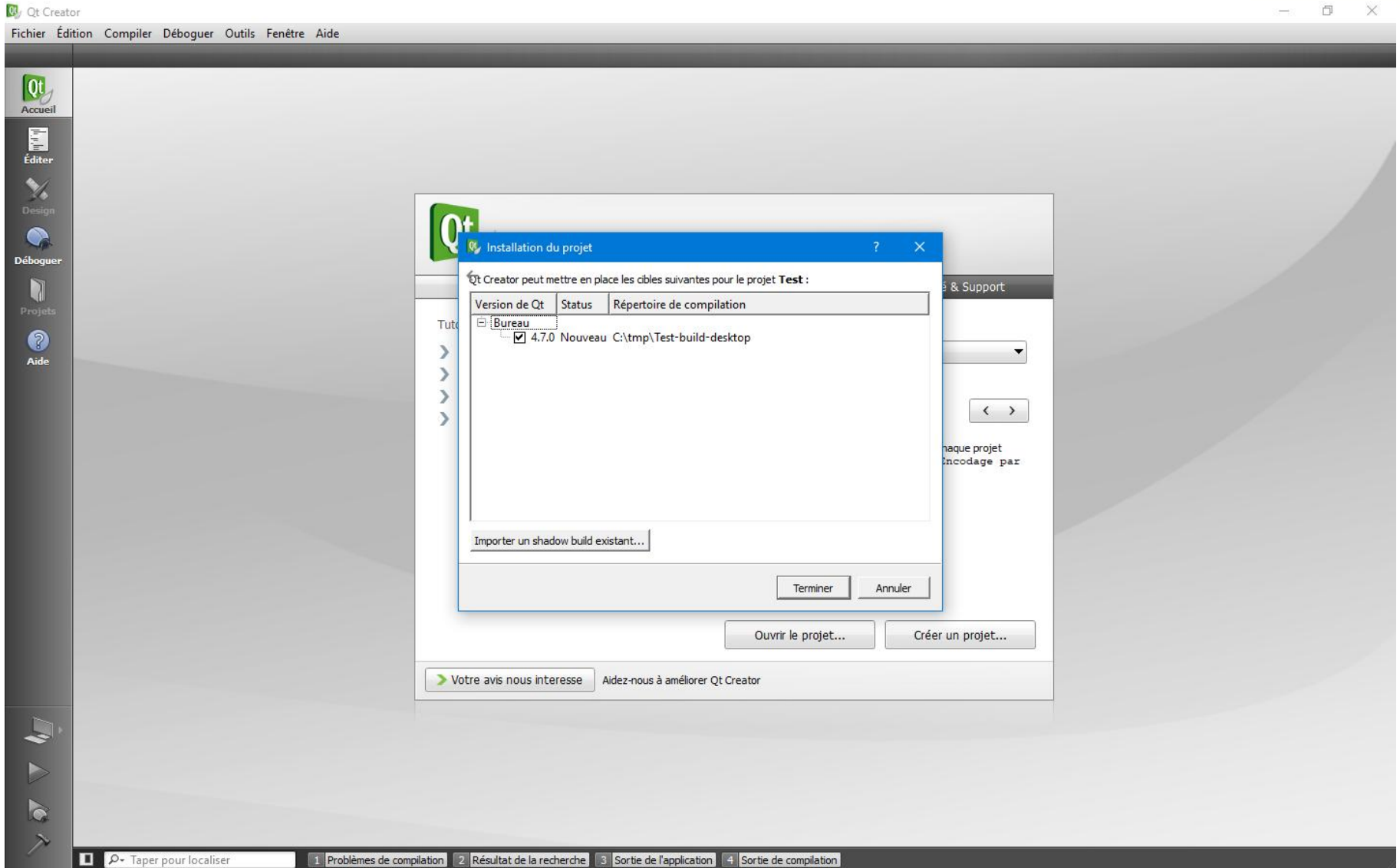
At the bottom of the dialog are "Suivant >" and "Annuler" buttons. Below the dialog, in the main window, are "Ouvrir le projet..." and "Créer un projet..." buttons. At the very bottom of the Qt Creator window, there is a search bar and a list of tabs: "1 Problèmes de compilation", "2 Résultat de la recherche", "3 Sortie de l'application", and "4 Sortie de compilation".

Créer un projet (non CMake) C simple avec Qt Creator



The screenshot shows the Qt Creator interface with the 'Gestion du projet' dialog box open. The dialog has a blue title bar 'Projet Qt vide' and a left sidebar with 'Emplacement' and 'Résumé' tabs. The 'Résumé' tab is active, showing two dropdown menus for 'Ajouter au projet' and 'Ajouter au gestionnaire de version', both set to '<aucun>'. Below these is a section 'Fichiers à ajouter dans' with a list containing 'C:\tmp\Test:' and 'Test.pro'. At the bottom of the dialog are buttons for '< Précédent', 'Terminer', and 'Annuler'. In the background, the main Qt Creator window shows a menu bar (Fichier, Édition, Compiler, Déboguer, Outils, Fenêtre, Aide) and a sidebar with icons for Accueil, Éditer, Design, Déboguer, Projets, and Aide. At the bottom of the main window, there are buttons for 'Ouvrir le projet...' and 'Créer un projet...'. A footer bar at the very bottom contains a search field and a list of search results: '1 Problèmes de compilation', '2 Résultat de la recherche', '3 Sortie de l'application', and '4 Sortie de compilation'.

Créer un projet (non CMake) C simple avec Qt Creator



Qt Creator

Fichier Édition Compiler Déboguer Outils Fenêtre Aide

Qt Creator peut mettre en place les cibles suivantes pour le projet Test :

Version de Qt	Status	Répertoire de compilation
Bureau		
4.7.0	<input checked="" type="checkbox"/>	C:\tmp\Test-build-desktop

Importer un shadow build existant...

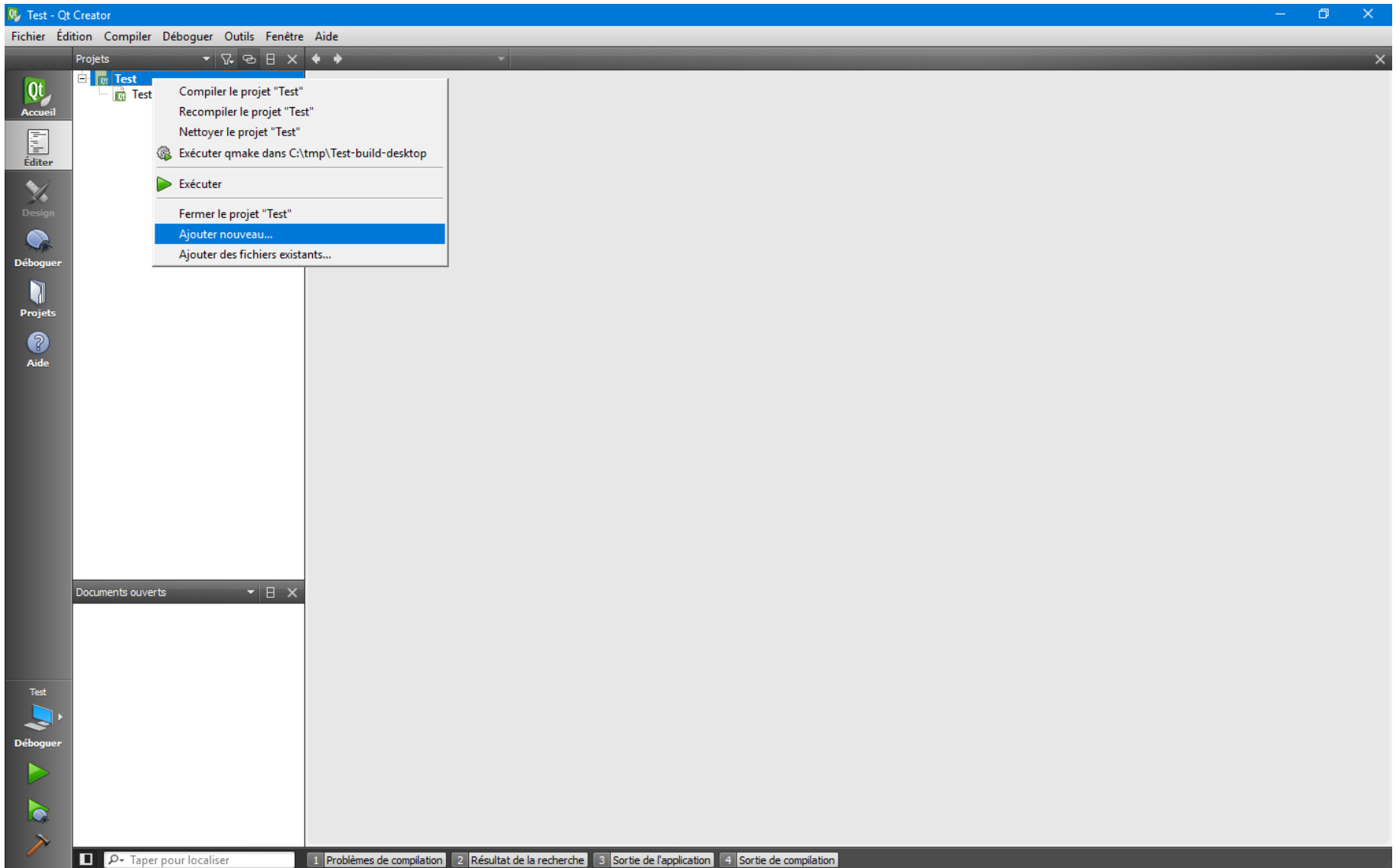
Terminer Annuler

Ouvrir le projet... Créer un projet...

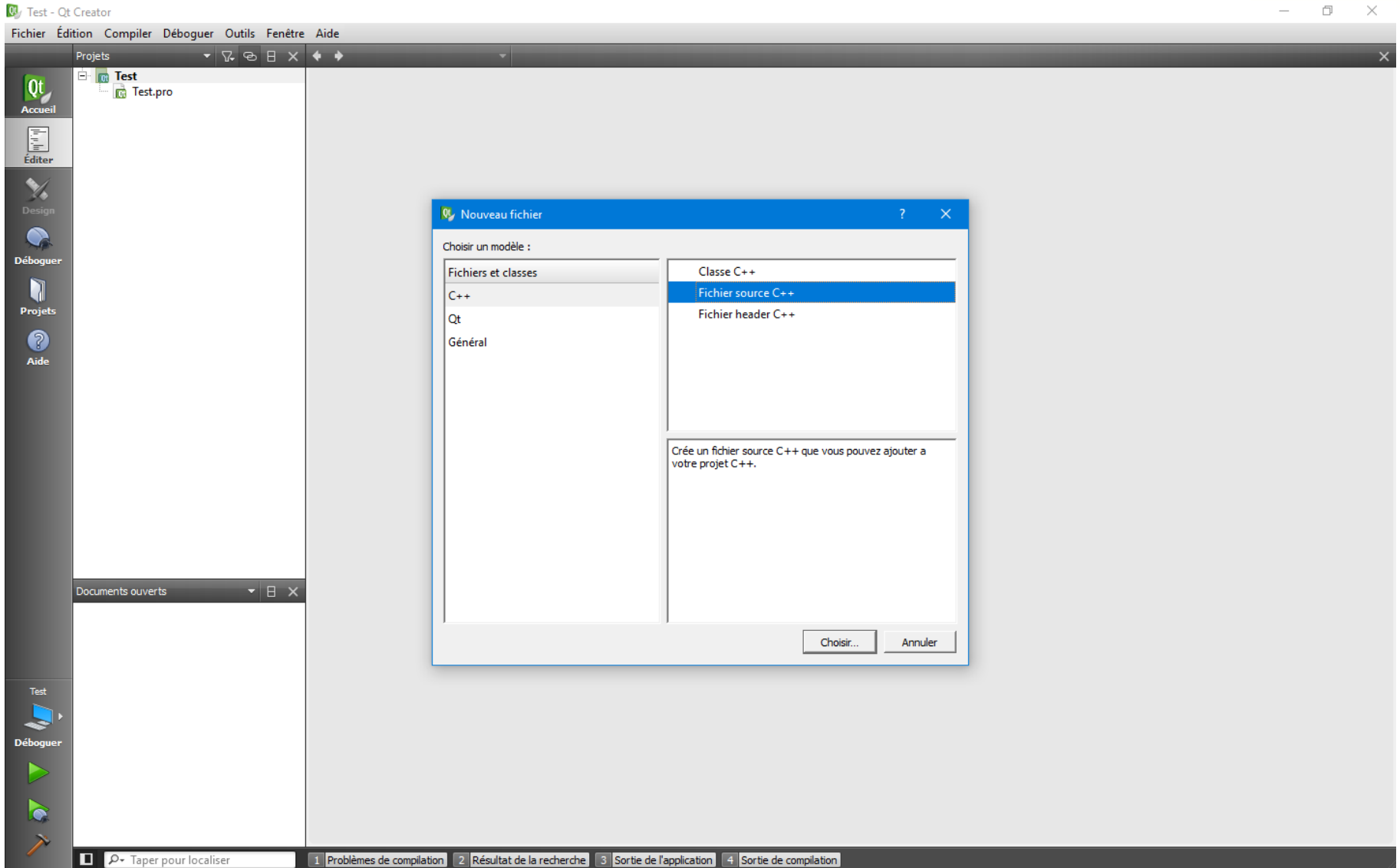
Votre avis nous interesse Aidez-nous à améliorer Qt Creator

1 Problèmes de compilation 2 Résultat de la recherche 3 Sortie de l'application 4 Sortie de compilation

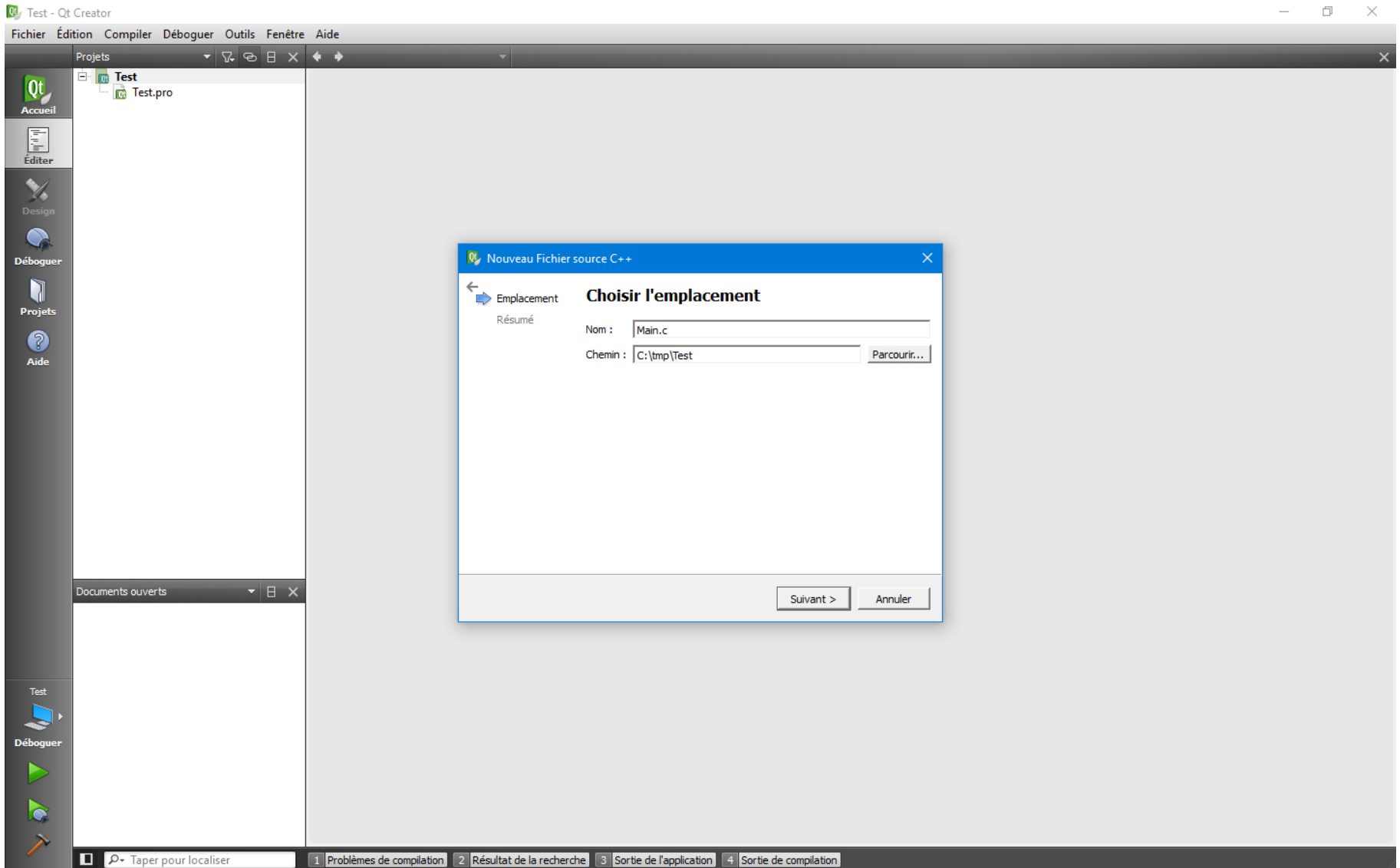
Créer un projet (non CMake) C simple avec Qt Creator



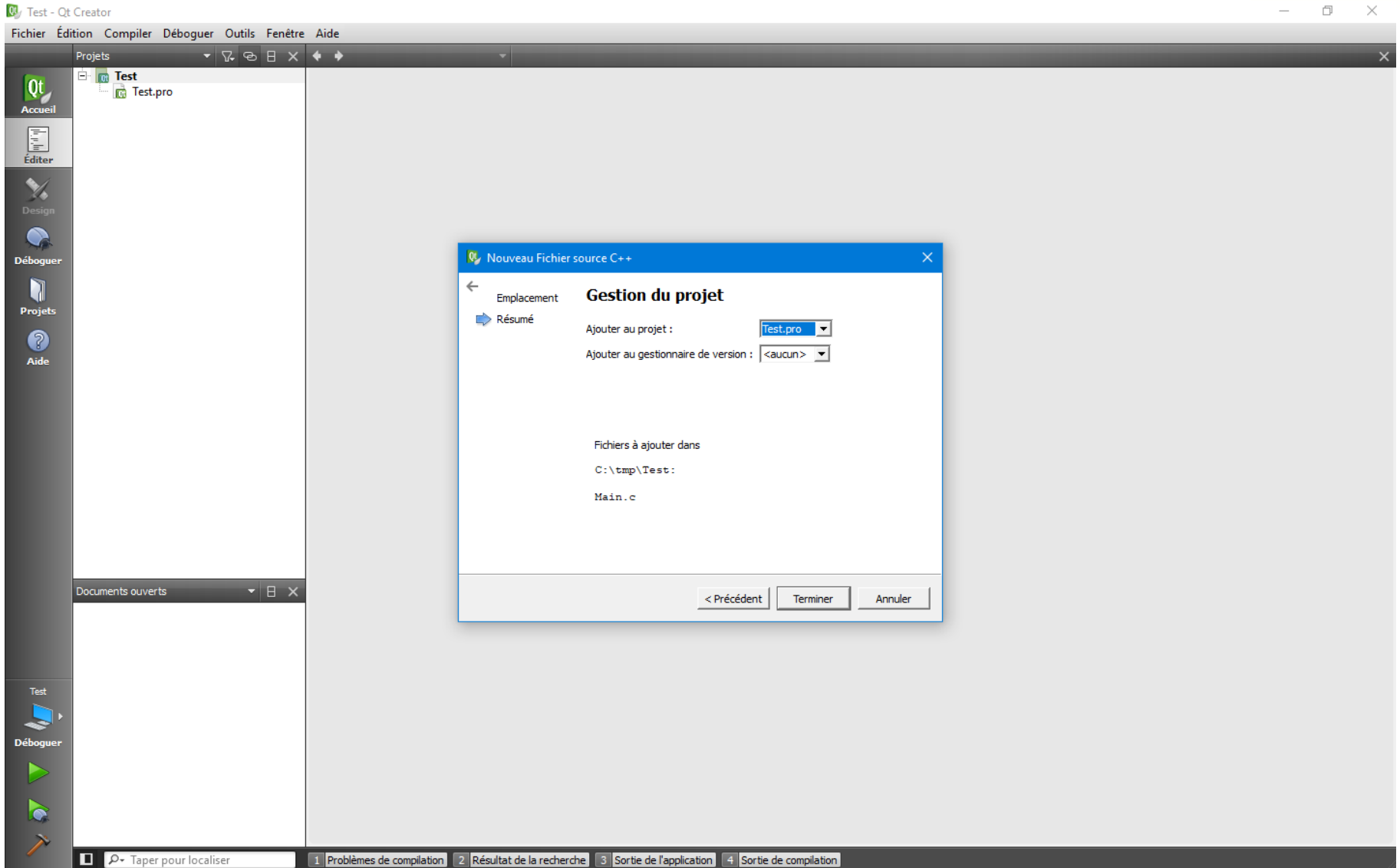
Créer un projet (non CMake) C simple avec Qt Creator



Créer un projet (non CMake) C simple avec Qt Creator



Créer un projet (non CMake) C simple avec Qt Creator

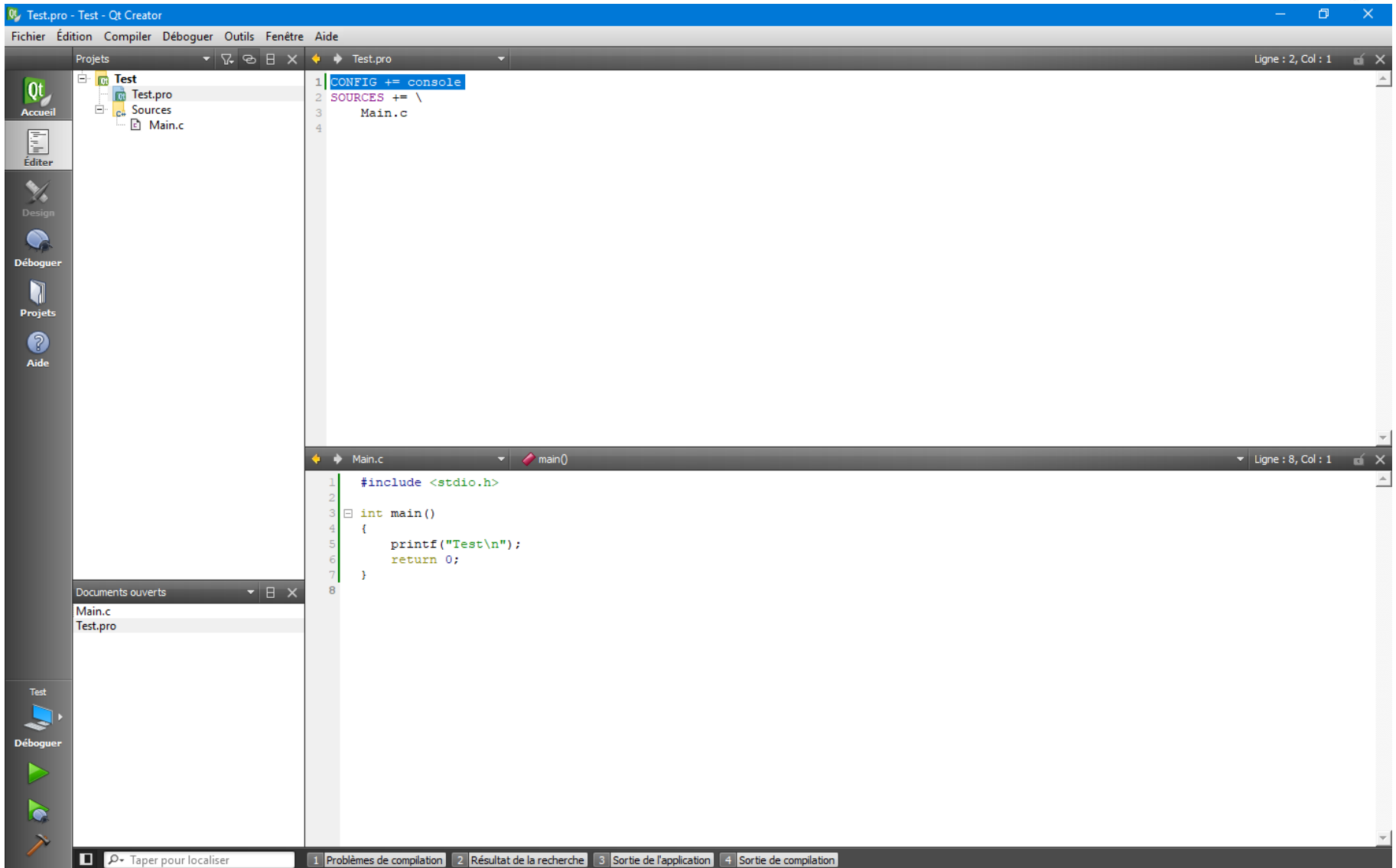


The screenshot shows the Qt Creator IDE interface. The main window is titled 'Test - Qt Creator' and displays a project named 'Test' with a file 'Test.pro'. A dialog box titled 'Nouveau Fichier source C++' is open, showing the 'Gestion du projet' (Project Management) tab. The dialog includes the following fields and options:

- Emplacement**: A back arrow icon.
- Résumé**: A right arrow icon.
- Ajouter au projet**: A dropdown menu with 'Test.pro' selected.
- Ajouter au gestionnaire de version**: A dropdown menu with '<aucun>' selected.
- Fichiers à ajouter dans**: A section with the following text:
 - C:\tmp\Test:
 - Main.c
- Buttons**: '< Précédent', 'Terminer', and 'Annuler'.

The status bar at the bottom of the IDE shows a search bar and four tabs: '1 Problèmes de compilation', '2 Résultat de la recherche', '3 Sortie de l'application', and '4 Sortie de compilation'.

