ENSTA Bretagne

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 chaine 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)…

C

Hello World!

```c
#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!

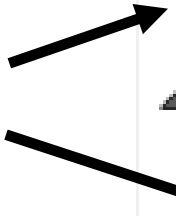
Notion de namespace

```
#include <iostream>

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

    return 0;
}
```

Opérateur

Namespace

Plus besoin d'utiliser std::

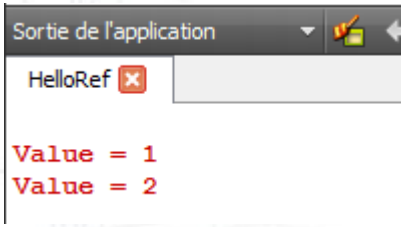


```
#include <iostream>

using namespace std;

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

    return 0;
}
```

Références

```cpp
#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

```h
#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

```cpp
#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

```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
```

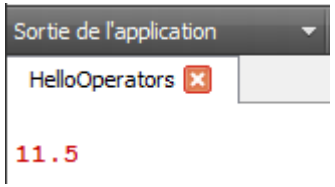
TestClass.h

```
#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.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

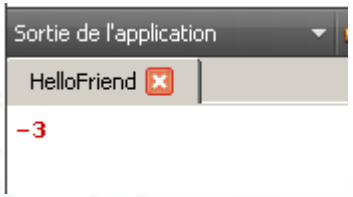
Fonctions amies

```cpp
#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

Classe de base

Classe dérivées

```
// 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;
}
```

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

mother: int parameter
son: int parameter
```

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

Classes abstraites

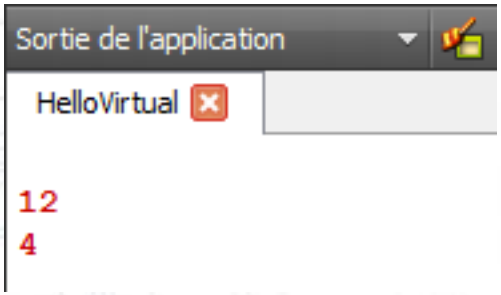
Classe de base abstraite

Classes dérivées

```
#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;
    }
};
```

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;
}
```



C++

- 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

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