Créer un projet (non CMake) C simple avec Qt Creator



The screenshot shows the Qt Creator IDE interface. The top menu bar includes "Fichier", "Édition", "Compiler", "Débuguer", "Outils", "Fenêtre", and "Aide". The "Projets" panel on the left shows a project named "Test" with sub-items "Test.pro", "Sources", and "Main.c". The main editor area displays the "Test.pro" file with the following content:

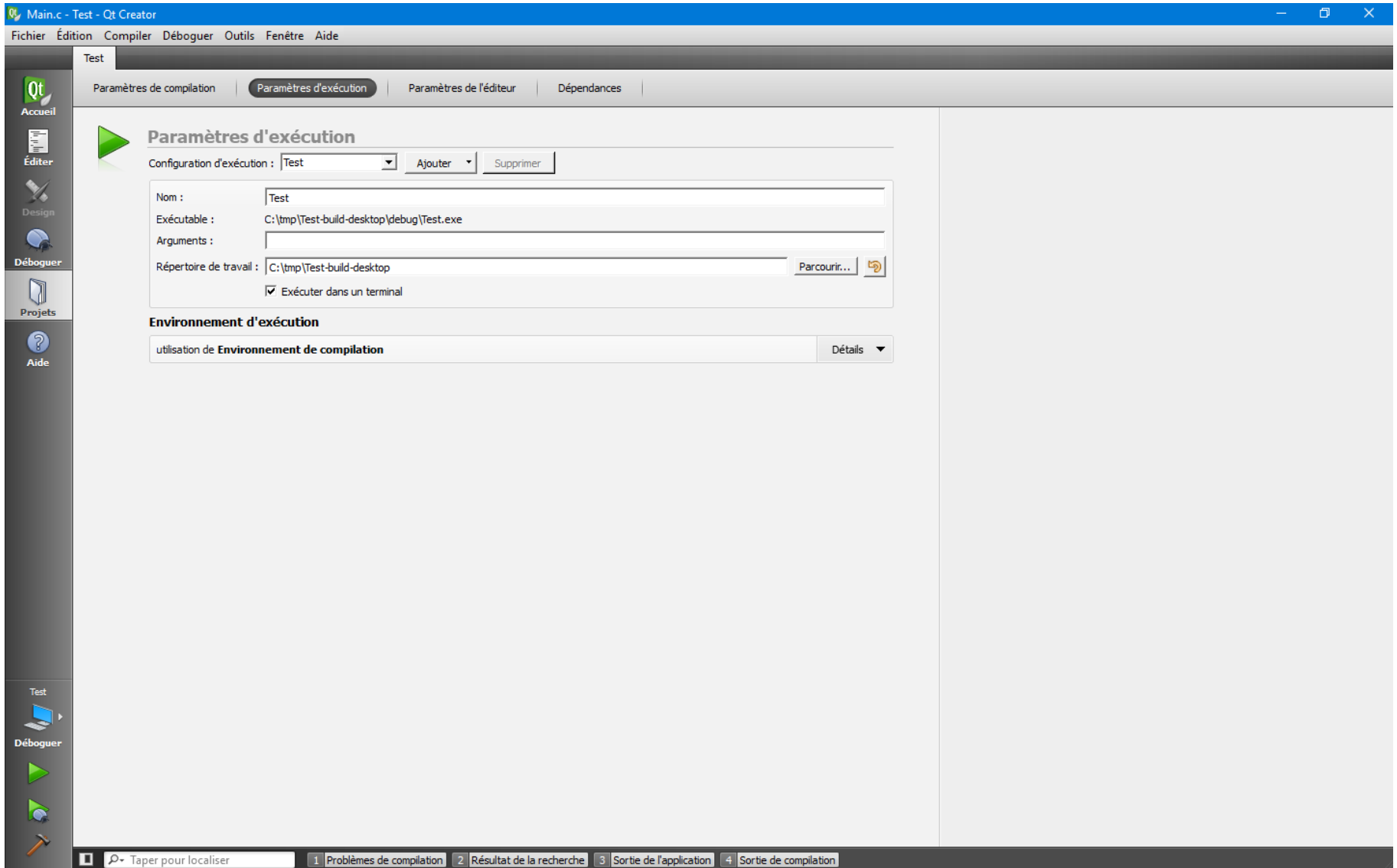
```
1 CONFIG += console
2 SOURCES += \
3     Main.c
4
```

The "Documents ouverts" panel at the bottom left shows "Main.c" and "Test.pro". The "Main.c" editor window displays the following code:

```
1 #include <stdio.h>
2
3 int main()
4 {
5     printf("Test\n");
6     return 0;
7 }
8
```

The status bar at the bottom indicates the current line and column: "Ligne : 8, Col : 1". The bottom toolbar shows icons for "Problèmes de compilation", "Résultat de la recherche", "Sortie de l'application", and "Sortie de compilation".

Créer un projet (non CMake) C simple avec Qt Creator

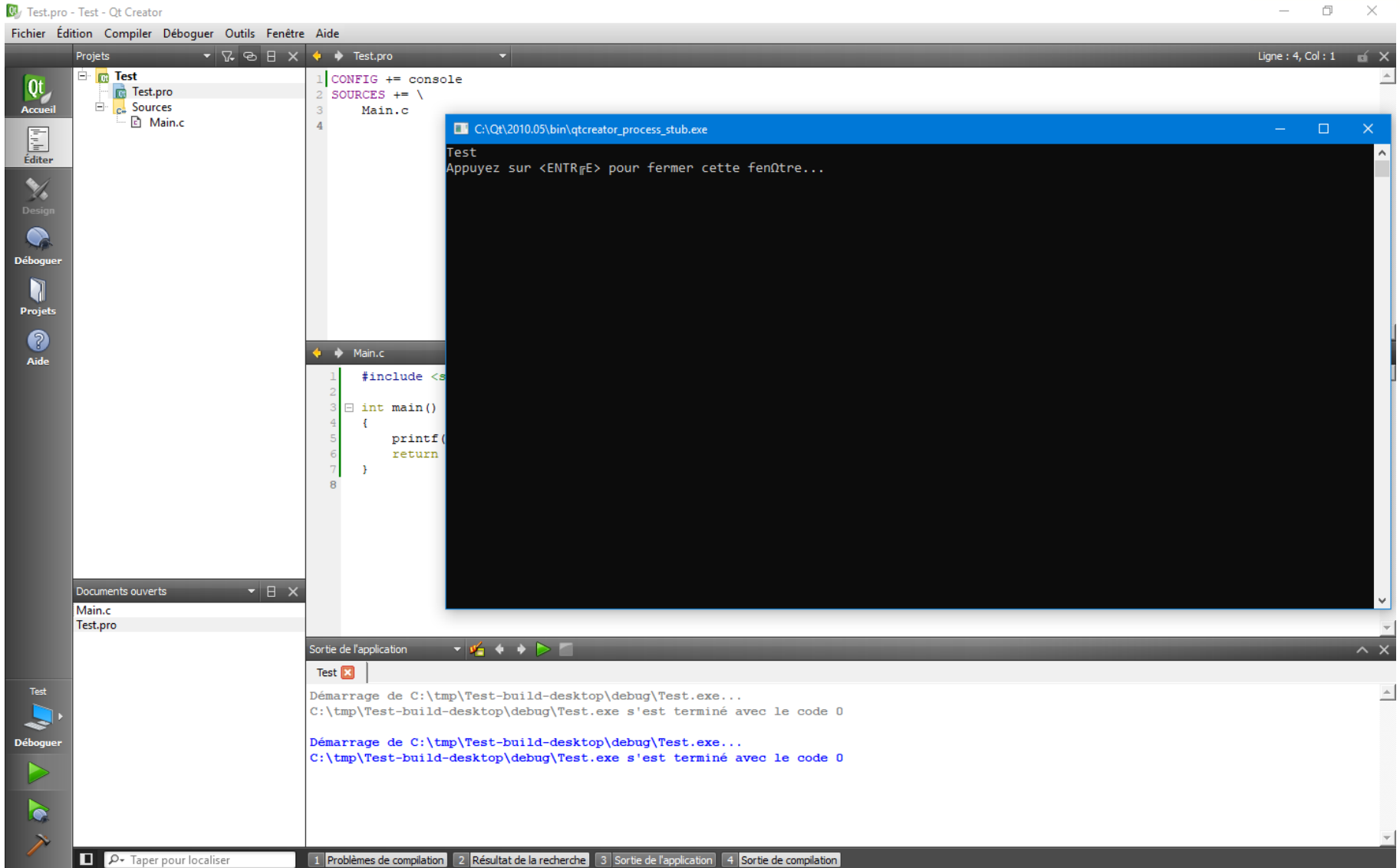


The screenshot shows the Qt Creator IDE interface. The main window is titled 'Main.c - Test - Qt Creator'. The menu bar includes 'Fichier', 'Édition', 'Compiler', 'Déboguer', 'Outils', 'Fenêtre', and 'Aide'. The left sidebar contains icons for 'Qt', 'Accueil', 'Éditer', 'Design', 'Déboguer', 'Projets', and 'Aide'. The main area is divided into two panes. The left pane is titled 'Test' and contains tabs for 'Paramètres de compilation', 'Paramètres d'exécution', 'Paramètres de l'éditeur', and 'Dépendances'. The 'Paramètres d'exécution' tab is active, showing the following settings:

- Configuration d'exécution : Test (dropdown), Ajouter (button), Supprimer (button)
- Nom : Test (text field)
- Exécutable : C:\tmp\Test-build-desktop\debug\Test.exe (text field)
- Arguments : (empty text field)
- Répertoire de travail : C:\tmp\Test-build-desktop (text field), Parcourir... (button), Refresh (button)
- Exécuter dans un terminal
- Environnement d'exécution : utilisation de Environnement de compilation (dropdown), Détails (dropdown arrow)

The bottom status bar shows a search field 'Taper pour localiser' and four numbered tabs: 1 Problèmes de compilation, 2 Résultat de la recherche, 3 Sortie de l'application, 4 Sortie de compilation.

Créer un projet (non CMake) C simple avec Qt Creator



The screenshot displays the Qt Creator IDE interface for a C project named "Test". The "Projets" pane on the left shows the project structure with "Test.pro", "Sources", and "Main.c". The main editor shows the "Test.pro" file with the following content:

```
1 CONFIG += console
2 SOURCES += \
3     Main.c
4
```

The "Main.c" file is also visible in the editor, showing a simple C program:

```
1 #include <stdio.h>
2
3 int main()
4 {
5     printf("Test\n");
6     return 0;
7 }
8
```

A console window titled "C:\Qt\2010.05\bin\qtcreator_process_stub.exe" is open, displaying the output of the program:

```
Test
Appuyez sur <ENTRÉE> pour fermer cette fenetre...
```

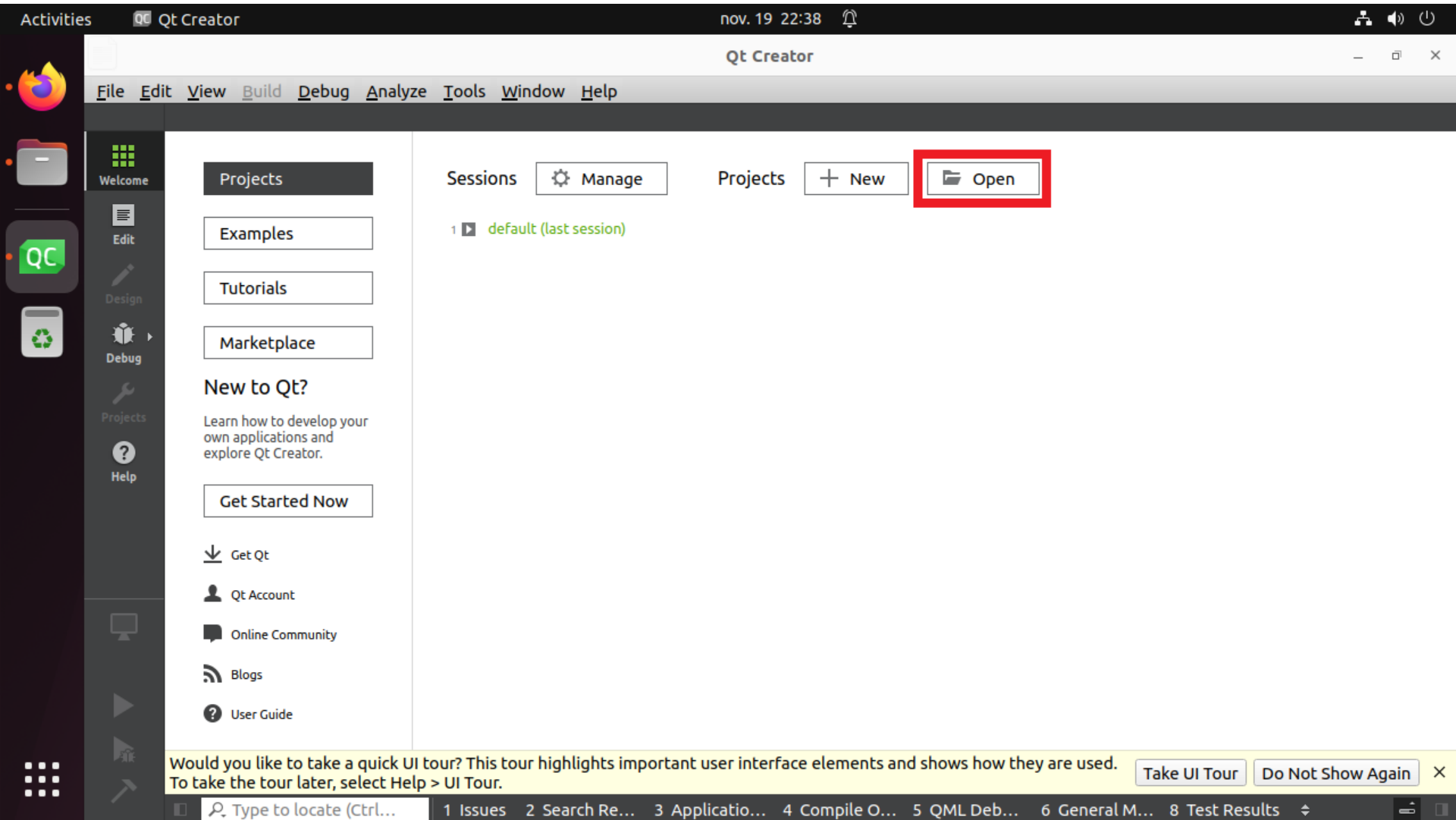
The "Sortie de l'application" (Application Output) window at the bottom shows the execution log:

```
Test [x]
Démarrage de C:\tmp\Test-build-desktop\debug\Test.exe...
C:\tmp\Test-build-desktop\debug\Test.exe s'est terminé avec le code 0

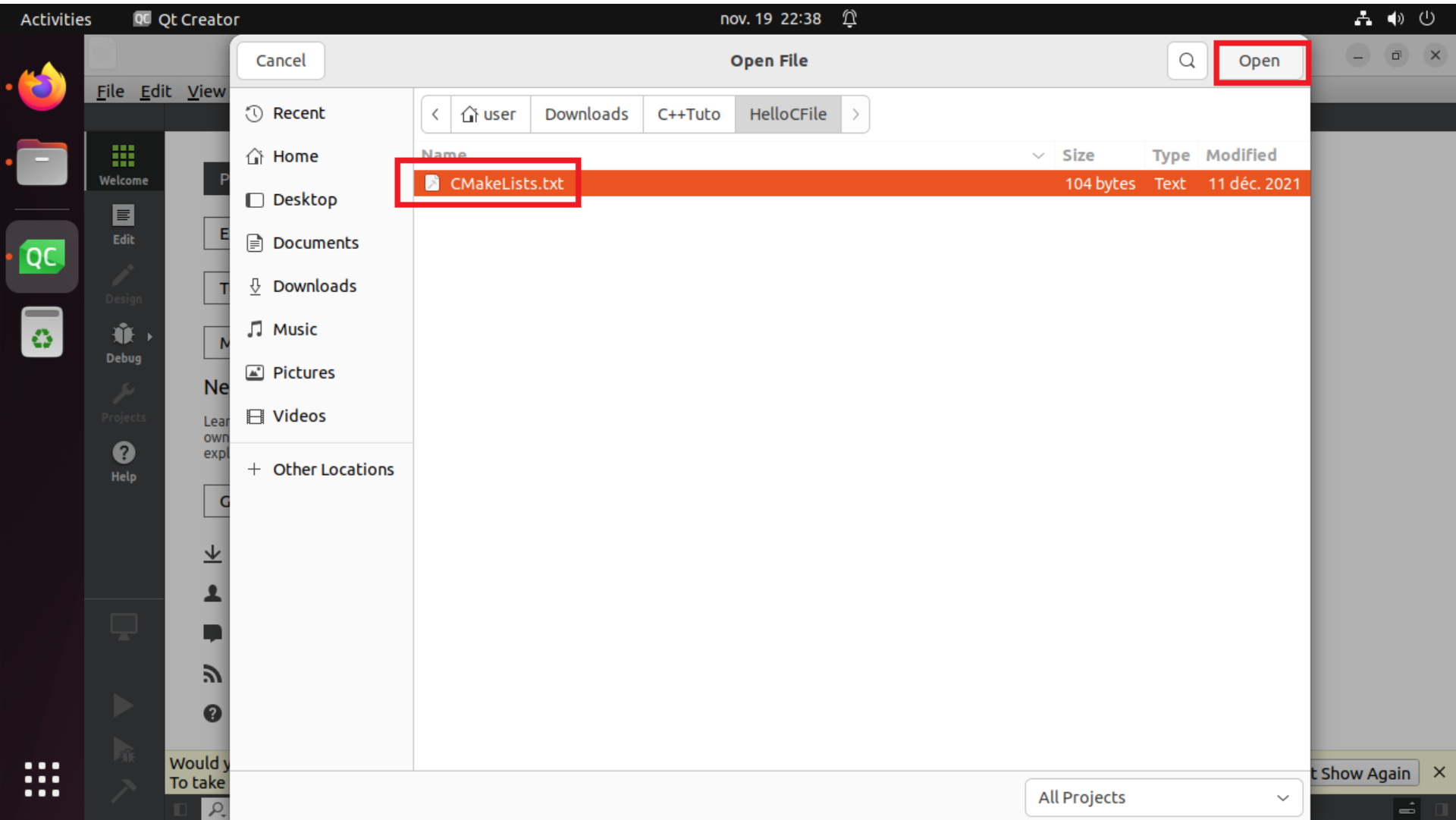
Démarrage de C:\tmp\Test-build-desktop\debug\Test.exe...
C:\tmp\Test-build-desktop\debug\Test.exe s'est terminé avec le code 0
```

The status bar at the bottom indicates the current view is "Sortie de l'application" (Application Output).

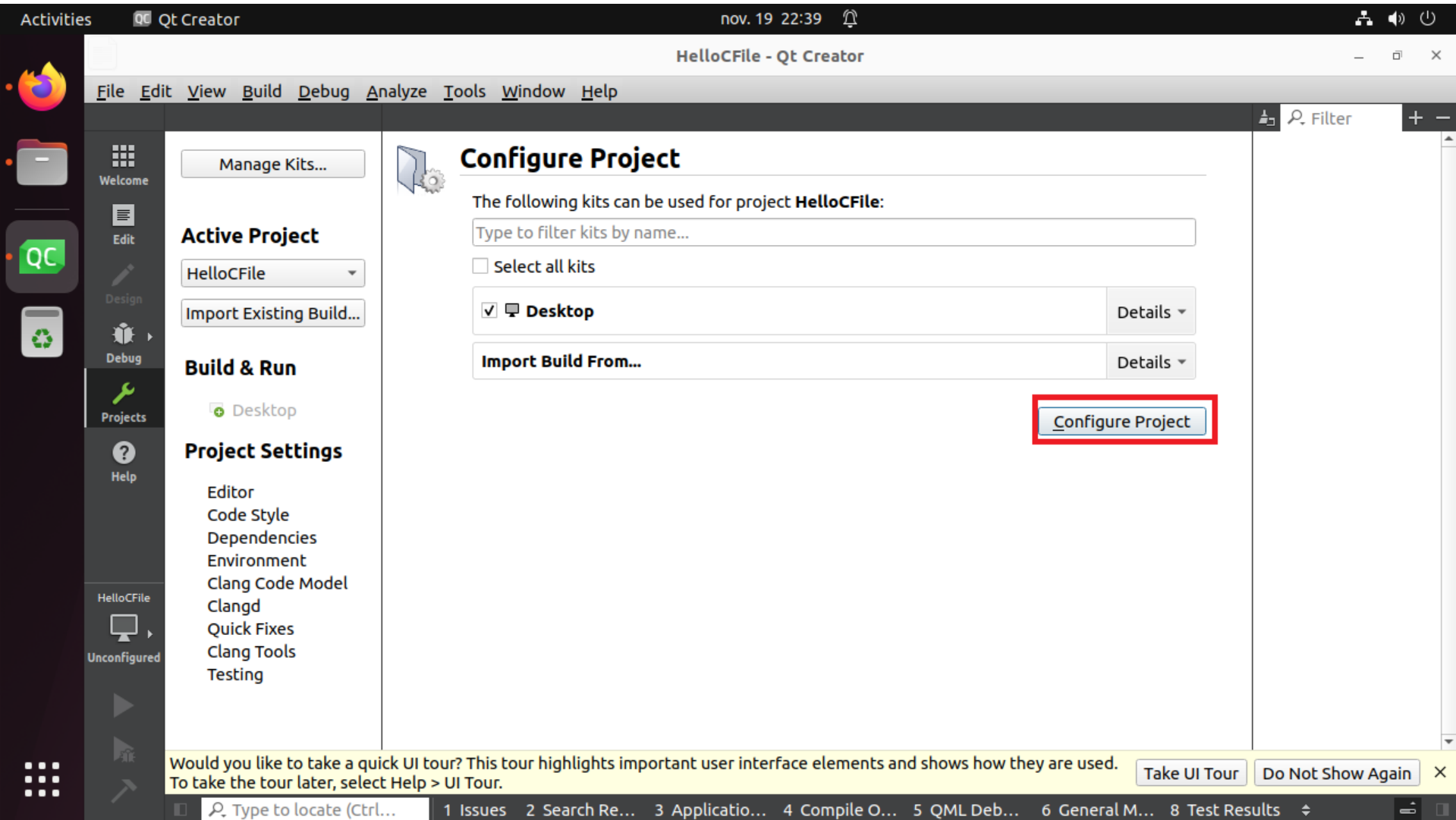
Opening and debugging a CMake project using Qt Creator



Opening and debugging a CMake project using Qt Creator



Opening and debugging a CMake project using Qt Creator



The screenshot shows the Qt Creator IDE interface. The main window title is "HelloCFile - Qt Creator". The menu bar includes File, Edit, View, Build, Debug, Analyze, Tools, Window, and Help. The left sidebar contains icons for Welcome, Edit, Design, Debug, Projects, and Help. The "Projects" section shows "HelloCFile" as the active project. The "Build & Run" section shows "Desktop" as the selected kit. The "Project Settings" section lists various settings like Editor, Code Style, Dependencies, Environment, Clang Code Model, Clang, Quick Fixes, Clang Tools, and Testing. The main area displays the "Configure Project" dialog for the "HelloCFile" project. It lists available kits: "Desktop" (checked) and "Import Build From...". A red box highlights the "Configure Project" button. A yellow notification bar at the bottom asks if the user wants to take a quick UI tour.

Activities Qt Creator nov. 19 22:39

HelloCFile - Qt Creator

File Edit View Build Debug Analyze Tools Window Help

Manage Kits...

Active Project
HelloCFile

Import Existing Build...

Build & Run
Desktop

Project Settings
Editor
Code Style
Dependencies
Environment
Clang Code Model
Clang
Quick Fixes
Clang Tools
Testing

Configure Project

The following kits can be used for project **HelloCFile**:

Type to filter kits by name...

Select all kits

Desktop Details ▾

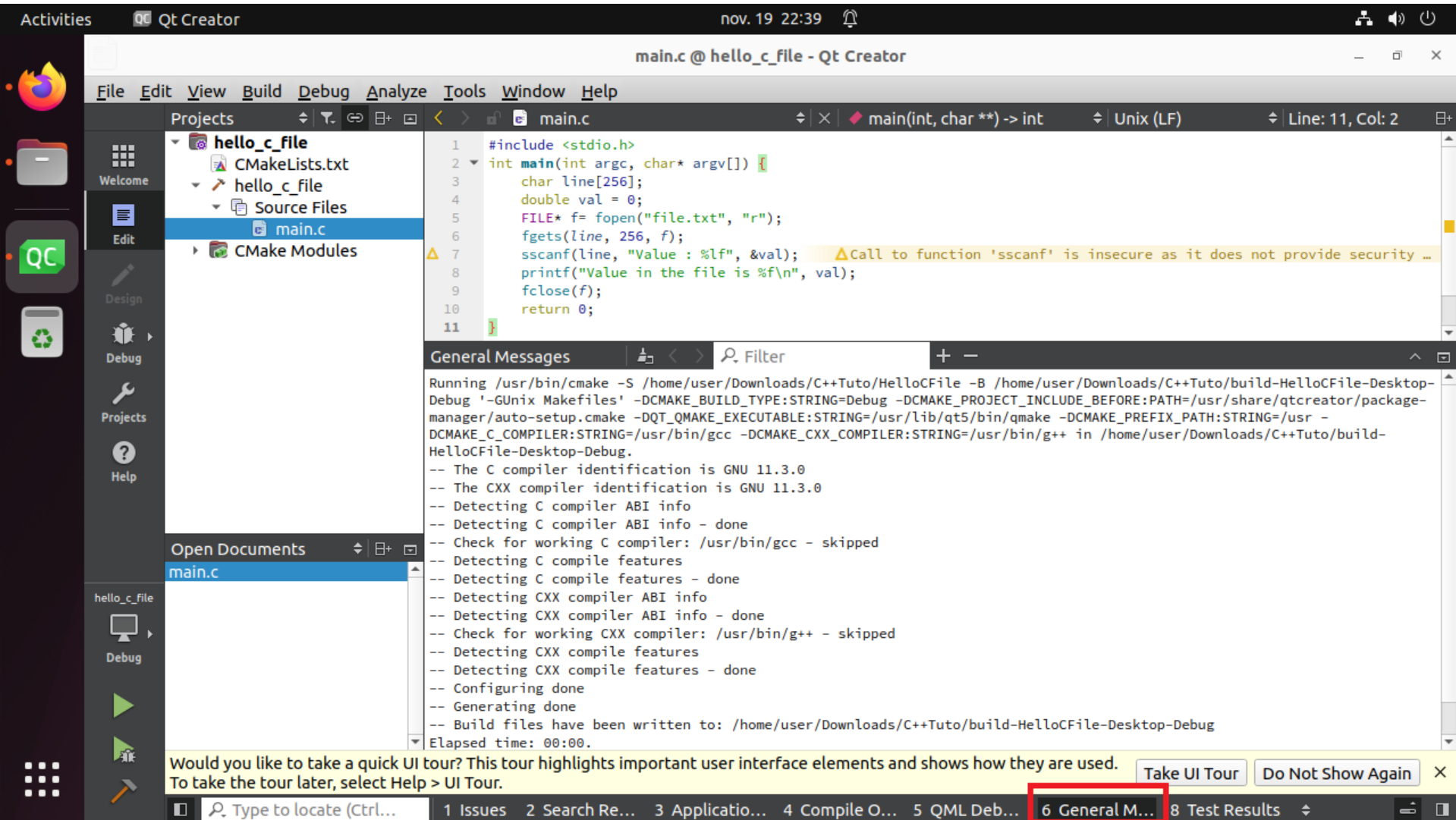
Import Build From... Details ▾

Configure Project

Would you like to take a quick UI tour? This tour highlights important user interface elements and shows how they are used. To take the tour later, select Help > UI Tour. Take UI Tour Do Not Show Again

Type to locate (Ctrl... 1 Issues 2 Search Re... 3 Applicatio... 4 Compile O... 5 QML Deb... 6 General M... 8 Test Results

Opening and debugging a CMake project using Qt Creator



Activities Qt Creator nov. 19 22:39

main.c @ hello_c_file - Qt Creator

File Edit View Build Debug Analyze Tools Window Help

Projects

- hello_c_file
 - CMakeLists.txt
 - hello_c_file
 - Source Files
 - main.c
 - CMake Modules

```
1 #include <stdio.h>
2 int main(int argc, char* argv[]) {
3     char line[256];
4     double val = 0;
5     FILE* f= fopen("file.txt", "r");
6     fgets(line, 256, f);
7     sscanf(line, "Value : %lf", &val);
8     printf("Value in the file is %f\n", val);
9     fclose(f);
10    return 0;
11 }
```

General Messages

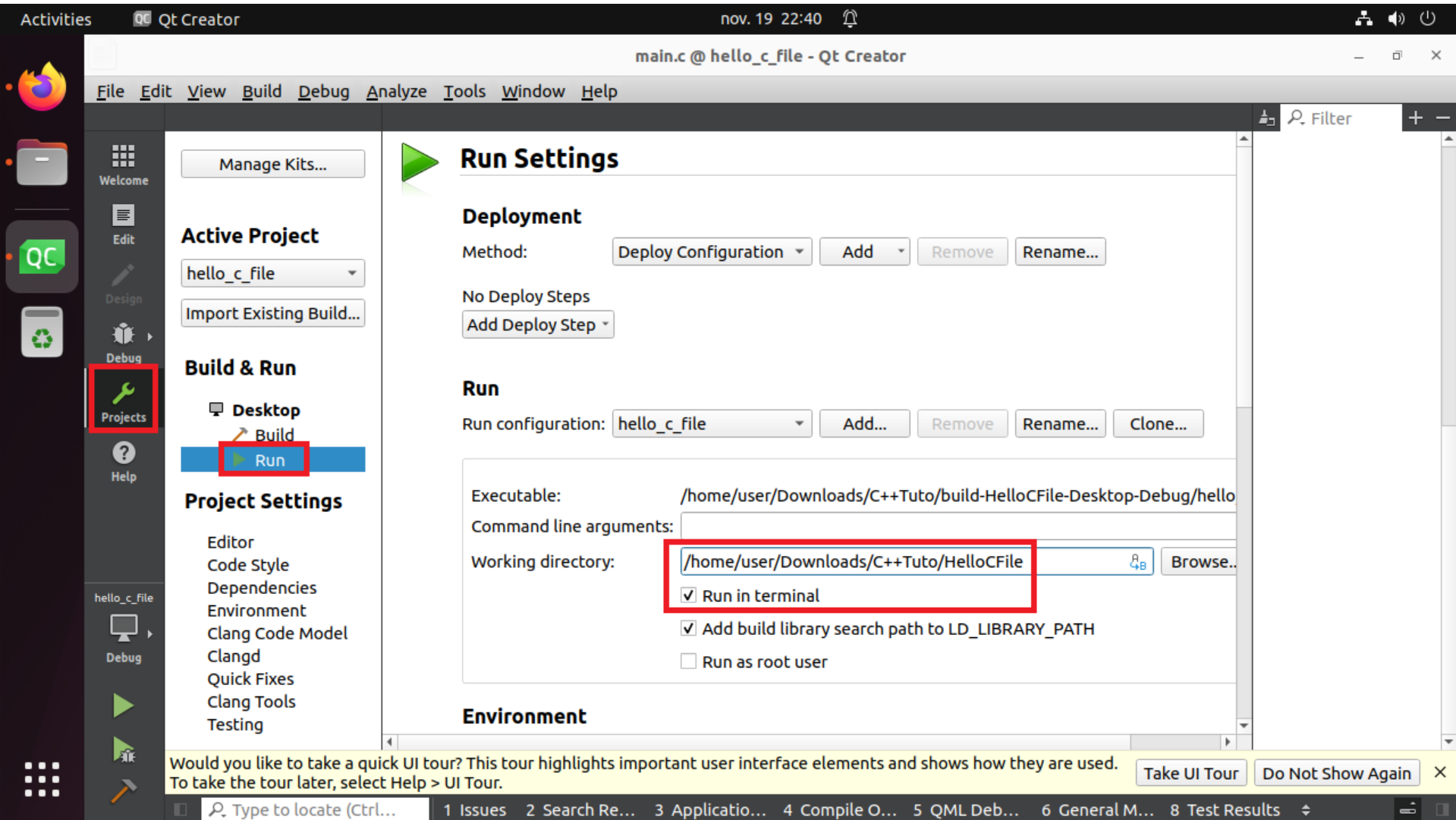
Running /usr/bin/cmake -S /home/user/Downloads/C++Tuto/HelloCFile -B /home/user/Downloads/C++Tuto/build-HelloCFile-Desktop-Debug -GUnix Makefiles -DCMAKE_BUILD_TYPE:STRING=Debug -DCMAKE_PROJECT_INCLUDE_BEFORE:PATH=/usr/share/qtcreator/package-manager/auto-setup.cmake -DQT_QMAKE_EXECUTABLE:STRING=/usr/lib/qt5/bin/qmake -DCMAKE_PREFIX_PATH:STRING=/usr -DCMAKE_C_COMPILER:STRING=/usr/bin/gcc -DCMAKE_CXX_COMPILER:STRING=/usr/bin/g++ in /home/user/Downloads/C++Tuto/build-HelloCFile-Desktop-Debug.

-- The C compiler identification is GNU 11.3.0
-- The CXX compiler identification is GNU 11.3.0
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working C compiler: /usr/bin/gcc - skipped
-- Detecting C compile features
-- Detecting C compile features - done
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Check for working CXX compiler: /usr/bin/g++ - skipped
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Configuring done
-- Generating done
-- Build files have been written to: /home/user/Downloads/C++Tuto/build-HelloCFile-Desktop-Debug
Elapsed time: 00:00.

Would you like to take a quick UI tour? This tour highlights important user interface elements and shows how they are used. To take the tour later, select Help > UI Tour. [Take UI Tour](#) [Do Not Show Again](#)

Type to locate (Ctrl... 1 Issues 2 Search Re... 3 Applicatio... 4 Compile O... 5 QML Deb... 6 General M... 8 Test Results

Opening and debugging a CMake project using Qt Creator

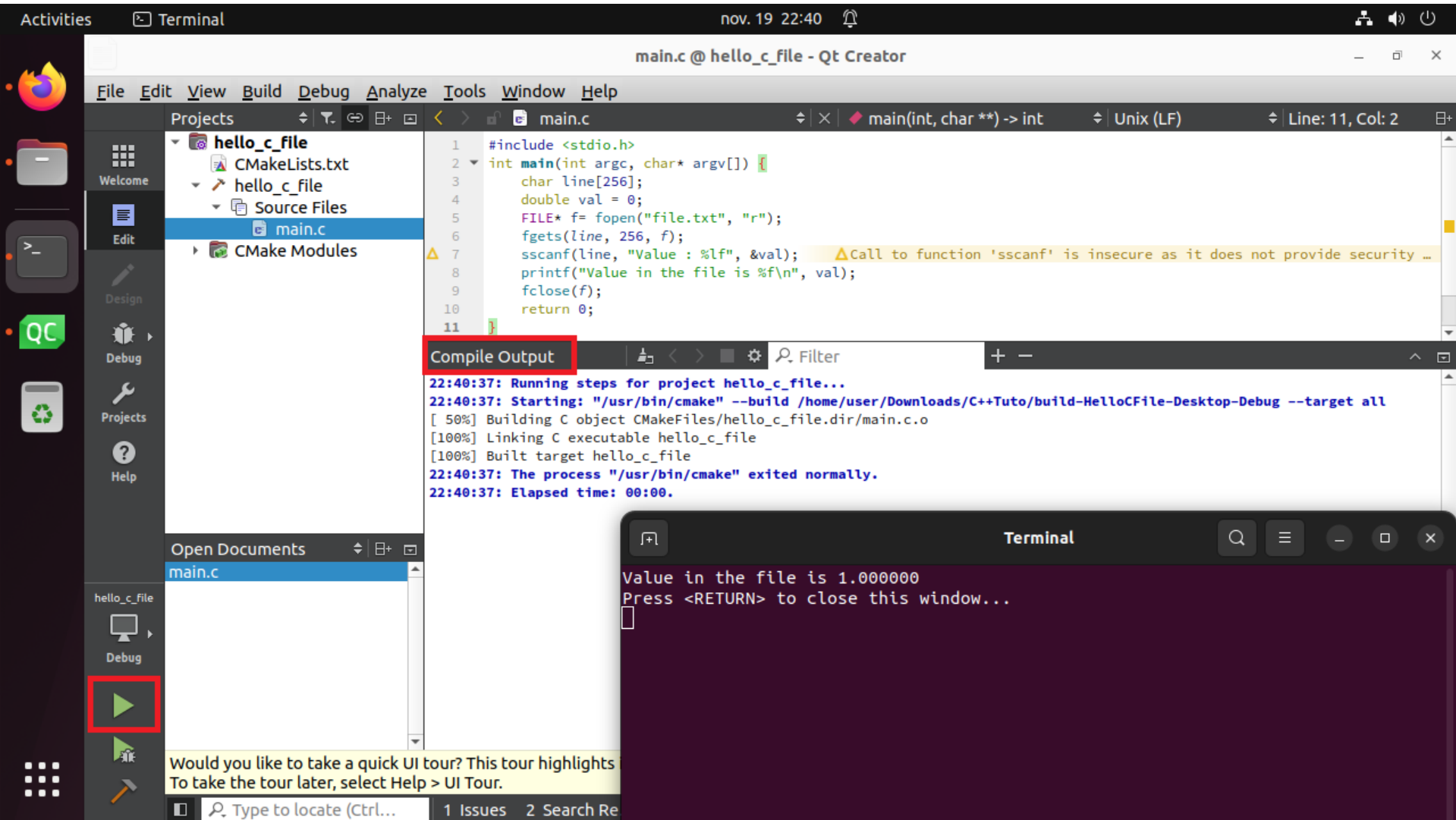


The screenshot shows the Qt Creator IDE interface. The main window displays the 'Run Settings' dialog for the project 'hello_c_file'. The 'Run' section is expanded, showing the following configuration:

- Run configuration: hello_c_file
- Executable: /home/user/Downloads/C++Tuto/build-HelloCFile-Desktop-Debug/hello
- Command line arguments: (empty)
- Working directory: /home/user/Downloads/C++Tuto/HelloCFile
- Run in terminal
- Add build library search path to LD_LIBRARY_PATH
- Run as root user

The 'Run' button in the 'Build & Run' section of the left sidebar is highlighted with a red box. The 'Working directory' field in the 'Run' section is also highlighted with a red box. A yellow notification bar at the bottom of the window reads: 'Would you like to take a quick UI tour? This tour highlights important user interface elements and shows how they are used. To take the tour later, select Help > UI Tour.' The notification bar includes 'Take UI Tour' and 'Do Not Show Again' buttons.

Opening and debugging a CMake project using Qt Creator



The screenshot shows the Qt Creator IDE interface. The main window displays the source code for `main.c` in a CMake project named `hello_c_file`. The code includes `<stdio.h>` and defines a `main` function that reads a file named `file.txt` and prints its contents. A warning is visible on line 7: `Call to function 'sscanf' is insecure as it does not provide security ...`. The `Compile Output` window shows the following output:

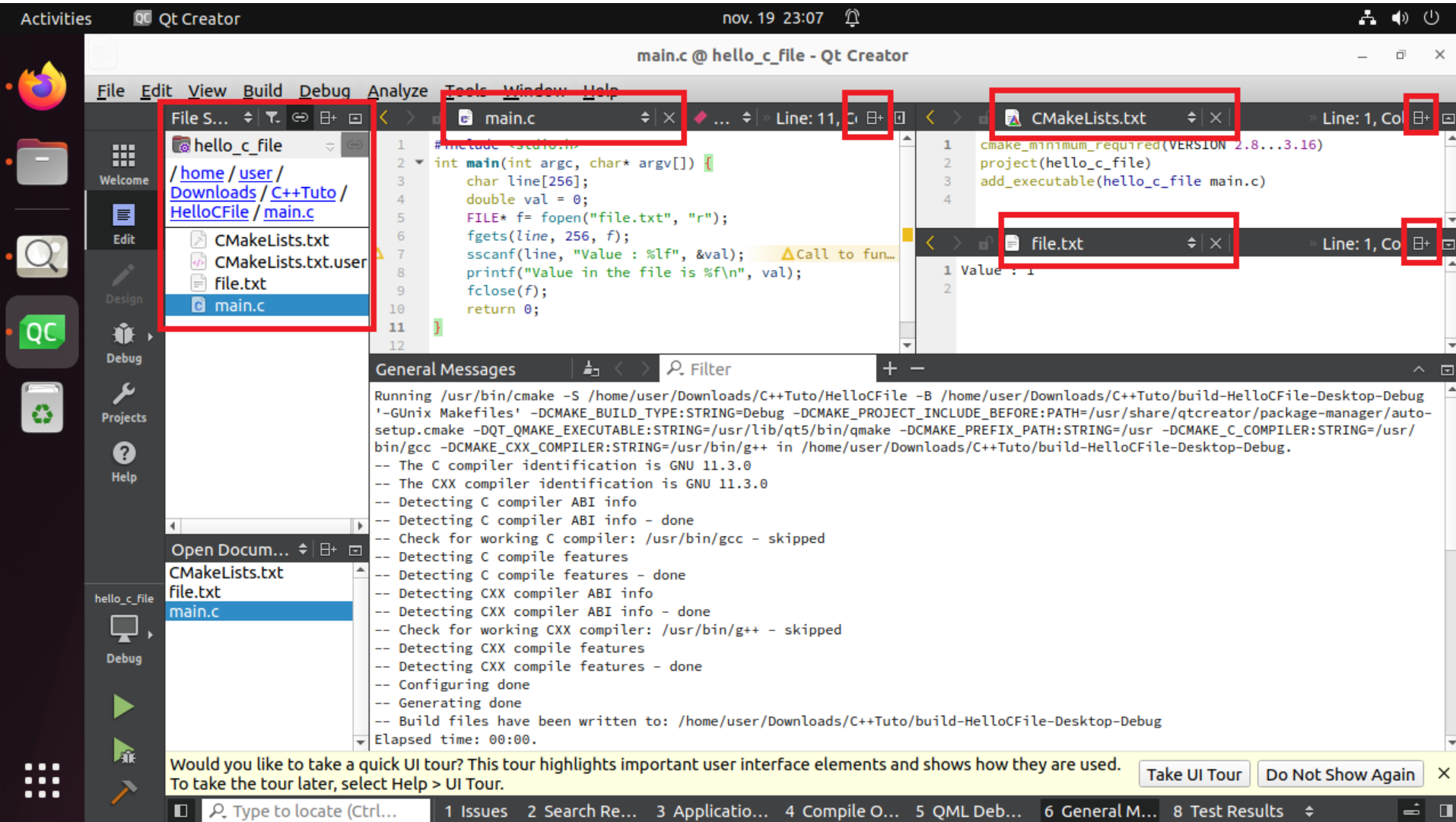
```
22:40:37: Running steps for project hello_c_file...
22:40:37: Starting: "/usr/bin/cmake" --build /home/user/Downloads/C++Tuto/build-HelloCFile-Desktop-Debug --target all
[ 50%] Building C object CMakeFiles/hello_c_file.dir/main.c.o
[100%] Linking C executable hello_c_file
[100%] Built target hello_c_file
22:40:37: The process "/usr/bin/cmake" exited normally.
22:40:37: Elapsed time: 00:00.
```

The `Terminal` window shows the program's output:

```
Value in the file is 1.000000
Press <RETURN> to close this window...
```

The `Debug` icon in the left sidebar is highlighted with a red box, indicating the next step in the process.

Opening and debugging a CMake project using Qt Creator



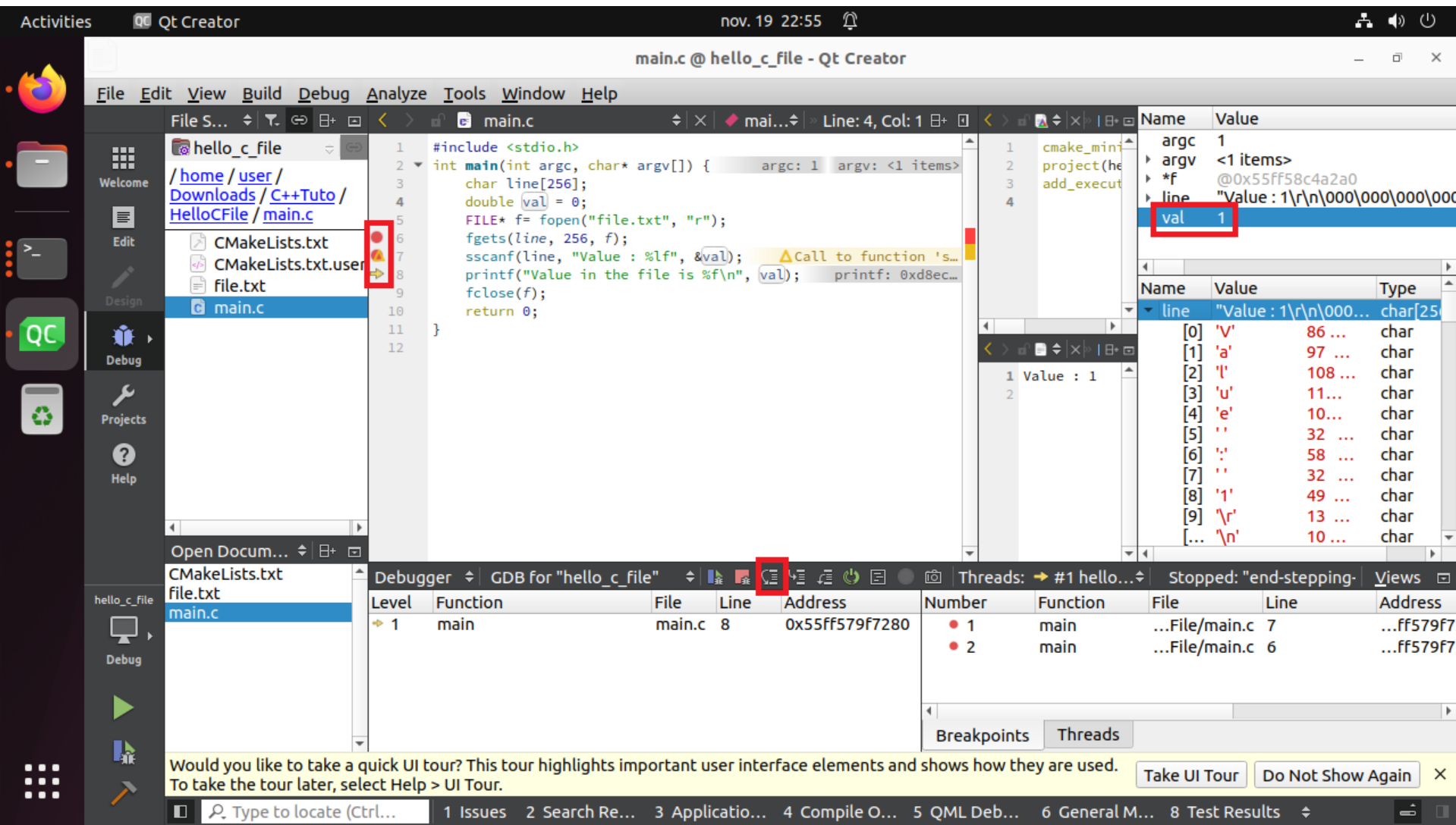
The screenshot displays the Qt Creator IDE interface. The top bar shows the date and time as "nov. 19 23:07". The main window title is "main.c @ hello_c_file - Qt Creator".

The interface is divided into several panes:

- File Explorer:** Located on the left, it shows the project structure under "hello_c_file". The files listed are "CMakeLists.txt", "CMakeLists.txt.user", "file.txt", and "main.c".
- Code Editor:** The central pane shows the source code for "main.c" and "CMakeLists.txt". The "main.c" code includes a header, a main function that reads a file, and a return statement. The "CMakeLists.txt" file contains CMake configuration for a project named "hello_c_file".
- Console:** The bottom pane shows the output of the CMake command. It includes the command used to run CMake, the compiler identification (GNU 11.3.0), and the configuration and build process details.

At the bottom of the IDE, there is a yellow notification bar that reads: "Would you like to take a quick UI tour? This tour highlights important user interface elements and shows how they are used. To take the tour later, select Help > UI Tour." There are two buttons: "Take UI Tour" and "Do Not Show Again".

Opening and debugging a CMake project using Qt Creator



The screenshot displays the Qt Creator IDE interface. The main editor shows the source code of `main.c` with a breakpoint set at line 8. The right sidebar shows the variable explorer with the variable `val` set to 1. The bottom debugger window shows the call stack with the current frame at `main.c:8`.

```
#include <stdio.h>
int main(int argc, char* argv[]) {
    char line[256];
    double val = 0;
    FILE* f = fopen("file.txt", "r");
    fgets(line, 256, f);
    sscanf(line, "Value : %lf", &val);
    printf("Value in the file is %f\n", val);
    fclose(f);
    return 0;
}
```

Name	Value	Type
argc	1	
argv	<1 items>	
*f	@0x55ff58c4a2a0	
line	"Value : 1\r\n000\000\000\000"	char[256]
val	1	double

Level	Function	File	Line	Address	Number	Function	File	Line	Address
1	main	main.c	8	0x55ff579f7280	1	main	...File/main.c	7	...ff579f7
2	main	main.c	6	...ff579f7	2	main	...File/main.c	6	...ff579f7

Tips Qt Creator

- The **current directory of the application** is set by default to some build folder, check project **Run Settings**
- Use **CTRL+SPACE** to get propositions of **auto code completion**.
- **Pause** during one or two seconds **the mouse above a variable or function** to get information on it.
- **Right-click** on a function or variable and choose **Follow Symbol Under Cursor** to see the corresponding declaration code in the source file.

Tips Qt Creator

- **Delete** the generated files and folders when moving your project or when the project behavior looks inconsistent (e.g. a wrong version of your program in another path is launched), and reopen the project to force Qt to regenerate them...
- If you do not see the output of **printf()** inside **Qt Creator Application Output** window, check that you have **CONFIG += console** in your **.pro**, try also **Projects > Run Settings > Run in terminal**
- See also <http://www.ensta-bretagne.fr/lebars/tutorials/>



Visual Studio

Visual Studio

■ Visual Studio

• Versions

Visual Studio 6 : date de 1998

Visual Studio 2002 (7), 2003 ou .Net (7.1) : refonte de l'IDE et ajout des projets .Net

Visual Studio 2005 (8), 2008 (9) : quelques mises à jour

Visual Studio 2010 (10) : changements significatifs

Visual Studio 2012 (11), 2013 (12), 2015 (14) : quelques mises à jour

Visual Studio 2017 (15), 2019 (16) : changements significatifs pour l'installation

Visual Studio 2022 (17) : IDE en 64 bit

Visual Studio 2026 : vient de sortir fin 2025, peu utilisé pour le moment donc risque de problèmes de compatibilité...

- Difficile à prendre en main au début : nombreux types de projets, nombreuses options complexes
- Mais assez abouti, très utilisé, beaucoup d'aide disponible sur Internet
- Installation : voir e.g. https://www.ensta-bretagne.fr/lebars/Share/setup_vs_opencv.pdf

- **Visual Studio Code est significativement différent de Visual Studio (Community, Pro, Enterprise, etc.)!**










- Visual Studio

- Organisation

Solution/Workspace (fichier `.dsw/.sln`) : ensemble de projets

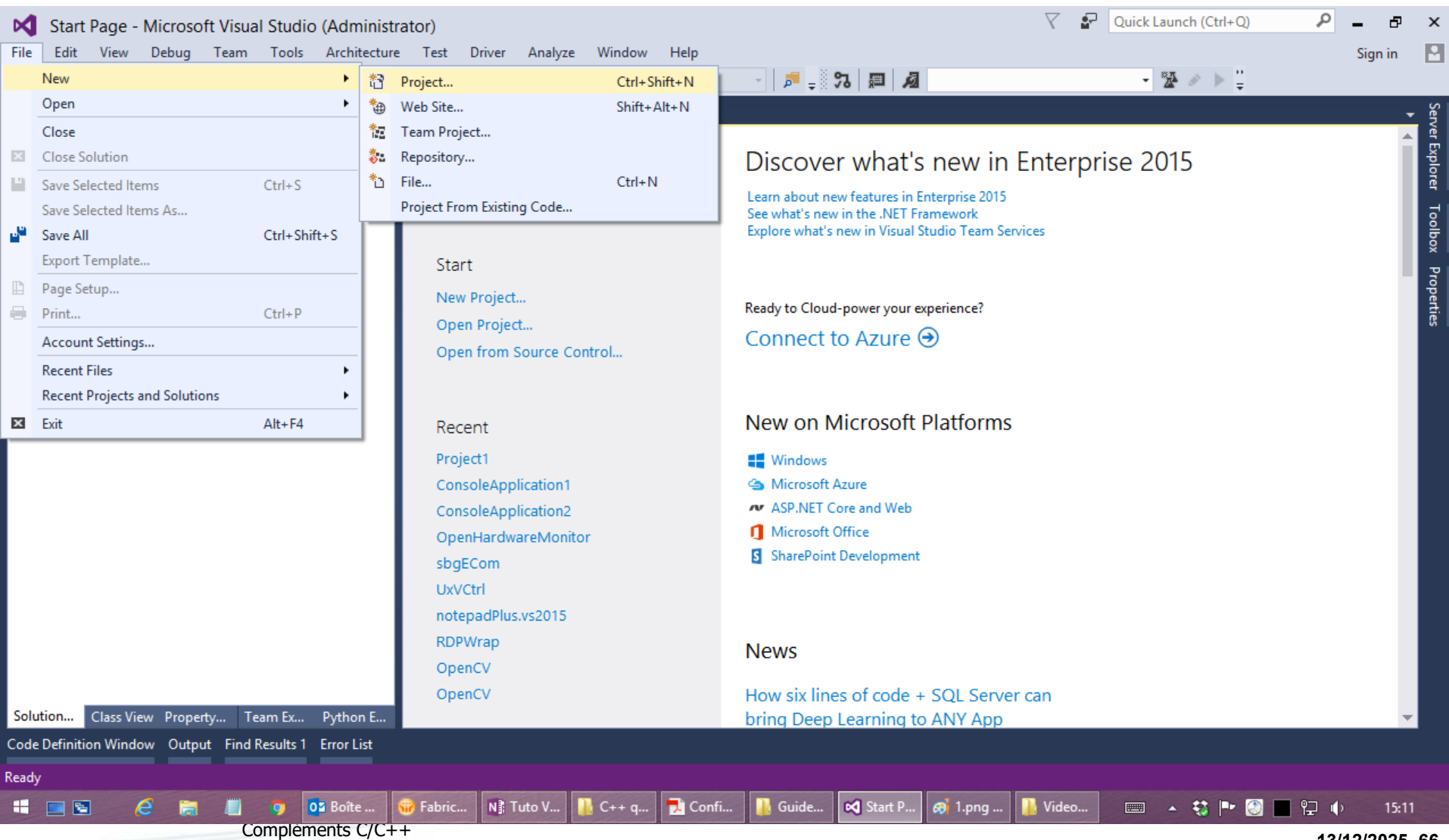
Projet (fichiers `.dsp/.vcproj/.vcxproj+.vcxproj.filters`) : ensemble de fichiers (`.c, .cpp, .h...`) et informations nécessaires à la génération d'un exécutable ou bibliothèque

	Visual Studio 6	Visual Studio 2002-2008	Visual Studio 2010-2022
Workspace/Solution	 <p>ServoTestVC6.dsw VC++ 6 Workspace 547 bytes</p>	 <p>FichierImageOpenCV.sln Microsoft Visual Studio Solution Version: Visual Studio 2008</p>	 <p>Test.sln Microsoft Visual Studio Solution Version: Visual Studio 14</p>
Projet	 <p>ServoTestVC6.dsp VC++ 6 Project 4.73 KB</p>	 <p>FichierImageOpenCV.vcproj VC++ Project 5 Ko</p>	 <p>Test.vcxproj VC++ Project 7.11 KB</p>  <p>Test.vcxproj.filters VC++ Project Filters File 954 bytes</p>

Créer un projet C simple (non CMake) avec Visual Studio

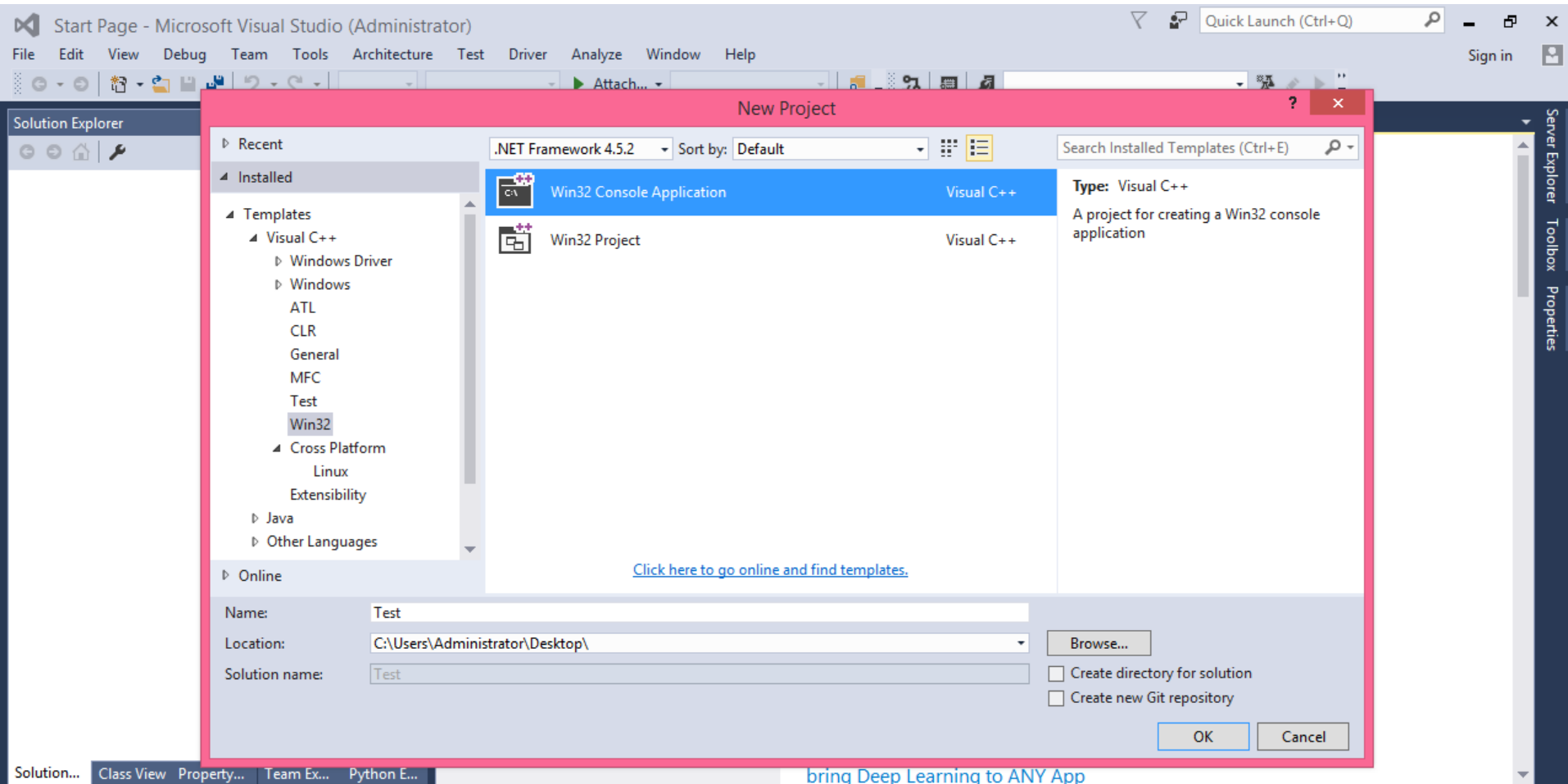
- Faire **File > New Project > Visual C++ > Win32 Console Application** et dans **Application Settings** cocher **Empty project** et décocher **Security Development Lifecycle (SDL) checks**
- Dans le projet faire **Add > New Item > C++ File**, même s'il ne propose pas de mettre des fichiers **.c** si on met **main.c** il va bien le prendre comme un **.c**
- Dans les **Propriétés** du projet, il peut être nécessaire de changer l'option **Character Set** en **Not Set**

Créer un projet C simple (non CMake) avec Visual Studio



The screenshot shows the Microsoft Visual Studio Start Page. The 'File' menu is open, displaying options such as 'New', 'Open', 'Close', 'Save Selected Items', 'Save All', 'Export Template...', 'Page Setup...', 'Print...', 'Account Settings...', 'Recent Files', 'Recent Projects and Solutions', and 'Exit'. The 'New' submenu is also open, showing 'Project...' (Ctrl+Shift+N), 'Web Site...' (Shift+Alt+N), 'Team Project...', 'Repository...', 'File...' (Ctrl+N), and 'Project From Existing Code...'. The main Start Page area includes a 'Discover what's new in Enterprise 2015' section, a 'Ready to Cloud-power your experience?' section with a 'Connect to Azure' button, a 'New on Microsoft Platforms' section listing Windows, Microsoft Azure, ASP.NET Core and Web, Microsoft Office, and SharePoint Development, and a 'News' section with a link to 'How six lines of code + SQL Server can bring Deep Learning to ANY App'. The taskbar at the bottom shows several open applications, including 'Complements C/C++', and the system tray displays the time as 15:11.

Créer un projet C simple (non CMake) avec Visual Studio



The screenshot shows the 'New Project' dialog box in Visual Studio. The 'Win32 Console Application' template is selected. The project name is 'Test' and the location is 'C:\Users\Administrator\Desktop\'. The solution name is also 'Test'. The dialog box is highlighted with a red border.

Start Page - Microsoft Visual Studio (Administrator)

File Edit View Debug Team Tools Architecture Test Driver Analyze Window Help

Quick Launch (Ctrl+Q)

Sign in

Solution Explorer

New Project

Recent .NET Framework 4.5.2 Sort by: Default Search Installed Templates (Ctrl+E)

Installed

- Templates
 - Visual C++
 - Windows Driver
 - Windows
 - ATL
 - CLR
 - General
 - MFC
 - Test
 - Win32
 - Cross Platform
 - Linux
 - Extensibility
 - Java
 - Other Languages

Online

Name: Test

Location: C:\Users\Administrator\Desktop\ Browse...

Solution name: Test

Create directory for solution

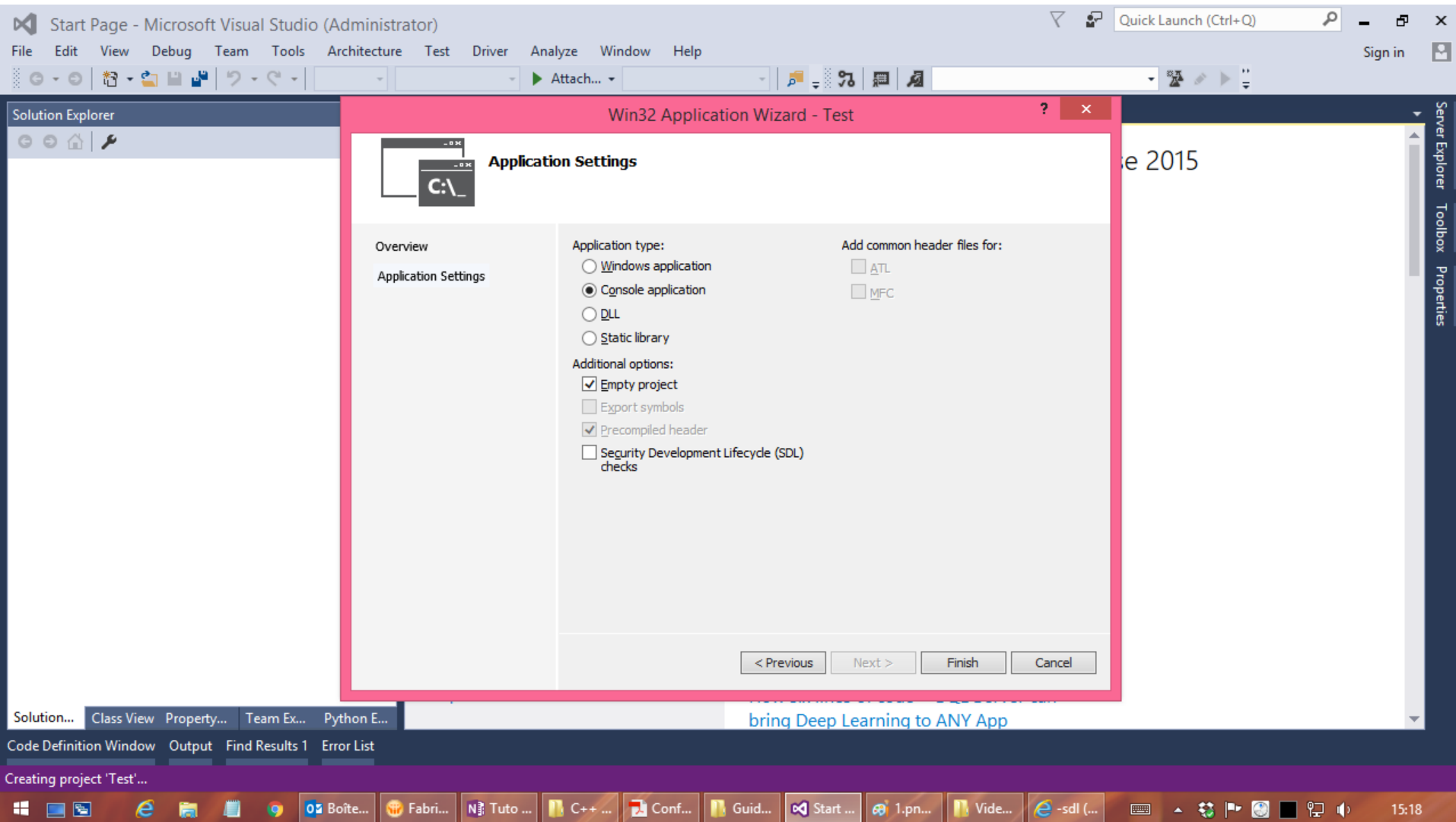
Create new Git repository

OK Cancel

[Click here to go online and find templates.](#)

bring Deep Learning to ANY App

Créer un projet C simple (non CMake) avec Visual Studio

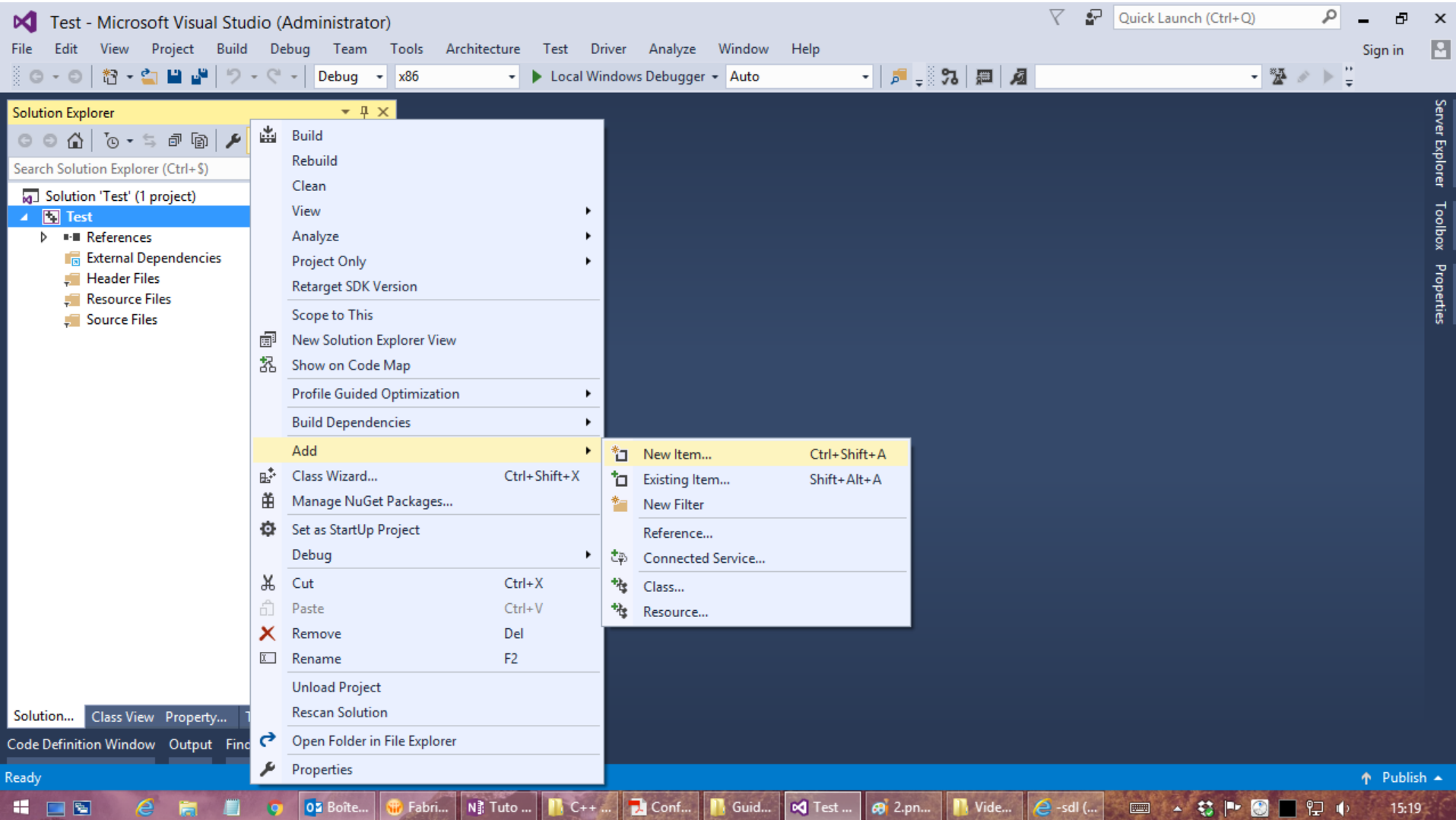


The screenshot shows the Visual Studio interface with the 'Win32 Application Wizard - Test' dialog box open. The dialog is titled 'Application Settings' and is divided into two main sections: 'Overview' and 'Application Settings'. The 'Application Settings' section is active and contains the following options:

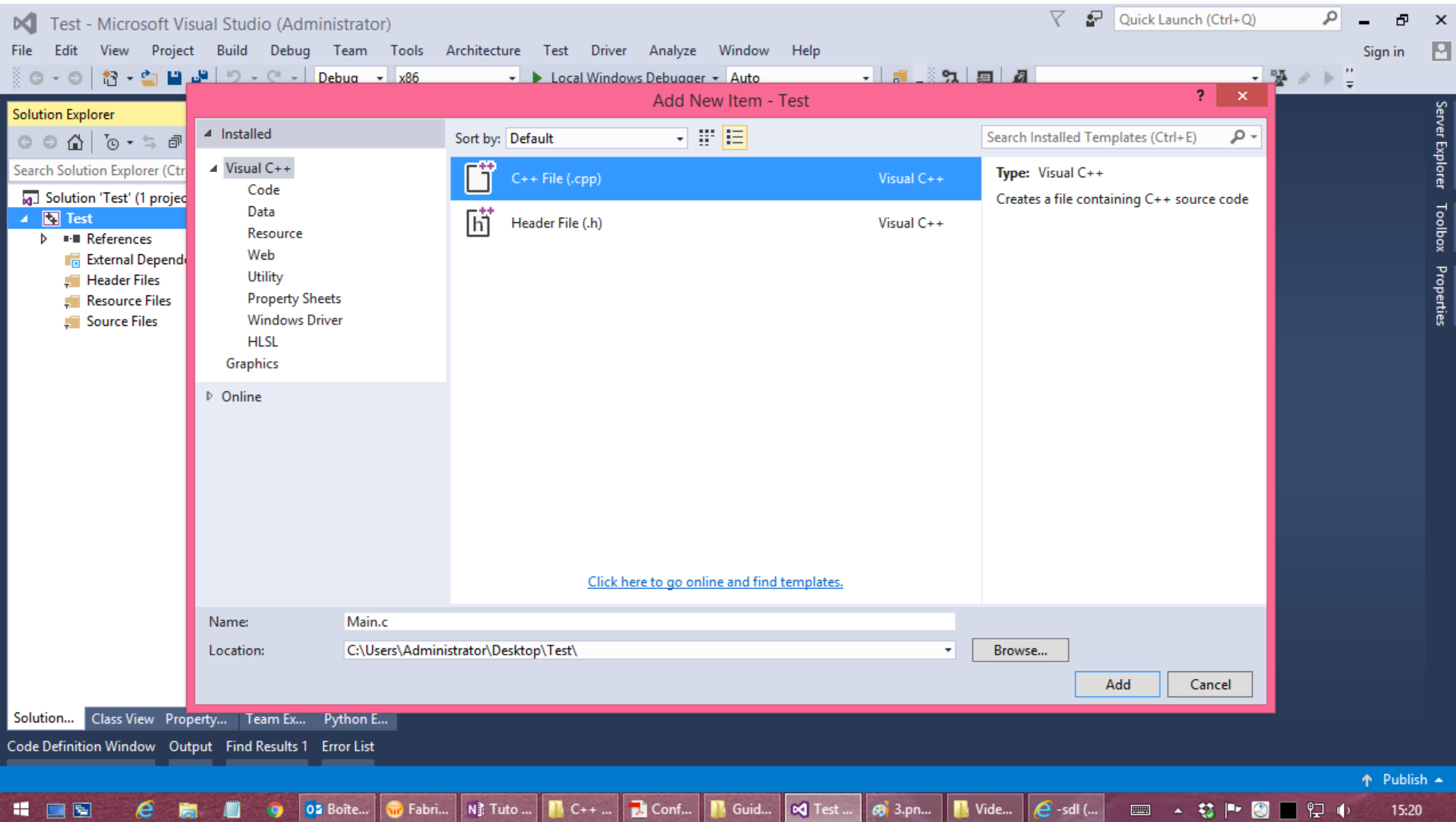
- Application type:**
 - Windows application
 - Console application
 - DLL
 - Static library
- Additional options:**
 - Empty project
 - Export symbols
 - Precompiled header
 - Security Development Lifecycle (SDL) checks
- Add common header files for:**
 - ATL
 - MFC

At the bottom of the dialog, there are four buttons: '< Previous', 'Next >', 'Finish', and 'Cancel'. The 'Next >' button is highlighted, indicating the user is ready to proceed to the next step in the wizard.

Créer un projet C simple (non CMake) avec Visual Studio



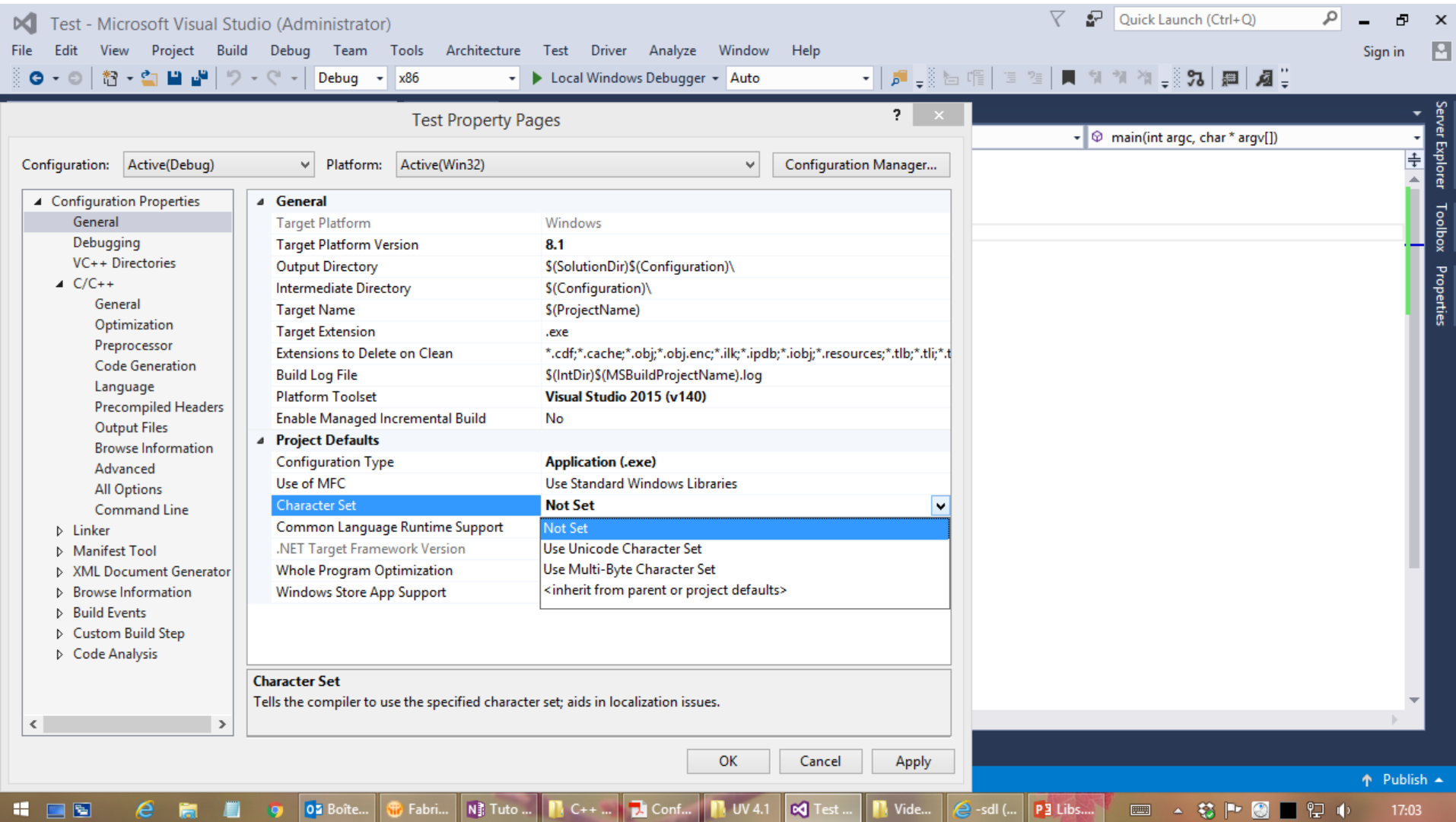
Créer un projet C simple (non CMake) avec Visual Studio



The screenshot shows the Visual Studio interface with the 'Add New Item' dialog box open. The dialog is titled 'Add New Item - Test' and displays a list of templates under the 'Visual C++' category. The 'C++ File (.cpp)' template is selected, and its details are shown on the right: 'Type: Visual C++' and 'Creates a file containing C++ source code'. The 'Name' field is set to 'Main.c' and the 'Location' is 'C:\Users\Administrator\Desktop\Test\'. The 'Add' button is highlighted.

Visual Studio interface showing the 'Add New Item' dialog box. The dialog is titled 'Add New Item - Test' and displays a list of templates under the 'Visual C++' category. The 'C++ File (.cpp)' template is selected, and its details are shown on the right: 'Type: Visual C++' and 'Creates a file containing C++ source code'. The 'Name' field is set to 'Main.c' and the 'Location' is 'C:\Users\Administrator\Desktop\Test\'. The 'Add' button is highlighted.

Créer un projet C simple (non CMake) avec Visual Studio



Test - Microsoft Visual Studio (Administrator)

File Edit View Project Build Debug Team Tools Architecture Test Driver Analyze Window Help

Debug x86 Local Windows Debugger Auto

Test Property Pages

Configuration: Active(Debug) Platform: Active(Win32) Configuration Manager...

- Configuration Properties
 - General
 - Debugging
 - VC++ Directories
 - C/C++
 - General
 - Optimization
 - Preprocessor
 - Code Generation
 - Language
 - Precompiled Headers
 - Output Files
 - Browse Information
 - Advanced
 - All Options
 - Command Line
 - Linker
 - Manifest Tool
 - XML Document Generator
 - Browse Information
 - Build Events
 - Custom Build Step
 - Code Analysis

General

Target Platform	Windows
Target Platform Version	8.1
Output Directory	\$(SolutionDir)\$(Configuration)\
Intermediate Directory	\$(Configuration)\
Target Name	\$(ProjectName)
Target Extension	.exe
Extensions to Delete on Clean	*.cdf;*.cache;*.obj;*.obj.enc;*.ilk;*.ipdb;*.iobj;*.resources;*.tlb;*.tli;*.t...
Build Log File	\$(IntDir)\$(MSBuildProjectName).log
Platform Toolset	Visual Studio 2015 (v140)
Enable Managed Incremental Build	No

Project Defaults

Configuration Type	Application (.exe)
Use of MFC	Use Standard Windows Libraries
Character Set	Not Set
Common Language Runtime Support	Not Set
.NET Target Framework Version	Use Unicode Character Set
Whole Program Optimization	Use Multi-Byte Character Set
Windows Store App Support	<inherit from parent or project defaults>

Character Set
Tells the compiler to use the specified character set; aids in localization issues.

OK Cancel Apply

main(int argc, char * argv[])

Server Explorer Toolbox Properties

Publish

17:03

Créer un projet C simple (non CMake) avec Visual Studio

- Bien régler les options pour toutes les **Configurations** (**Debug**, **Release**) et pour toutes les **Plateformes** (**x86**, **x64**) si nécessaire...



CMake support

- See https://www.ensta-bretagne.fr/lebars/tutorials/vs_cmake.txt



Tips Visual Studio

- Raccourcis utiles :
 - **Commenter** : sélectionner le code désiré et faire **CTRL+K,CTRL+C**
 - **Décommenter** : sélectionner le code désiré et faire **CTRL+K,CTRL+U**
 - **Auto-complétion** : **CTRL+SPACE**
 - **Indentation automatique** : sélectionner le code désiré et faire **CTRL+K,CTRL+F**
 - Passer en **majuscules** : sélectionner le code désiré et faire **CTRL+SHIFT+U**
 - Passer en **minuscules** : sélectionner le code désiré et faire **CTRL+U**

■ Raccourcis utiles :

- Right-click on a function or variable and choose **Go to Definition** to see the corresponding declaration code in the source file
- Vertical selection: **SHIFT+ALT+click** and hold the left mouse button at the start of the text you want to select, then drag the mouse down to the end of the text



Tips Visual Studio

- Utiliser les fenêtres :
 - **View > Code Definition Window**
 - **View > Error List**
 - **View > Output**

- Ouverture de projets créés avec d'autres versions de Visual Studio :
 - **Project > Retarget solution** si erreur liée à **Windows SDK Version** ou **Platform Toolset**

- Extensions :
 - **Image Watch** : voir le contenu des images OpenCV en debug
 - **VsDiff** : comparer facilement le contenu de fichiers

Tips Visual Studio

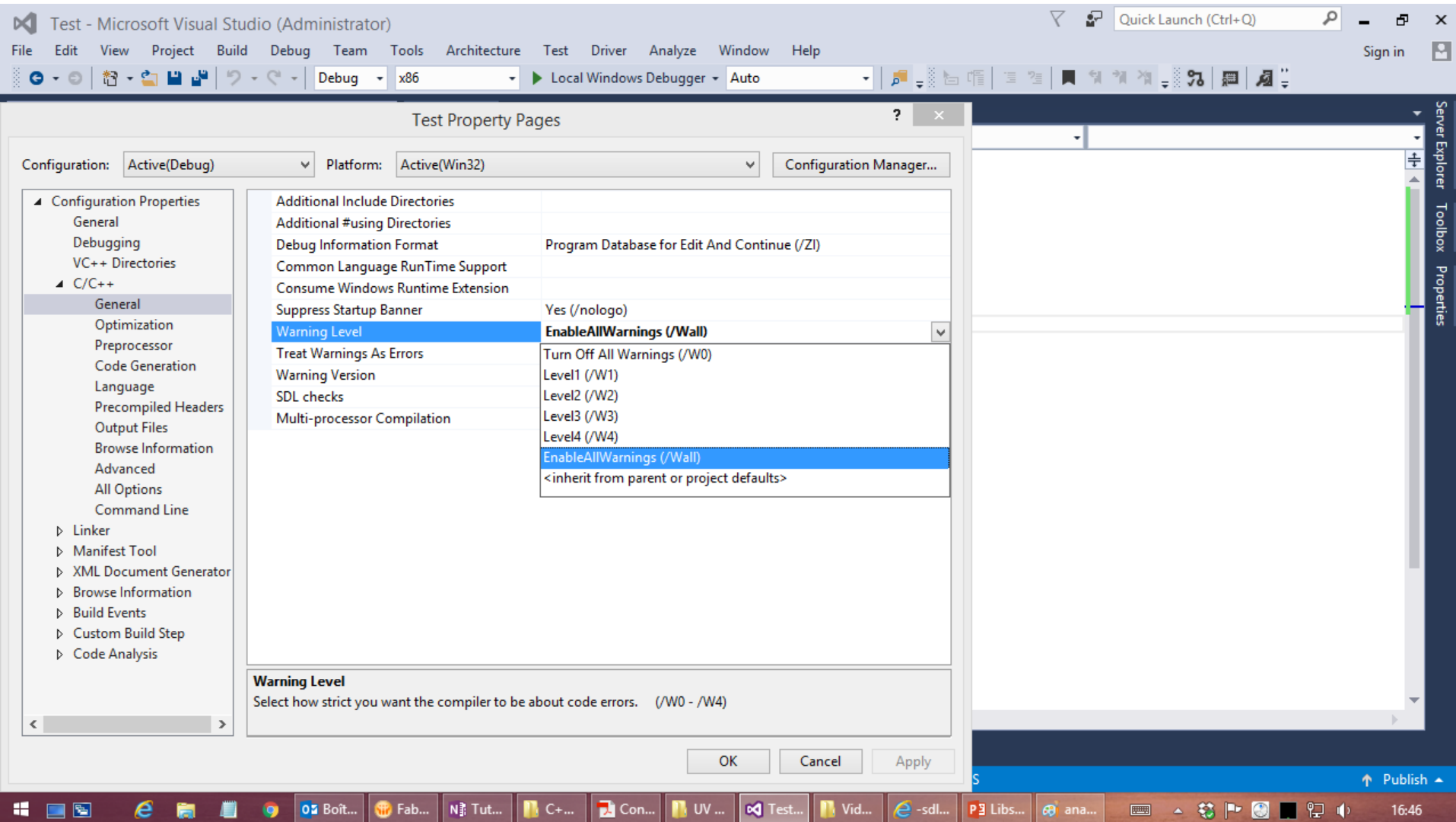
- See also <http://www.ensta-bretagne.fr/lebars/tutorials>



Débugger rapidement avec Visual Studio

- Analyse statique de code :
 - Activer tous les avertissements (**/W4** ou même **/Wall**)
 - Enable SDL checks (**/sdl**) : makes some aggressive checks and turns some warnings into errors, therefore might need to be **disabled for compatibility** with some libraries, however can be good to enable to help debugging if you can...
 - Aller dans le menu **Analyze > Run Code Analysis**
- Note : certaines options peuvent ne pas être disponibles s'il n'y a pas de fichiers C/C++ dans le projet, selon l'édition de Visual Studio (Ultimate/Enterprise recommandée), etc.

Débugger rapidement avec Visual Studio



The screenshot shows the Visual Studio interface with the 'Test Property Pages' dialog box open. The dialog is for the 'Active(Debug)' configuration on the 'Active(Win32)' platform. The 'Warning Level' property is selected, and the dropdown menu is open, showing options: 'EnableAllWarnings (/Wall)', 'Turn Off All Warnings (/W0)', 'Level1 (/W1)', 'Level2 (/W2)', 'Level3 (/W3)', 'Level4 (/W4)', 'EnableAllWarnings (/Wall)', and '<inherit from parent or project defaults>'. The 'EnableAllWarnings (/Wall)' option is highlighted. The dialog also shows other properties like 'Additional Include Directories', 'Debug Information Format', and 'Suppress Startup Banner'. The background shows the Visual Studio IDE with the 'Server Explorer', 'Toolbox', and 'Properties' windows visible.

Test Property Pages

Configuration: Active(Debug) Platform: Active(Win32) Configuration Manager...

Configuration Properties

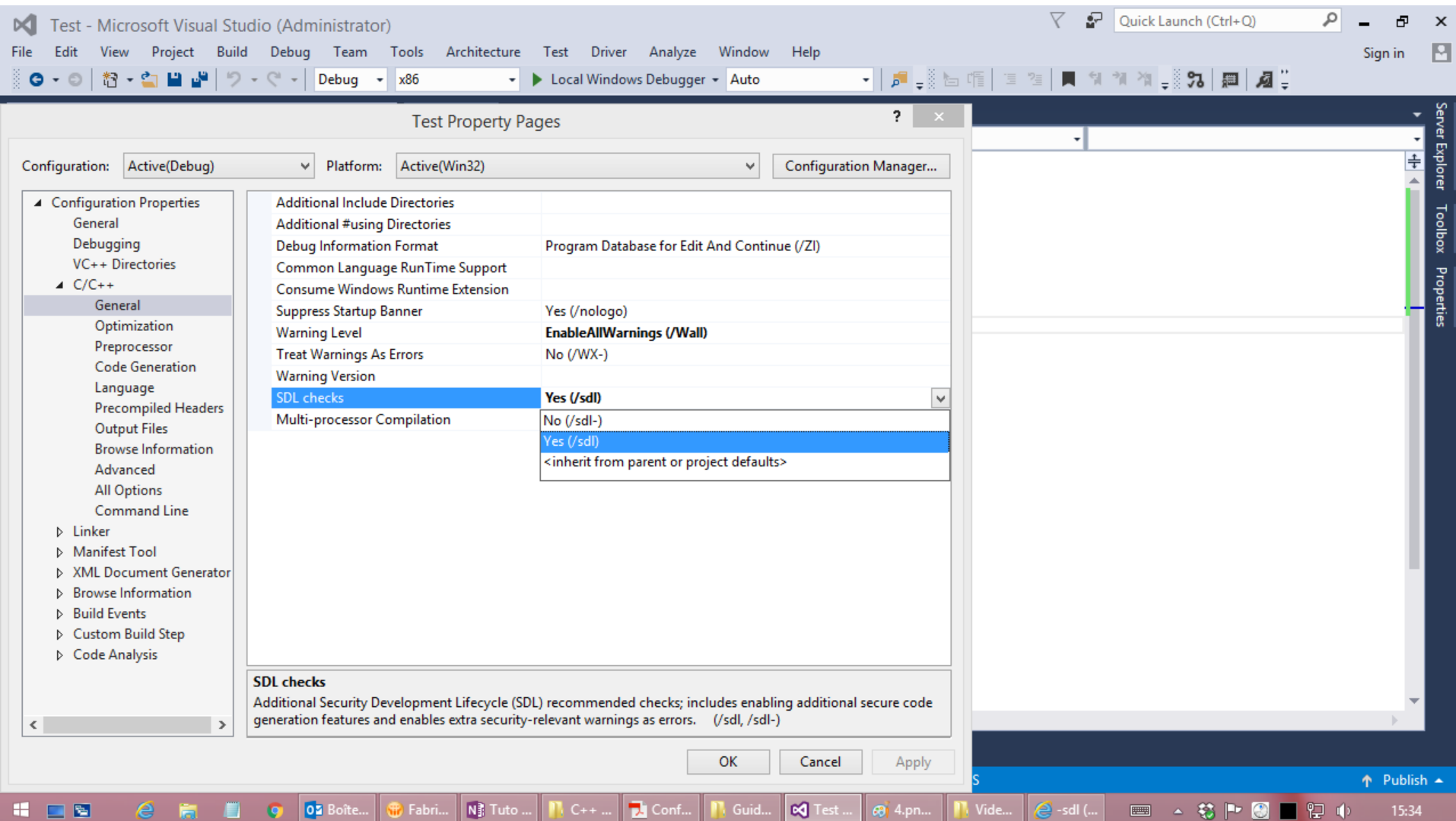
- General
- Debugging
- VC++ Directories
- C/C++
 - General
 - Optimization
 - Preprocessor
 - Code Generation
 - Language
 - Precompiled Headers
 - Output Files
 - Browse Information
 - Advanced
 - All Options
 - Command Line
- Linker
- Manifest Tool
- XML Document Generator
- Browse Information
- Build Events
- Custom Build Step
- Code Analysis

Additional Include Directories	
Additional #using Directories	
Debug Information Format	Program Database for Edit And Continue (/ZI)
Common Language RunTime Support	
Consume Windows Runtime Extension	
Suppress Startup Banner	Yes (/nologo)
Warning Level	EnableAllWarnings (/Wall)
Treat Warnings As Errors	Turn Off All Warnings (/W0)
Warning Version	Level1 (/W1)
SDL checks	Level2 (/W2)
Multi-processor Compilation	Level3 (/W3)
	Level4 (/W4)
	EnableAllWarnings (/Wall)
	<inherit from parent or project defaults>

Warning Level
Select how strict you want the compiler to be about code errors. (/W0 - /W4)

OK Cancel Apply

Débugger rapidement avec Visual Studio



The screenshot shows the Visual Studio interface with the 'Test Property Pages' dialog box open. The dialog is configured for 'Active(Debug)' and 'Active(Win32)'. The 'C/C++' section is expanded, and the 'SDL checks' property is selected. The 'SDL checks' dropdown menu is open, showing the following options:

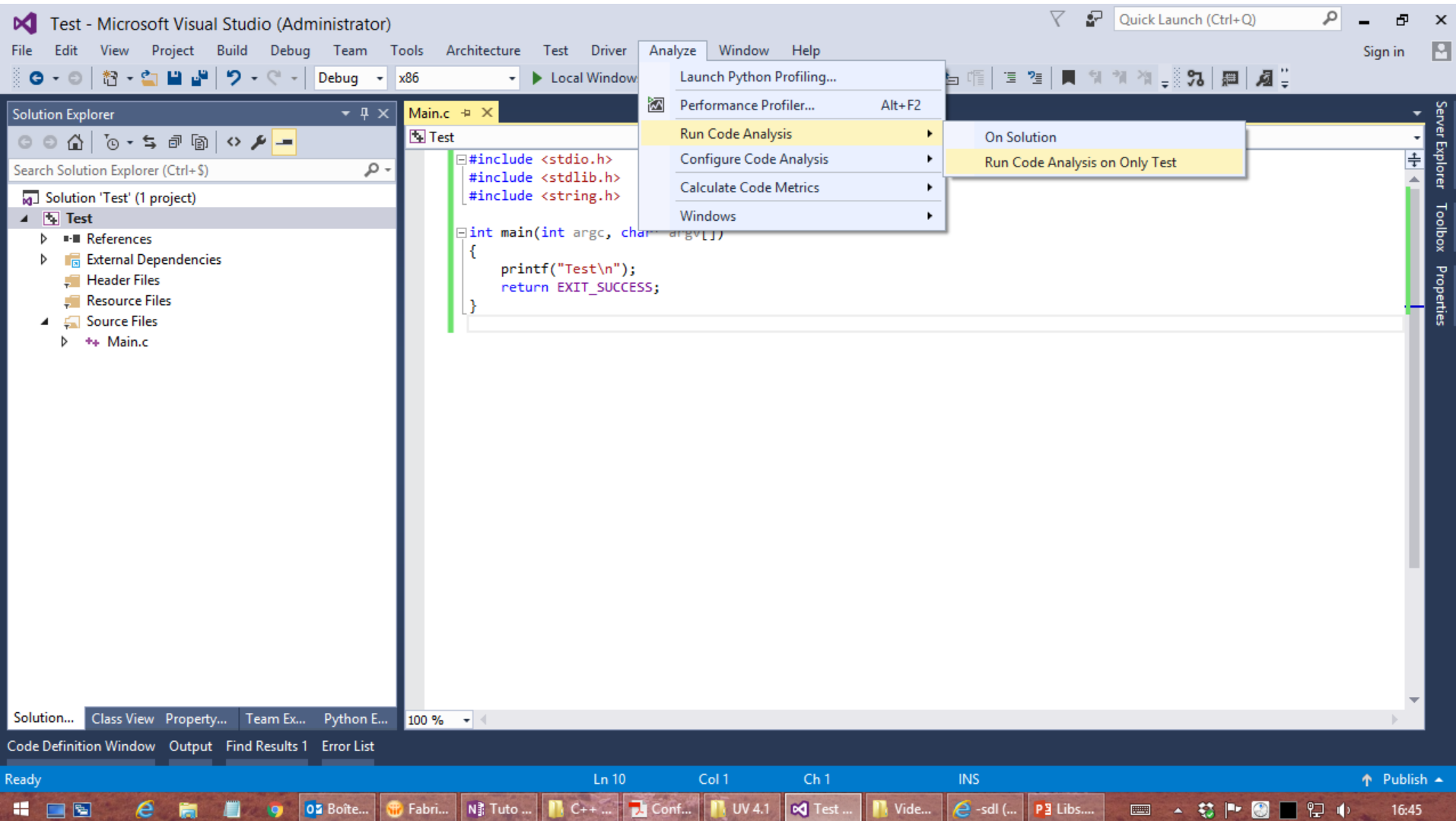
Additional Include Directories	
Additional #using Directories	
Debug Information Format	Program Database for Edit And Continue (/ZI)
Common Language RunTime Support	
Consume Windows Runtime Extension	
Suppress Startup Banner	Yes (/nologo)
Warning Level	EnableAllWarnings (/Wall)
Treat Warnings As Errors	No (/WX-)
Warning Version	
SDL checks	Yes (/sdl)
Multi-processor Compilation	No (/sdl-)
	Yes (/sdl)
	<inherit from parent or project defaults>

Below the dropdown, the 'SDL checks' section is expanded, showing the following text:

SDL checks
Additional Security Development Lifecycle (SDL) recommended checks; includes enabling additional secure code generation features and enables extra security-relevant warnings as errors. (/sdl, /sdl-)

The dialog box has 'OK', 'Cancel', and 'Apply' buttons at the bottom.

Débugger rapidement avec Visual Studio



The screenshot shows the Visual Studio interface with the 'Analyze' menu open. The 'Run Code Analysis on Only Test' option is highlighted. The code editor displays the following C code:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(int argc, char *argv[])
{
    printf("Test\n");
    return EXIT_SUCCESS;
}
```

The Solution Explorer on the left shows a project named 'Test' with a source file 'Main.c'. The status bar at the bottom indicates 'Ready' and '16:45'.

Débugger rapidement avec Visual Studio

■ Analyse dynamique de code :

• Rajouter

```
#define _CRTDBG_MAP_ALLOC
#include <stdlib.h>
#include <crtdbg.h>
```

et appeler en début de main() :

```
_CrtSetDbgFlag(_CRTDBG_ALLOC_MEM_DF |
_CRTDBG_LEAK_CHECK_DF);
```

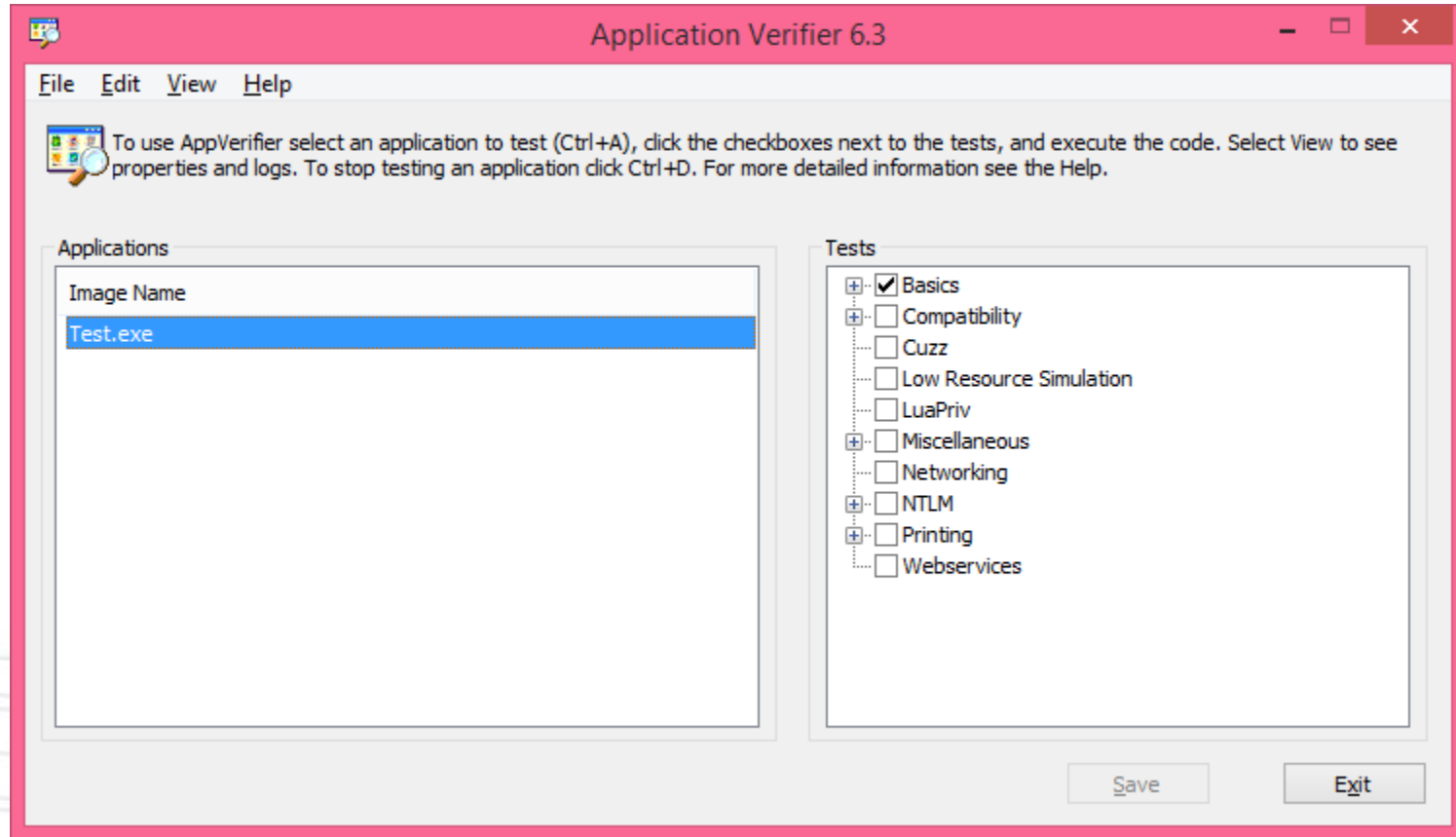
pour obtenir des infos sur les fuites mémoires dans la fenêtre

Output > Debug

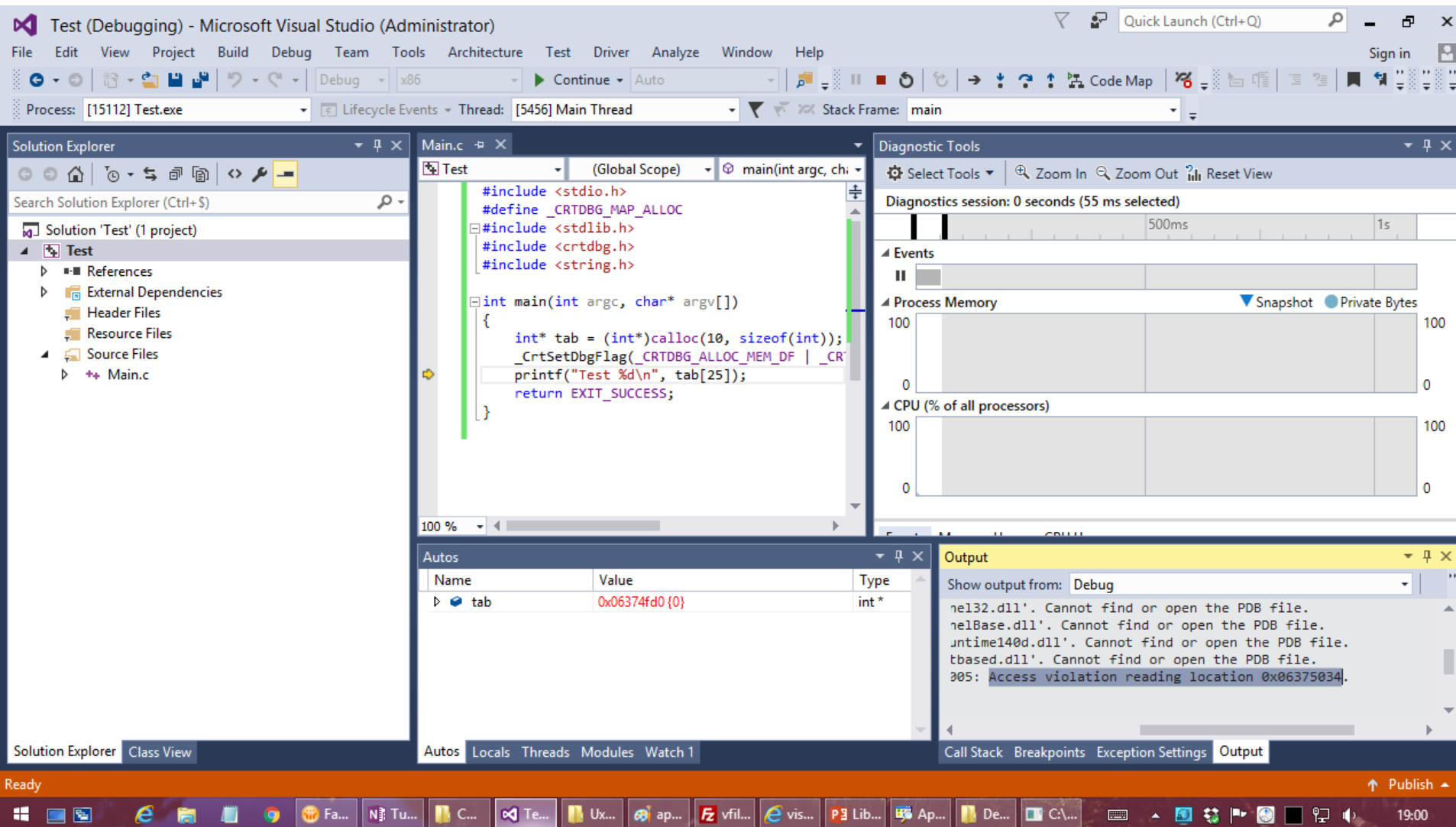
- Installer **Application Verifier**, le lancer et rajouter son application à sa liste, ainsi il l'analysera la prochaine fois qu'on la lancera avec le debugger de Visual Studio

- Analyse dynamique de code :
 - Exemple : programme Test.exe qui **alloue un tableau de 10 entiers**, essaye d'**afficher la valeur du 25^{ème}** et **s'arrête sans libérer la mémoire** allouée
 - Remarque : si on exécute le programme en dehors d'un debugger et sans la surveillance d'Application Verifier, il est très probable qu'il ne plante pas... Par contre, si on prend une valeur significativement plus grande que 25, il est probable que la tentative de lecture de la valeur soit dans une zone mémoire en dehors de celle réservée à notre programme, ce que l'OS va détecter et forcera alors l'arrêt immédiat du programme (pour des raisons de sécurité)...

Débugger rapidement avec Visual Studio



Débugger rapidement avec Visual Studio



The screenshot shows the Visual Studio IDE in a debugging state. The main window displays the source code for `Main.c` with a breakpoint set at the `printf` statement. The `Autos` window shows the variable `tab` with a value of `0x06374fd0 (0)`. The `Output` window displays a message box error: `305: Access violation reading location 0x06375034.` The `Diagnostic Tools` window is also visible, showing the process memory and CPU usage.

Test (Debugging) - Microsoft Visual Studio (Administrator)

File Edit View Project Build Debug Team Tools Architecture Test Driver Analyze Window Help

Process: [15112] Test.exe Lifecycle Events Thread: [5456] Main Thread Stack Frame: main

Solution Explorer

Search Solution Explorer (Ctrl+S)

Solution 'Test' (1 project)

- Test
 - References
 - External Dependencies
 - Header Files
 - Resource Files
 - Source Files
 - Main.c

Main.c

```
#include <stdio.h>
#define _CRTDBG_MAP_ALLOC
#include <stdlib.h>
#include <crtdbg.h>
#include <string.h>

int main(int argc, char* argv[])
{
    int* tab = (int*)calloc(10, sizeof(int));
    _CrtSetDbgFlag(_CRTDBG_ALLOC_MEM_DF | _CR
    printf("Test %d\n", tab[25]);
    return EXIT_SUCCESS;
}
```

Diagnostic Tools

Select Tools Zoom In Zoom Out Reset View

Diagnostics session: 0 seconds (55 ms selected)

Events

Process Memory Snapshot Private Bytes

CPU (% of all processors)

Autos

Name	Value	Type
tab	0x06374fd0 (0)	int*

Output

Show output from: Debug

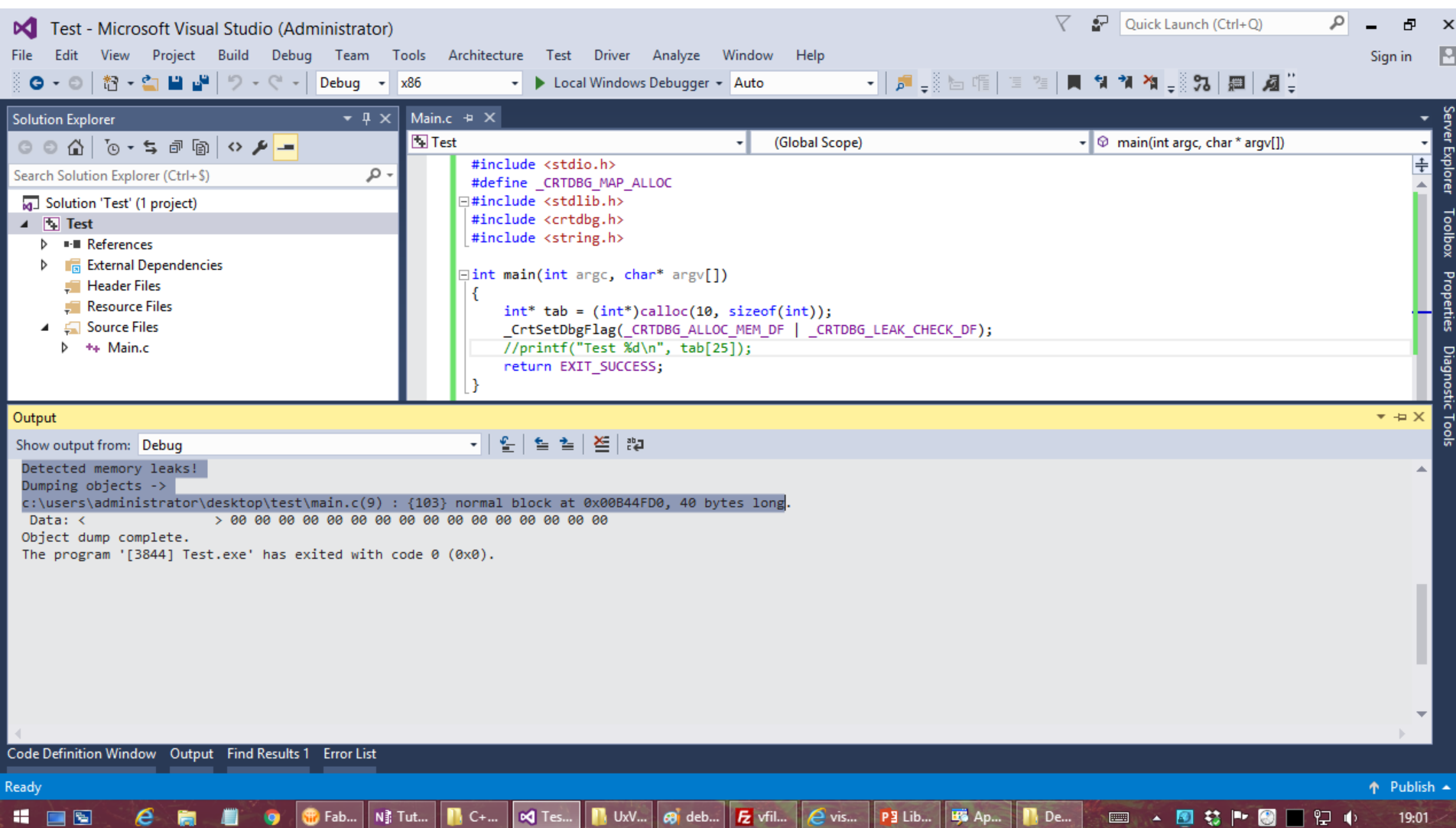
```
nel32.dll'. Cannot find or open the PDB file.
nelbase.dll'. Cannot find or open the PDB file.
untime140d.dll'. Cannot find or open the PDB file.
tbased.dll'. Cannot find or open the PDB file.
305: Access violation reading location 0x06375034.
```

Solution Explorer Class View Autos Locals Threads Modules Watch 1 Call Stack Breakpoints Exception Settings Output

Ready Publish

19:00

Débugger rapidement avec Visual Studio



The screenshot shows the Visual Studio IDE with a C program named 'Main.c' open. The code includes `<stdio.h>`, `<stdlib.h>`, `<crtdbg.h>`, and `<string.h>`. The `main` function calls `calloc(10, sizeof(int))` to allocate memory, prints "Test %d\n", and returns `EXIT_SUCCESS`. The debugger output window shows a detected memory leak at line 9 of `main.c`, with a dump of the memory block at `0x00B44FD0`. The program exited with code 0 (0x0).

```
#include <stdio.h>
#define _CRTDBG_MAP_ALLOC
#include <stdlib.h>
#include <crtdbg.h>
#include <string.h>

int main(int argc, char* argv[])
{
    int* tab = (int*)calloc(10, sizeof(int));
    _CrtSetDbgFlag(_CRTDBG_ALLOC_MEM_DF | _CRTDBG_LEAK_CHECK_DF);
    //printf("Test %d\n", tab[25]);
    return EXIT_SUCCESS;
}
```

Output window content:

```
Detected memory leaks!
Dumping objects ->
c:\users\administrator\Desktop\test\main.c(9) : {103} normal block at 0x00B44FD0, 40 bytes long.
Data: < > 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Object dump complete.
The program '[3844] Test.exe' has exited with code 0 (0x0).
```

Débugger rapidement avec Visual Studio

- Analyse dynamique de code :
 - On vérifie bien que dans l'exemple
 $0x06375034 - 0x06374fd0 = 0x64 \rightarrow 100 \rightarrow 25 * \text{sizeof}(\text{int})$
 $0x06375034$ étant l'adresse qui a posé problème et $0x06374fd0$ le début du tableau d'entiers
 - Cependant, il semblerait que certains accès en dehors de la mémoire ne soient pas toujours détectés...



Equivalents sous Linux

- Analyse statique de code :
 - Activer tous les avertissements et autres options de debug (**-g -Wall -Wextra -Winline**)
 - Voir aussi **-pedantic, -fpermissive**



Equivalents sous Linux

- Analyse dynamique de code :
 - Installer **valgrind** (principalement pour voir les fuites mémoire)
 - Rajouter

```
#include <valgrind/memcheck.h>
```

et appeler la macro

```
VALGRIND_DO_LEAK_CHECK;
```

à plusieurs endroits pour comparer et tracer plus précisément les opérations sur la mémoire
 - **valgrind --leak-check=full -v ./a.out**
 - Consulter la ligne **in use at exit** pour déterminer s'il y a des fuites mémoires en fin d'exécution

Equivalents sous Linux

- Analyse dynamique de code :
 - Installer **electric-fence** (pour détecter les accès mémoire erronés)

Exécuter **a.out** de cette manière

```
LD_PRELOAD=libefence.so.0.0 ./a.out
```

Si crash, analyser le fichier **core** (penser à effacer les éventuels précédents avant exécution) avec

```
gdb ./a.out core
```

Si le fichier **core** n'est pas créé, réessayer avec

```
ulimit -c unlimited
```



Visual Studio Code

Visual Studio Code

- **Visual Studio Code est significativement différent de Visual Studio (Community, Pro, Enterprise, etc.)!**
 - Open source
 - Pas de compilateur fourni
 - Des extensions doivent être installées pour avoir des outils d'aide à la programmation en C/C++ similaires à Visual Studio (e.g. suggestions, informations sur les variables et fonctions, etc.)
 - Fonctionne sous Windows, Linux, macOS



Visual Studio Code

- Visual Studio Code
 - Extensions souvent utiles
 - ms-vscode.cpptools-extension-pack
 - ms-vscode.hexeditor



Visual Studio Code

■ Visual Studio Code CMake support

- Prérequis

Avoir installé un compilateur et CMake :

Ubuntu 24.04 :

```
sudo apt -y install build-essential cmake-gui
```

Windows : install [Chocolatey package manager](#) and then

```
choco install -y mingw --version=13.2.0
```

```
choco install -y cmake.install --installargs 'ADD_CMAKE_TO_PATH=System'
```

- Installation de Visual Studio Code : <https://code.visualstudio.com/>
- La commande pour le lancer est souvent **code**
- Installer l'extension **ms-vscode.cpptools-extension-pack**

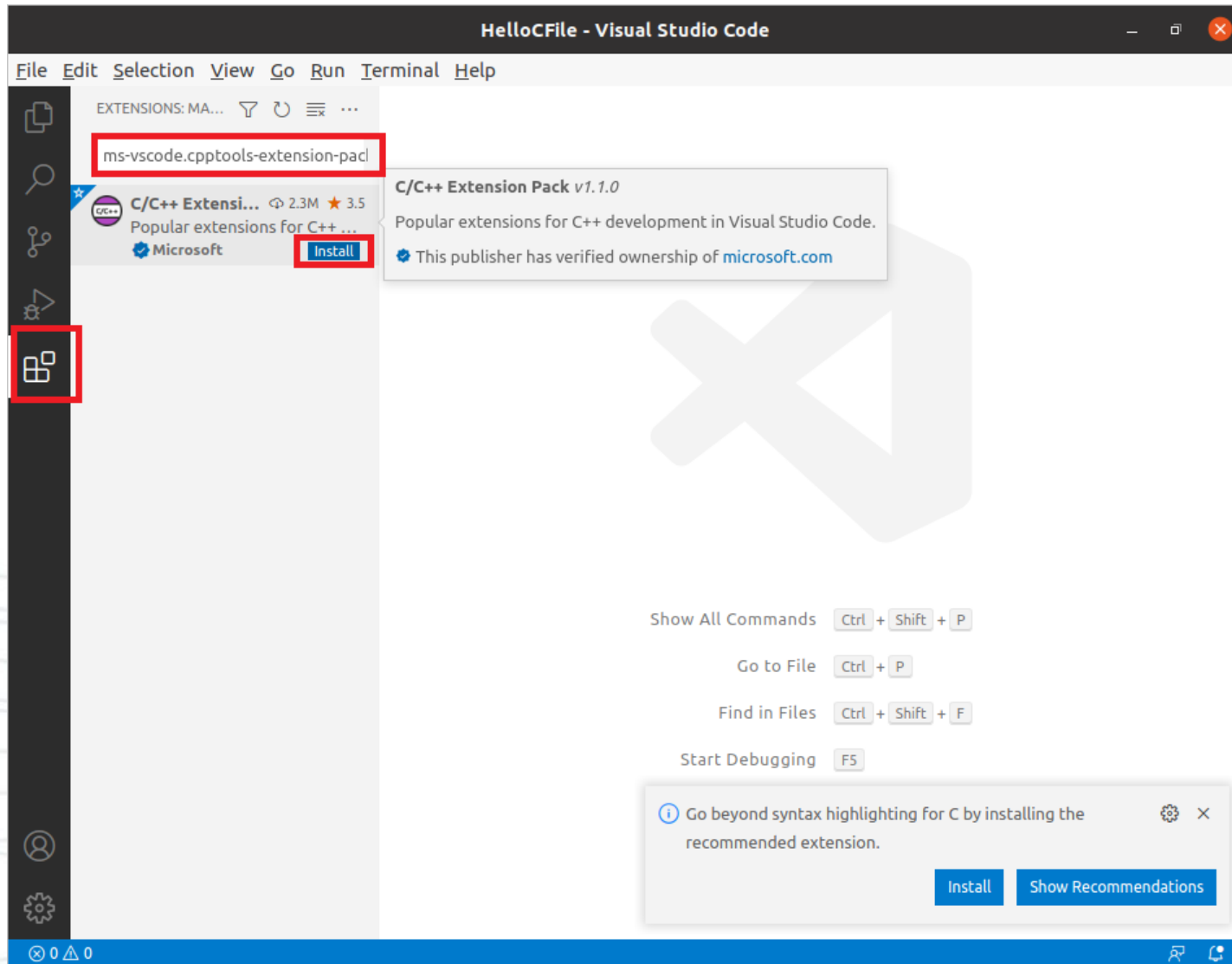
■ Visual Studio Code CMake support

- Tips

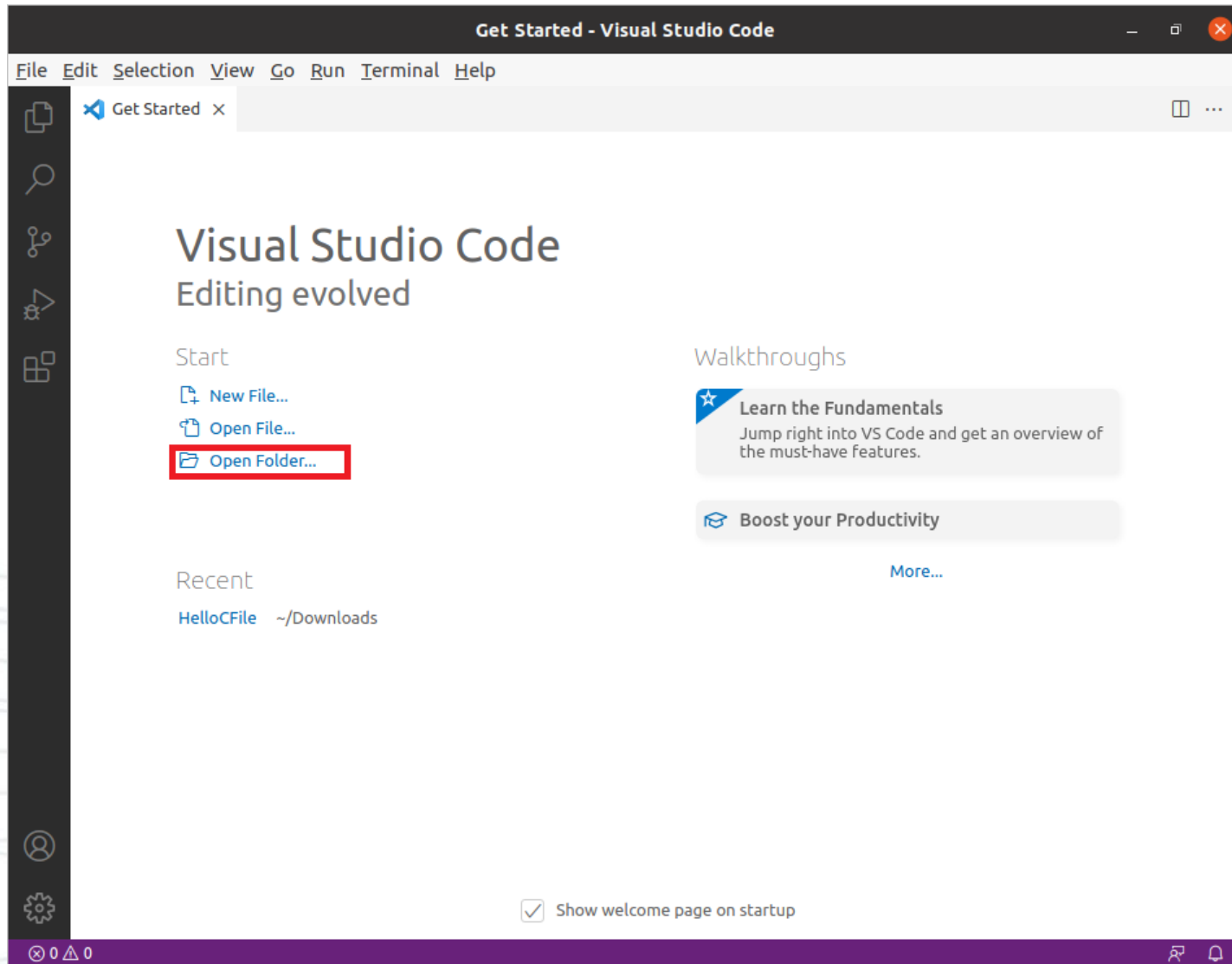
Pour changer le dossier courant qui est par défaut `"${workspaceFolder}/build"` lors de l'exécution en mode débogage, créer dans le dossier du projet **`.vscode/settings.json`** avec dedans e.g. (si le programme à déboguer cherche à ouvrir des fichiers dans le dossier où se trouve `CMakeLists.txt`) :

```
{  
    "cmake.debugConfig": {  
        "cwd": "${workspaceFolder}"  
    }  
}
```

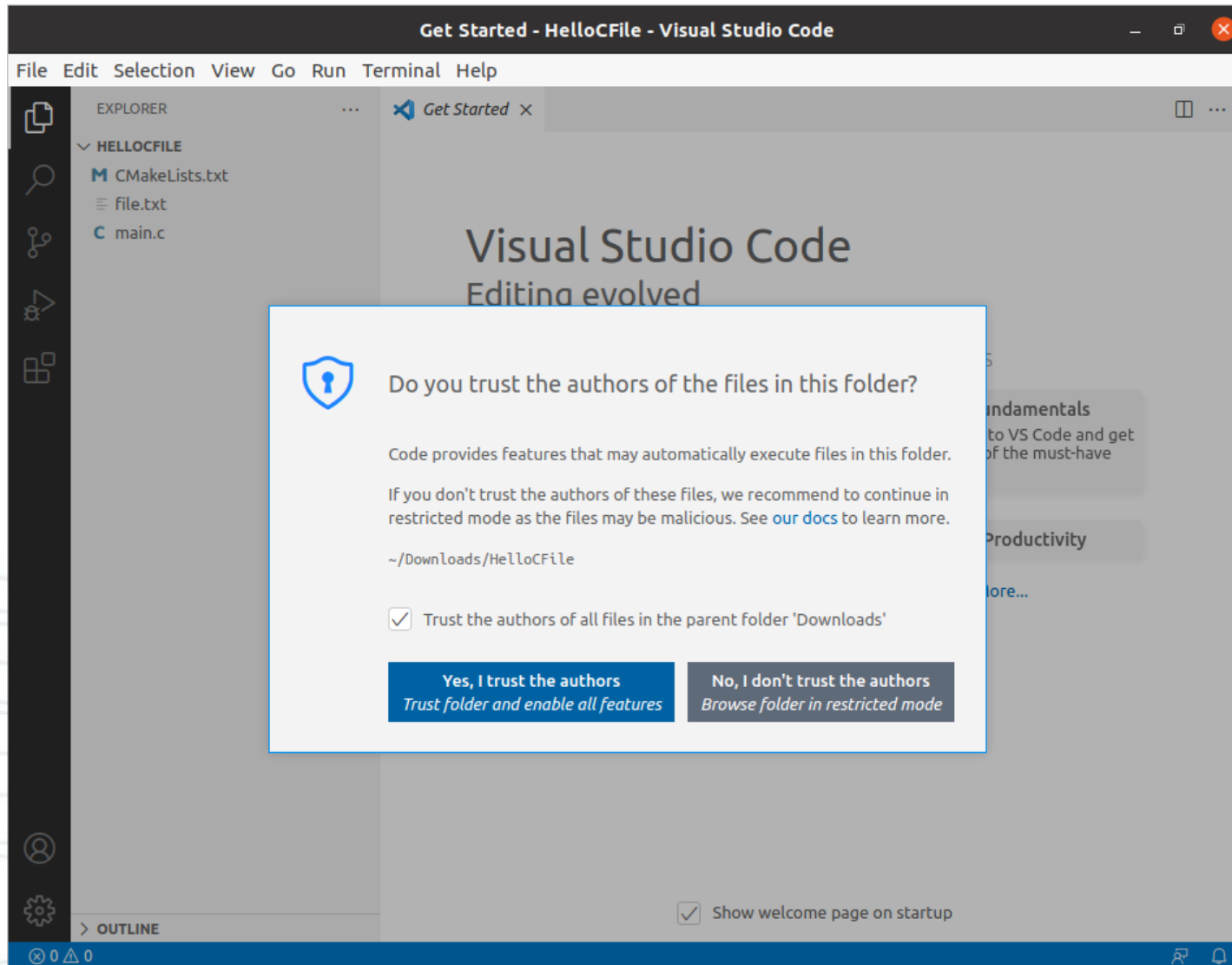
Visual Studio Code



Visual Studio Code



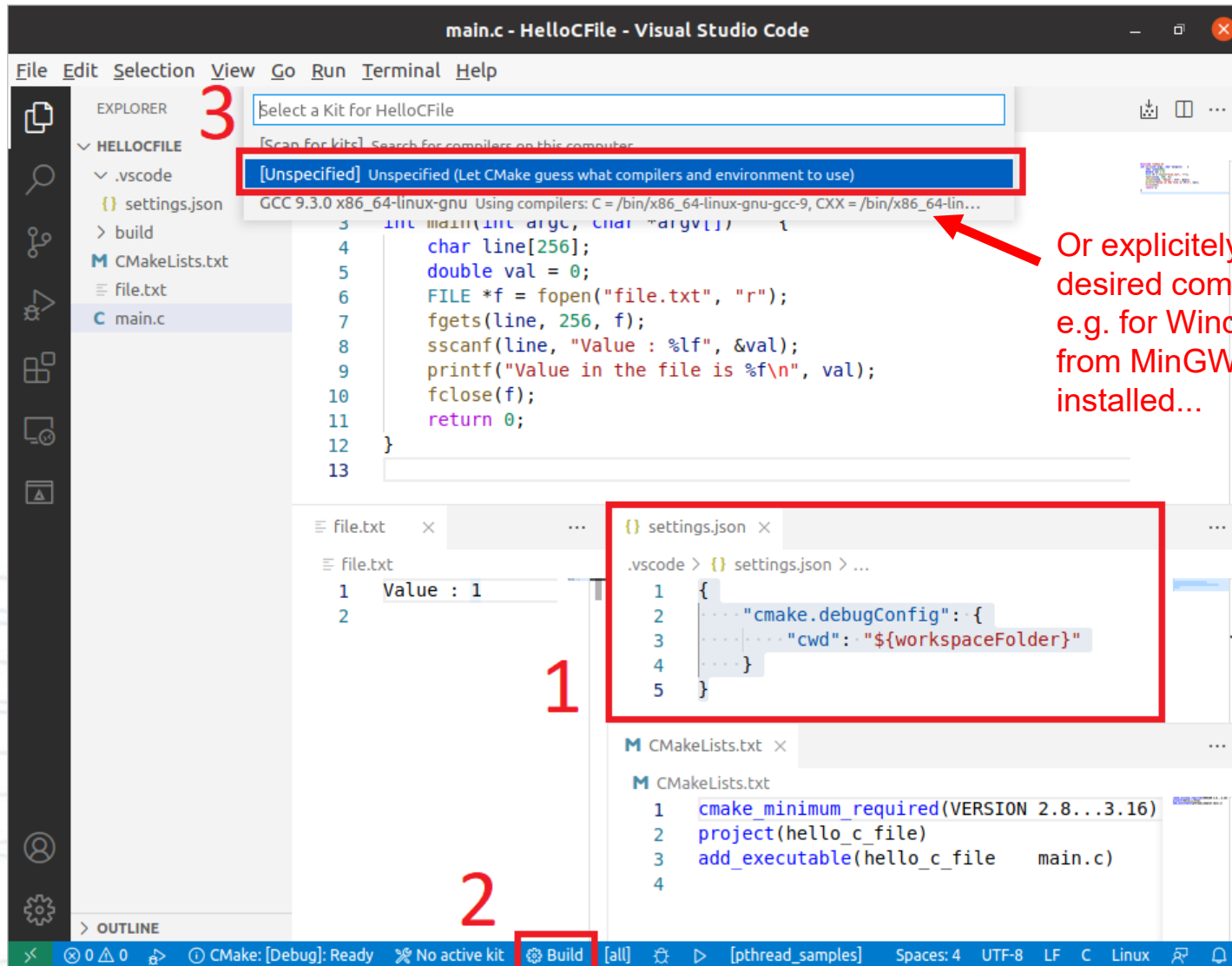
Visual Studio Code



Visual Studio Code

```
user@Ubuntu64: ~/Downloads/HelloCFile
user@Ubuntu64:~/Downloads/HelloCFile$ code
user@Ubuntu64:~/Downloads/HelloCFile$ mkdir -p .vscode
user@Ubuntu64:~/Downloads/HelloCFile$ touch .vscode/settings.json
user@Ubuntu64:~/Downloads/HelloCFile$
```

Visual Studio Code



main.c - HelloCFile - Visual Studio Code

File Edit Selection View Go Run Terminal Help

3 select a Kit for HelloCFile

[Unspecified] Unspecified (Let CMake guess what compilers and environment to use)

GCC 9.3.0 x86_64-linux-gnu Using compilers: C = /bin/x86_64-linux-gnu-gcc-9, CXX = /bin/x86_64-lin...

```
1 int main(int argc, char *argv[]) {
2
3
4     char line[256];
5     double val = 0;
6     FILE *f = fopen("file.txt", "r");
7     fgets(line, 256, f);
8     sscanf(line, "Value : %lf", &val);
9     printf("Value in the file is %f\n", val);
10    fclose(f);
11    return 0;
12 }
13
```

1

```
{
1  "cmake.debugConfig": {
2  "cwd": "${workspaceFolder}"
3  }
4  }
```

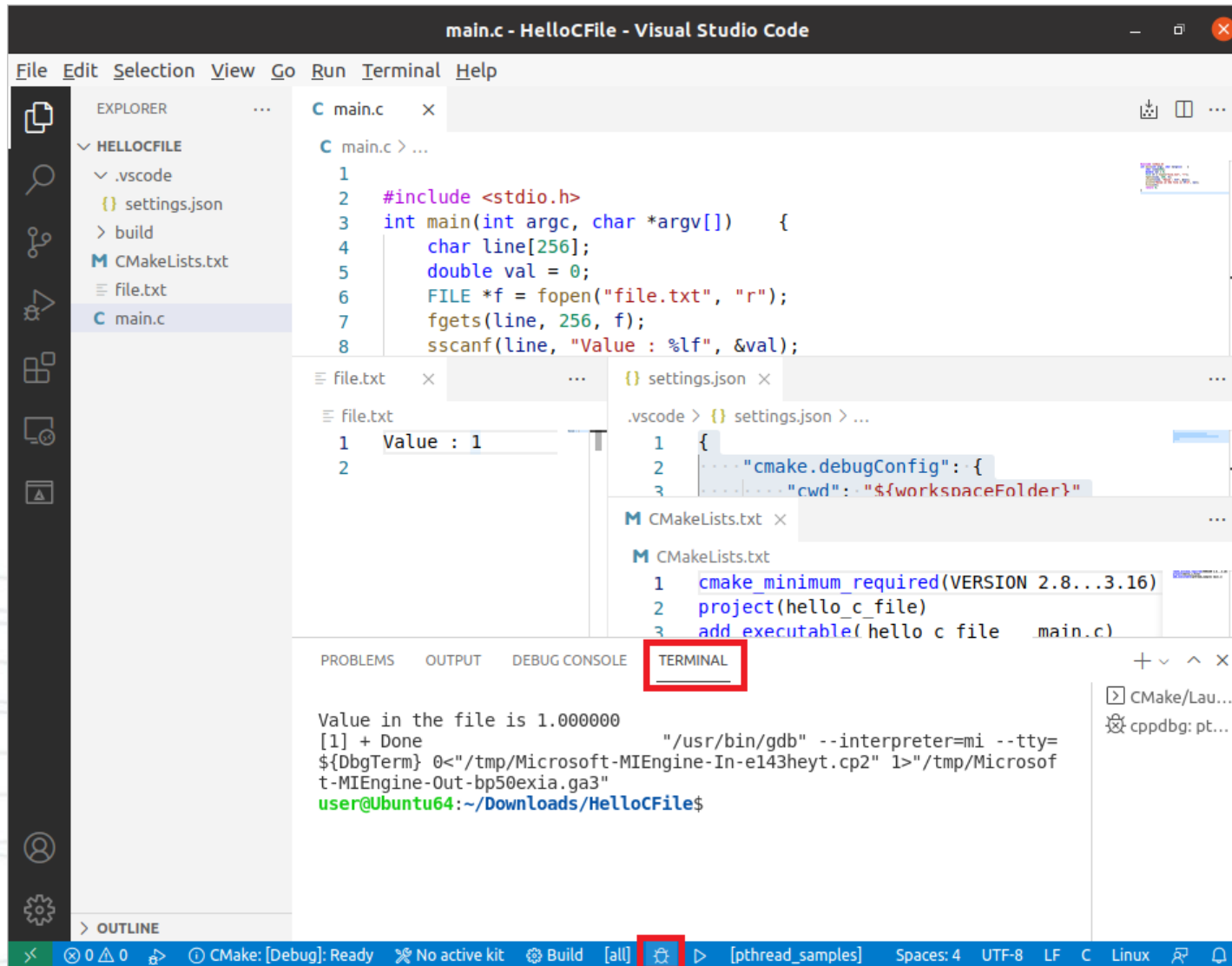
2

```
1 cmake_minimum_required(VERSION 2.8...3.16)
2 project(hello_c_file)
3 add_executable(hello_c_file main.c)
4
```

Or explicitly choose the desired compiler in the list, e.g. for Windows the GCC from MinGW previously installed...

2 Build [all] [pthread_samples] Spaces: 4 UTF-8 LF C Linux

Visual Studio Code



The screenshot displays the Visual Studio Code interface with the following components:

- EXPLORER:** Shows the project structure for 'HELLOCFILE', including files like `settings.json`, `build`, `CMakeLists.txt`, `file.txt`, and `main.c`.
- EDITOR:** Displays the source code for `main.c` and `CMakeLists.txt`.

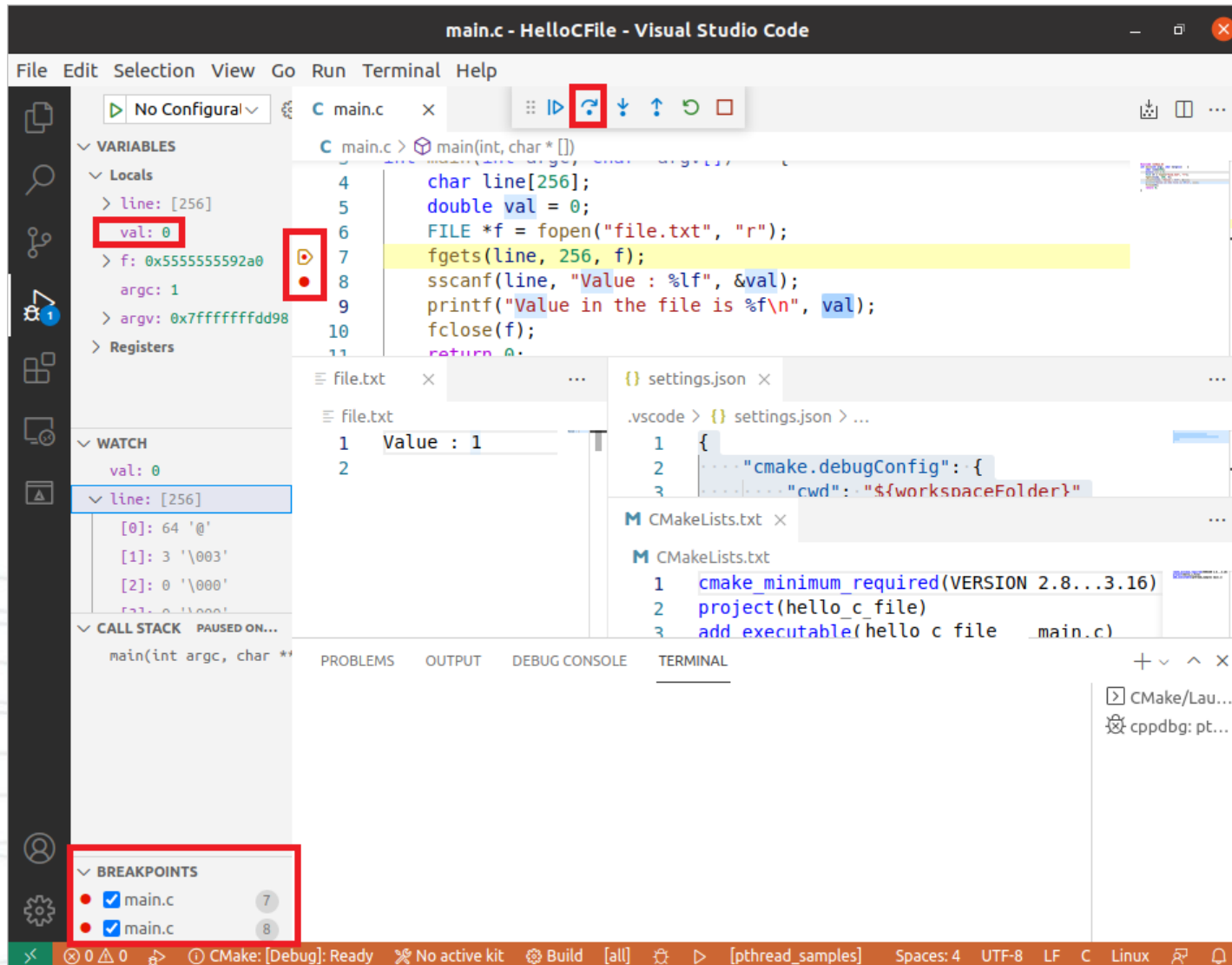
```
C main.c > ...
1
2 #include <stdio.h>
3 int main(int argc, char *argv[]) {
4     char line[256];
5     double val = 0;
6     FILE *f = fopen("file.txt", "r");
7     fgets(line, 256, f);
8     sscanf(line, "Value : %lf", &val);

.settings.json > ...
1 {
2     ... "cmake.debugConfig": {
3     ... "cwd": "${workspaceFolder}"

CMakeLists.txt > ...
1 cmake_minimum_required(VERSION 2.8...3.16)
2 project(hello_c_file)
3 add_executable(hello_c_file main.c)
```
- TERMINAL:** Shows the execution output:

```
Value in the file is 1.000000
[1] + Done          "/usr/bin/gdb" --interpreter=mi --tty=
${DbgTerm} 0<"/tmp/Microsoft-MIEngine-In-e143heyt.cp2" 1>"/tmp/Microsof
t-MIEngine-Out-bp50exia.ga3"
user@Ubuntu64:~/Downloads/HelloCFile$
```
- STATUS BAR:** Shows the current build configuration: `CMake: [Debug]: Ready`, `No active kit`, `Build`, `[all]`, `[pthread_samples]`, `Spaces: 4`, `UTF-8`, `LF`, `C`, `Linux`.

Visual Studio Code



main.c - HelloCFile - Visual Studio Code

File Edit Selection View Go Run Terminal Help

No Configur... C main.c x

VARIABLES

- Locals
 - line: [256]
 - val: 0
 - f: 0x5555555592a0
 - argc: 1
 - argv: 0x7fffffffdd98
 - Registers
- WATCH
 - val: 0
 - line: [256]
 - [0]: 64 '0'
 - [1]: 3 '\003'
 - [2]: 0 '\000'
- CALL STACK PAUSED ON...
 - main(int argc, char **
- BREAKPOINTS
 - main.c 7
 - main.c 8

```
C main.c > main(int, char * [])
4 char line[256];
5 double val = 0;
6 FILE *f = fopen("file.txt", "r");
7 fgets(line, 256, f);
8 sscanf(line, "Value : %lf", &val);
9 printf("Value in the file is %f\n", val);
10 fclose(f);
11 return 0;
```

file.txt x

```
1 Value : 1
2
```

settings.json x

```
1 {
2   ... "cmake.debugConfig": {
3     ... "cwd": "${workspaceFolder}"
```

CMakeLists.txt x

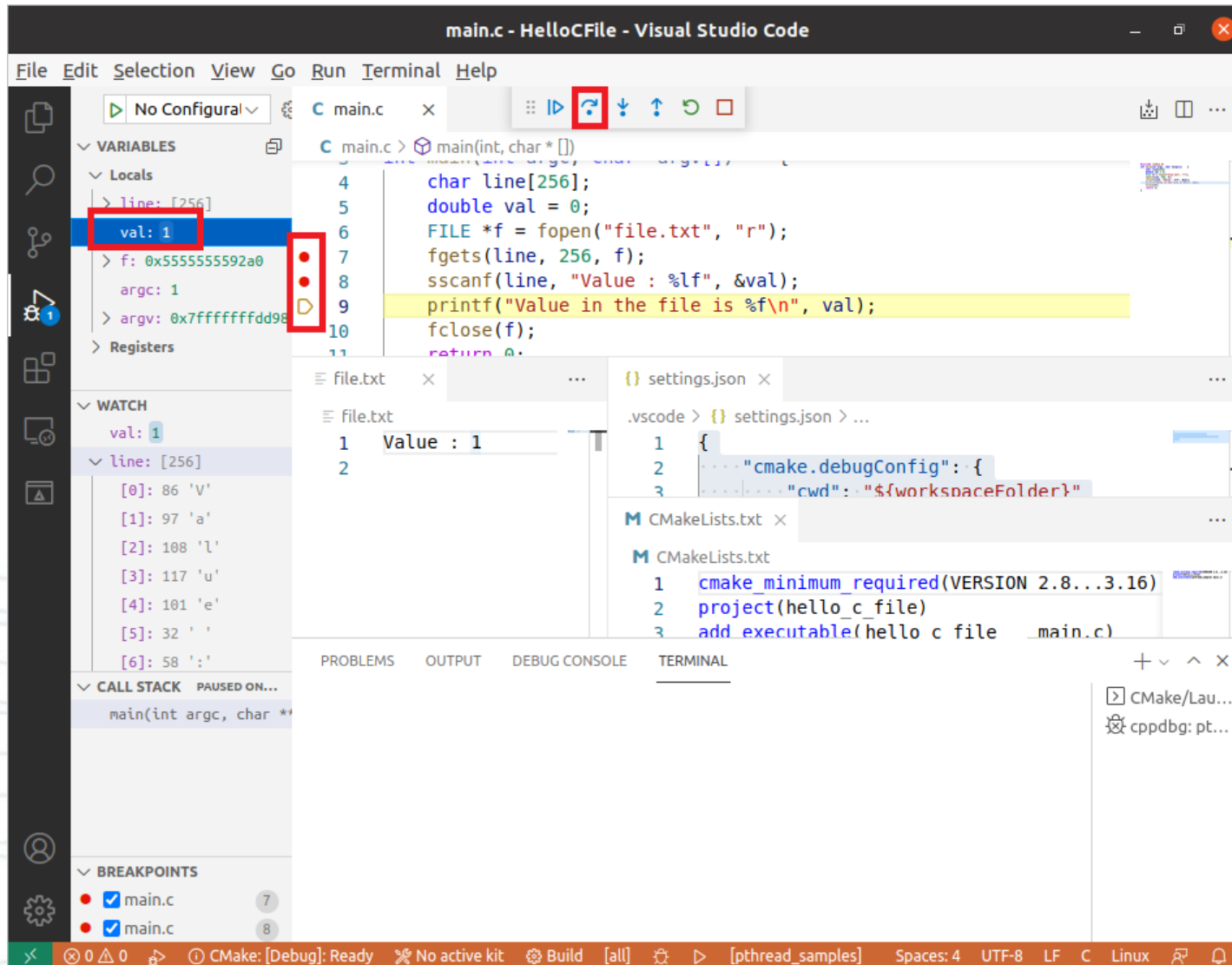
```
1 cmake_minimum_required(VERSION 2.8...3.16)
2 project(hello_c_file)
3 add_executable(hello_c_file main.c)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

CMake/Lau...
cppdbg: pt...

CMake: [Debug]: Ready No active kit Build [all] [pthread_samples] Spaces: 4 UTF-8 LF C Linux

Visual Studio Code



The screenshot displays the Visual Studio Code interface during a C program debug session. The main window shows the source code for `main.c` with the following content:

```
1 int main(int argc, char * argv[]) {
2     FILE *f;
3     char line[256];
4     double val = 0;
5     FILE *f = fopen("file.txt", "r");
6     fgets(line, 256, f);
7     sscanf(line, "Value : %lf", &val);
8     printf("Value in the file is %f\n", val);
9     fclose(f);
10    return 0;
11 }
```

The debugger is currently paused at line 8. The **VARIABLES** pane on the left shows the following state:

- Locals:**
 - `line: [256]`
 - `val: 1` (highlighted with a red box)
 - `f: 0x5555555592a0`
 - `argc: 1`
 - `argv: 0x7fffffffdd98`
- Registers:** (empty)

The **WATCH** pane shows the memory address of `val` and its contents:

```
val: 1
line: [256]
[0]: 86 'V'
[1]: 97 'a'
[2]: 108 'l'
[3]: 117 'u'
[4]: 101 'e'
[5]: 32 ' '
[6]: 58 ':'
```

The **OUTPUT** pane shows the program's output:

```
1 Value : 1
2
```

The **TERMINAL** pane shows the CMake configuration and build process:

```
.vscode > {} settings.json > ...
1 {
2   ... "cmake.debugConfig": {
3     ... "cwd": "${workspaceFolder}"
```

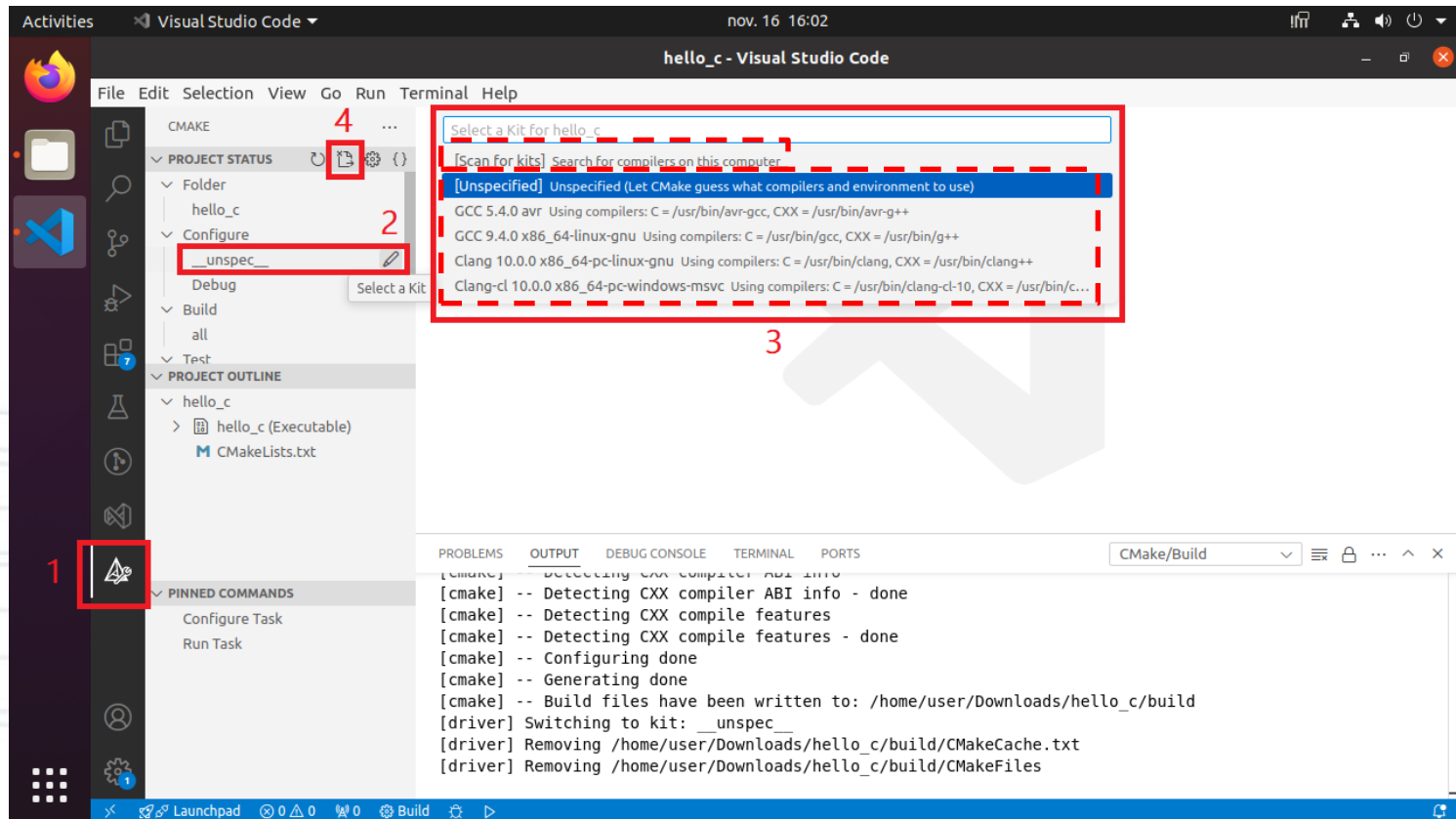
```
M CMakeLists.txt > ...
M CMakeLists.txt
1 cmake_minimum_required(VERSION 2.8...3.16)
2 project(hello_c_file)
3 add_executable(hello_c_file main.c)
```

The status bar at the bottom indicates the current configuration: `CMake: [Debug]: Ready`, `No active kit`, `Build [all]`, `[pthread_samples]`, `Spaces: 4`, `UTF-8`, `LF`, `C`, `Linux`.

■ Visual Studio Code CMake support

- Tips

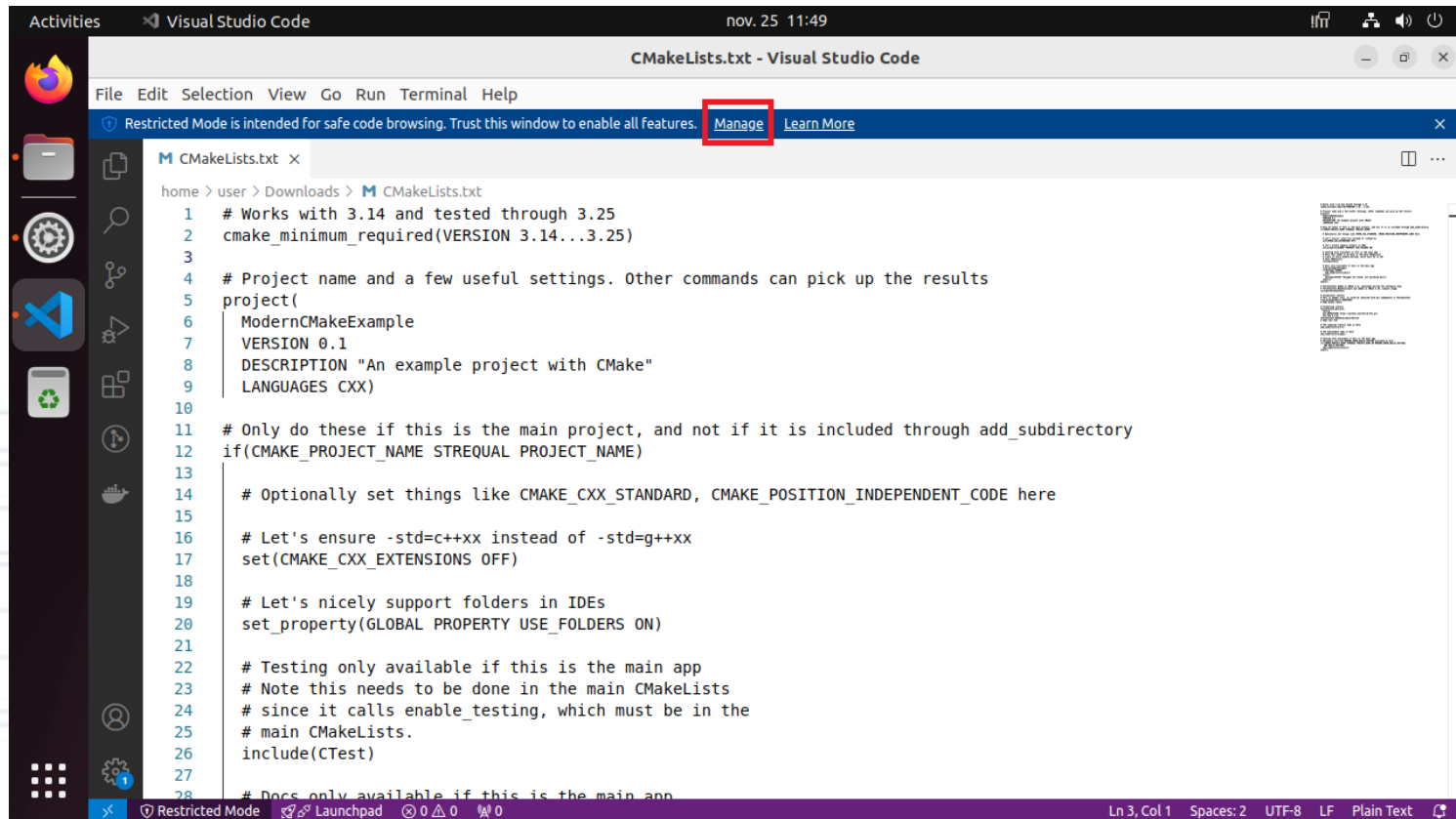
Changing the compiler used by CMake:



■ Visual Studio Code CMake support

- Tips

Check the **Restricted Mode** bar if e.g. missing syntax highlighting:



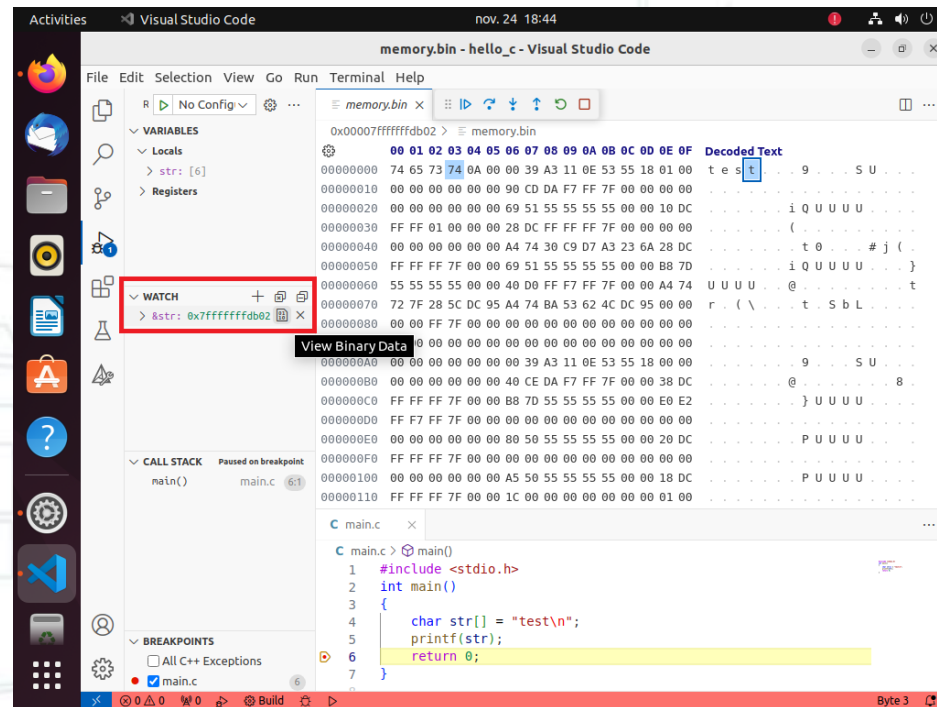
The screenshot shows the Visual Studio Code interface with a CMakeLists.txt file open. A blue bar at the top of the editor area contains the text: "Restricted Mode is intended for safe code browsing. Trust this window to enable all features." The word "Manage" is highlighted with a red box. The code in the editor is as follows:

```
1 # Works with 3.14 and tested through 3.25
2 cmake_minimum_required(VERSION 3.14...3.25)
3
4 # Project name and a few useful settings. Other commands can pick up the results
5 project(
6   ModernCMakeExample
7   VERSION 0.1
8   DESCRIPTION "An example project with CMake"
9   LANGUAGES CXX)
10
11 # Only do these if this is the main project, and not if it is included through add_subdirectory
12 if(CMAKE_PROJECT_NAME STREQUAL PROJECT_NAME)
13
14   # Optionally set things like CMAKE_CXX_STANDARD, CMAKE_POSITION_INDEPENDENT_CODE here
15
16   # Let's ensure -std=c++xx instead of -std=g++xx
17   set(CMAKE_CXX_EXTENSIONS OFF)
18
19   # Let's nicely support folders in IDEs
20   set_property(GLOBAL PROPERTY USE_FOLDERS ON)
21
22   # Testing only available if this is the main app
23   # Note this needs to be done in the main CMakeLists
24   # since it calls enable_testing, which must be in the
25   # main CMakeLists.
26   include(CTest)
27
28   # Docs only available if this is the main app
```

■ Visual Studio Code CMake support

- Tips

Memory view: see <https://stackoverflow.com/a/77476922> (add **&var** to the **WATCH** panel, then click the binary icon following the address, need **ms-vscode.hexeditor** extension)



Visual Studio Code

- Visual Studio Code CMake support

- Tips

- Right-click on a function or variable and choose **Go to Definition** to see the corresponding declaration code in the source file

- Vertical selection: **SHIFT+ALT+click** and hold the left mouse button at the start of the text you want to select, then drag the mouse down to the end of the text





Utilisation de code existant trouvé sur Internet

Utilisation de code existant trouvé sur Internet

■ Questions à se poser

• Cas particulier : code existant sur GitHub :

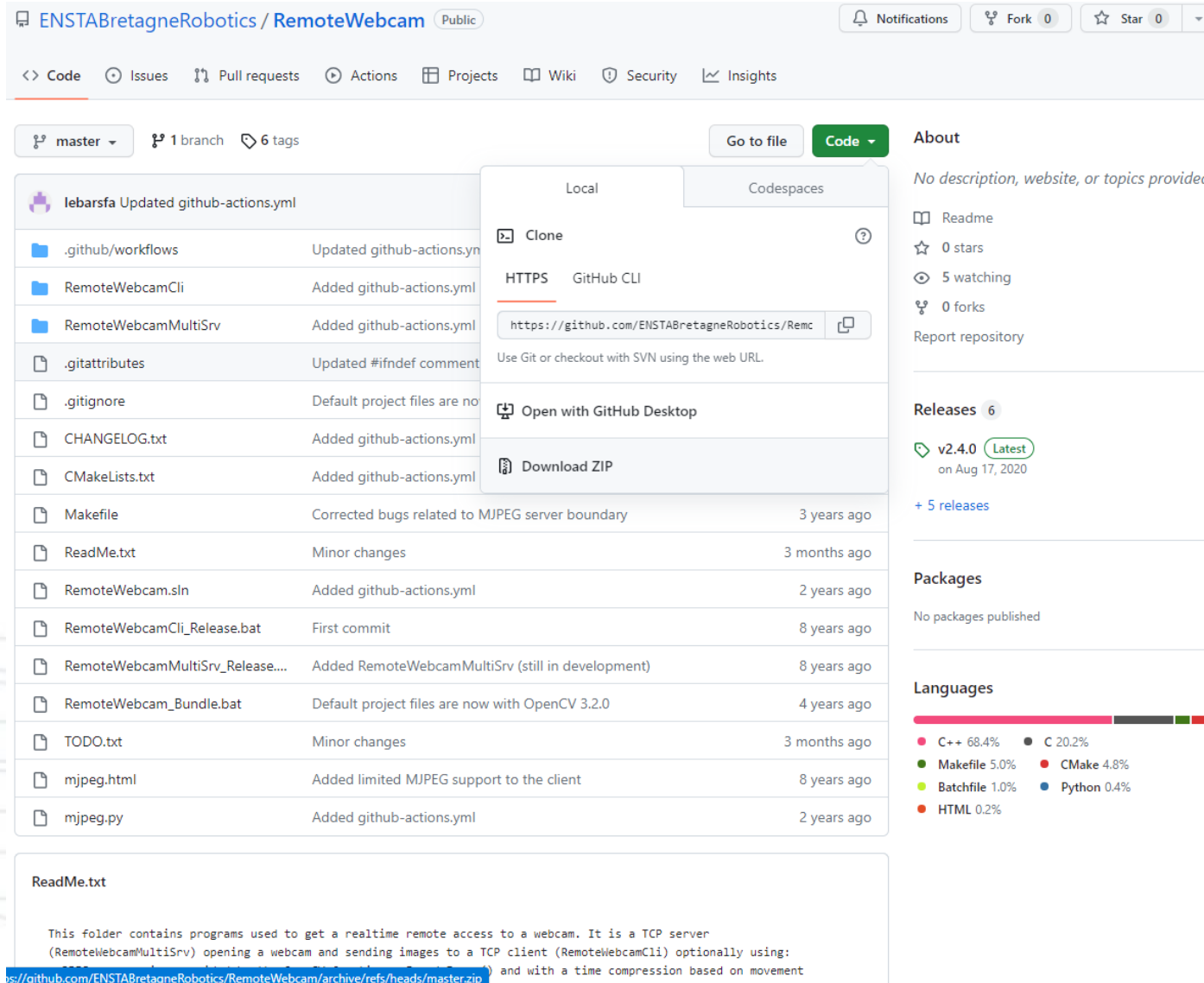
Lire le **ReadMe** éventuel qui apparait **en bas** de la page principale d'un projet
Voir s'il y a des **Releases** à droite, ils pourraient contenir une version précompilée du projet

Éventuellement utiliser **Code > Download ZIP** pour télécharger rapidement le code sans l'historique des versions

Penser à regarder les **branches** et **forks** des projets et l'onglet **Insights > Network** pour voir ce qui pourrait être corrigé/amélioré par rapport à la branche principale en cas de problème



Utilisation de code existant trouvé sur Internet



ENSTABretagneRobotics / RemoteWebcam Public

Notifications Fork 0 Star 0

<> Code Issues Pull requests Actions Projects Wiki Security Insights

master 1 branch 6 tags Go to file Code

lebarsfa Updated github-actions.yml

File/Folder	Description	Time
.github/workflows	Updated github-actions.yml	
RemoteWebcamCli	Added github-actions.yml	
RemoteWebcamMultiSrv	Added github-actions.yml	
.gitattributes	Updated #ifndef comment	
.gitignore	Default project files are no	
CHANGELOG.txt	Added github-actions.yml	
CMakeLists.txt	Added github-actions.yml	
Makefile	Corrected bugs related to MJPEG server boundary	3 years ago
ReadMe.txt	Minor changes	3 months ago
RemoteWebcam.sln	Added github-actions.yml	2 years ago
RemoteWebcamCli_Release.bat	First commit	8 years ago
RemoteWebcamMultiSrv_Release...	Added RemoteWebcamMultiSrv (still in development)	8 years ago
RemoteWebcam_Bundle.bat	Default project files are now with OpenCV 3.2.0	4 years ago
TODO.txt	Minor changes	3 months ago
mjpeg.html	Added limited MJPEG support to the client	8 years ago
mjpeg.py	Added github-actions.yml	2 years ago

Local Codespaces

Clone ?

HTTPS GitHub CLI

https://github.com/ENSTABretagneRobotics/Remc

Use Git or checkout with SVN using the web URL.

Open with GitHub Desktop

Download ZIP

About

No description, website, or topics provided.

Readme

0 stars

5 watching

0 forks

Report repository

Releases 6

v2.4.0 Latest on Aug 17, 2020

+ 5 releases

Packages

No packages published

Languages

- C++ 68.4%
- C 20.2%
- Makefile 5.0%
- CMake 4.8%
- Batchfile 1.0%
- Python 0.4%
- HTML 0.2%

ReadMe.txt

This folder contains programs used to get a realtime remote access to a webcam. It is a TCP server (RemoteWebcamMultiSrv) opening a webcam and sending images to a TCP client (RemoteWebcamCli) optionally using: <https://github.com/ENSTABretagneRobotics/RemoteWebcam/archive/refs/heads/master.zip> and with a time compression based on movement

■ Questions à se poser

- Le code trouvé est pour quel **OS** (e.g. Windows, Ubuntu, macOS, Red Hat...), **compilateur/IDE** (GCC, Visual Studio...), **machine** (x86/i386/i586/i686, x64/amd64/x86_64, armhf, arm64...)?
- Y-a-t-il des **dépendances** à installer avant?
 - => Chercher s'il y a un fichier **ReadMe.txt**/ReadMe.md/README ou **Install.txt**/INSTALL, lire le site web...
- Présence d'un fichier **CMakeLists.txt** => CMake (utiliser cmake-gui permet parfois de voir facilement les options activables éventuelles)
- Présence de fichiers **configure** et **Makefile** => **./configure** ; **make** ; **make install** (ou parfois pas de configure, besoin aussi de **sudo** parfois pour **make install**...)
- Présence de **.sln**, **.vcproj/.vcxproj**, version de Visual Studio, fichiers projet d'autre IDE (e.g. **.pro** pour **Qt Creator**)...



Utilisation de bibliothèques de fonctions externes

Utilisation de bibliothèques de fonctions externes

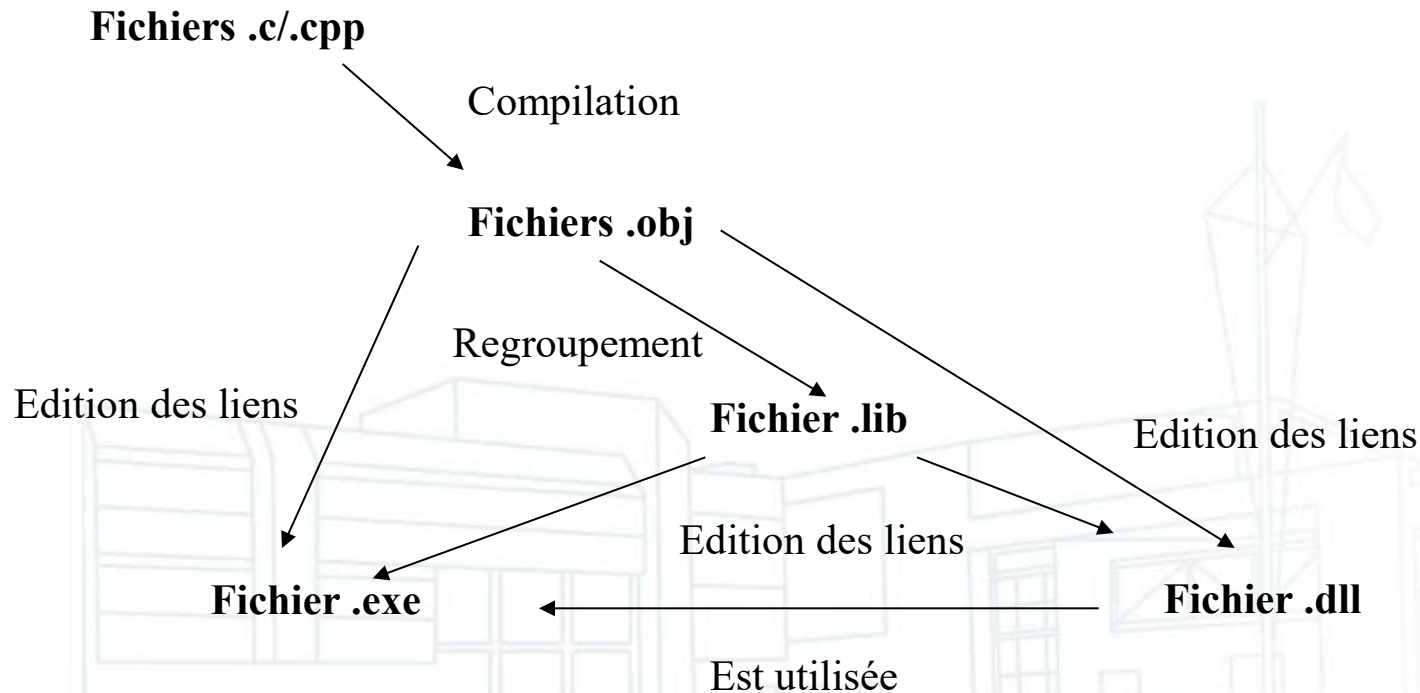
■ Vocabulaire

- On appelle souvent **API** (Application Programming Interface) l'ensemble des fonctions, classes, constantes utilisables d'une bibliothèque. En C/C++, elles sont en général déclarées dans des fichiers **.h**



Utilisation de bibliothèques de fonctions externes

- Utilisation de bibliothèques de fonctions externes



Utilisation de bibliothèques de fonctions externes

- Utilisation de bibliothèques de fonctions externes

- Cas où on possède des fichiers **.h** et **.c/.cpp**

Exemple : on a **Main.cpp** qui doit appeler des fonctions déclarées dans **Lib.h** et définies dans **Lib.cpp**

On met **#include « Lib.h »** dans **Main.cpp**

On copie **Lib.h** et **Lib.cpp** dans le dossier de **Main.cpp**

On compile et lie **Lib.cpp** et **Main.cpp** en les ajoutant au projet



Utilisation de bibliothèques de fonctions externes

- Utilisation de bibliothèques de fonctions externes
 - Cas où on possède seulement des fichiers **.h** (toutes les fonctions sont a priori définies **inline**)

Exemple : on a **Main.cpp** qui doit appeler des fonctions définies dans **Lib.h**

On met **#include « Lib.h »** dans **Main.cpp**

On ajoute le dossier de **Lib.h** dans les chemins de recherche de fichiers **.h** du projet

On compile **Main.cpp** en l'ajoutant au projet



Utilisation de bibliothèques de fonctions externes

- Utilisation de bibliothèques de fonctions externes

- Cas où on possède des fichiers **.h**, **.lib** et **.dll**

Exemple : on a **Main.cpp** qui doit appeler des fonctions déclarées dans **Lib.h** et définies dans **Lib.lib** et **Lib.dll**

On met **#include « Lib.h »** dans **Main.cpp**

On ajoute le dossier de **Lib.h** dans les chemins de recherche de fichiers **.h** du projet

On ajoute le dossier de **Lib.lib** dans les chemins de recherche de fichiers **.lib** du projet (sinon il faudra mettre le chemin complet dans l'étape suivante)

On compile **Main.cpp** et lie avec **Lib.lib** en les ajoutant au projet

On ajoute le dossier de **Lib.dll** à la variable d'environnement **PATH** du système (**LD_LIBRARY_PATH** pour les **.so** sous Linux)

Utilisation de bibliothèques de fonctions externes

- Utilisation de bibliothèques de fonctions externes
 - Spécifique **Visual Studio** (similaire aussi pour **MinGW**) : cas où on possède des fichiers **.h** et **.dll**, sans **.lib**

Exemple : on a **Main.cpp** qui doit appeler des fonctions déclarées dans **Lib.h** et définies dans **Lib.dll**

Il faut générer **Lib.lib** à partir de **Lib.dll** :

_ Run a **Visual Studio Developer Command Prompt** and run the following command in the folder where **Lib.dll** is :

dumpbin.exe /EXPORTS Lib.dll>Lib.exports

_ Paste the names of the needed functions (remove the line and hex numbers using **ALT** key and mouse selection in Visual Studio) from **Lib.exports** into a new **Lib.def** file. Add a line with the word **EXPORTS** at the top of this file.

_ Run the following command :

lib /def:Lib.def /out:Lib.lib
to generate **Lib.lib**

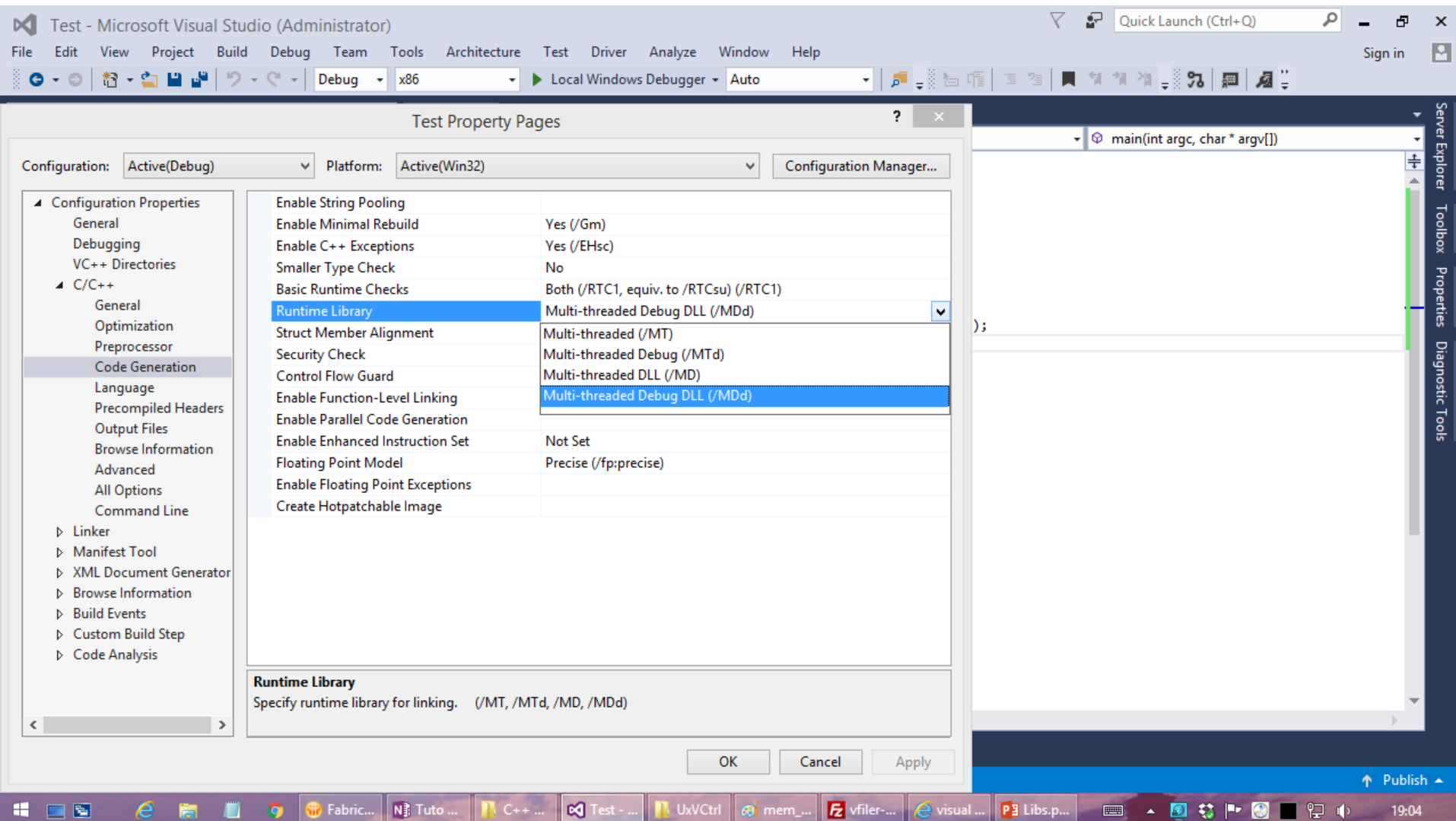
- Sous Linux le **.a** n'est pas forcément nécessaire si on a un **.so** (dans ce cas le **.so** a le double rôle du **.lib** et **.dll** de Windows)

Utilisation de bibliothèques de fonctions externes

■ Problèmes courants :

- Bibliothèque compilée avec une version différente du compilateur
- Pour Visual Studio, vérifier la cohérence des options de **Configuration (Debug, Release)** et **Platform (x86, x64)**, ainsi que **Properties > C/C++ > Code Generation > Runtime Library** utilisées par la bibliothèque par rapport à celles utilisées par l'application voulant utiliser la bibliothèque, si c'est incohérent il faut **recompiler la bibliothèque** avec les bonnes options. Dans l'idéal il faudrait 8 versions : Debug /MDd x86, Release /MD x86, Debug /MTd x86, Release /MT x86, Debug /MDd x64, Release /MD x64, Debug /MTd x64, Release /MT x64. Il est aussi parfois nécessaire de désactiver les options /GL et /LCTG lors de la compilation de .lib pour améliorer leur compatibilité entre différentes révisions mineures des compilateurs...
- Sous Linux, l'équivalent correspond à la version de **libc**, le fait qu'elle soit compilée en static ou dynamic, etc. De plus, il y a aussi la variable **LD_LIBRARY_PATH** qui doit parfois être réglée

Utilisation de bibliothèques de fonctions externes



The screenshot shows the Visual Studio interface with the 'Test Property Pages' dialog open. The dialog is configured for 'Active(Debug)' and 'Active(Win32)'. The 'Runtime Library' property is highlighted, showing the value 'Multi-threaded Debug DLL (/MDd)'. The 'Linker' and 'Manifest Tool' sections are also visible in the left sidebar.

Configuration: Active(Debug) Platform: Active(Win32) Configuration Manager...

Property	Value
Enable String Pooling	
Enable Minimal Rebuild	Yes (/Gm)
Enable C++ Exceptions	Yes (/EHsc)
Smaller Type Check	No
Basic Runtime Checks	Both (/RTC1, equiv. to /RTCsu) (/RTC1)
Runtime Library	Multi-threaded Debug DLL (/MDd)
Struct Member Alignment	Multi-threaded (/MT)
Security Check	Multi-threaded Debug (/MTd)
Control Flow Guard	Multi-threaded DLL (/MD)
Enable Function-Level Linking	Multi-threaded Debug DLL (/MDd)
Enable Parallel Code Generation	
Enable Enhanced Instruction Set	Not Set
Floating Point Model	Precise (/fp:precise)
Enable Floating Point Exceptions	
Create Hotpatchable Image	

Runtime Library
Specify runtime library for linking. (/MT, /MTd, /MD, /MDd)

OK Cancel Apply

Utilisation de bibliothèques de fonctions externes

■ Problèmes courants :

- Dans la même idée que les points précédents, il est possible que tous les fichiers binaires fournis (.dll, .exe, .so) n'aient pas été compilés avec le même compilateur (e.g. erreurs mentionnant **MSVCR100.dll**, **MSVCP110.dll**, etc. sous Windows, **libstdc++.so**, **libgcc_s.so**, etc. sous Linux), il faut donc parfois installer plusieurs versions de **Visual Studio Redistributable** pour Windows (voir e.g. <https://docs.microsoft.com/en-us/cpp/windows/latest-supported-vc-redist>) et plusieurs versions de **libc** sous Linux (utiliser la variable **LD_PRELOAD** peut parfois aider à sélectionner une version particulière dans certains cas)



Utilisation d'OpenCV 1.X avec Visual C++ 6

- Les chemins suivants sont des chemins absolus considérant l'installation par défaut dans **C:\Program Files\OpenCV**
- Créer un projet **Win32ConsoleApplication**
- Dans le menu "**Project**", "**Settings**", "**C/C++**", catégorie "**Preprocessor**".
Ajouter les chemins suivants dans "**Additional include directories**":
 - **C:\Program Files\OpenCV\cv\include,**
 - **C:\Program Files\OpenCV\cvaux\include,**
 - **C:\Program Files\OpenCV\cxcore\include,**
 - **C:\Program Files\OpenCV\otherlibs\highgui**
- Dans le menu "**Project**", "**Settings**", "**Link**", catégorie "**Input**".
Ajouter les bibliothèques suivantes dans "**Object/library modules**":
 - **cv.lib cvaux.lib cxcore.lib highgui.lib**
- Ajouter le chemin suivant dans "**Additional library path**":
 - **C:\Program Files\OpenCV\lib**
- Modifier la variable d'environnement "**PATH**" de Windows en ajoutant:
 - **C:\Program Files\OpenCV\bin**
- Dans le code, ajouter
 - **#include "cvaux.h"**
 - **#include "highgui.h"**

Utilisation d'OpenCV 1.X avec Visual Studio 2008

- Voir <http://www.ensta-bretagne.fr/lebars/tutorials/Config%20Visual%20Studio%20OpenCV.pdf>



Utilisation de différentes versions d'OpenCV avec Visual Studio, Qt Creator, etc.

- Voir <http://www.ensta-bretagne.fr/lebars/tutorials>



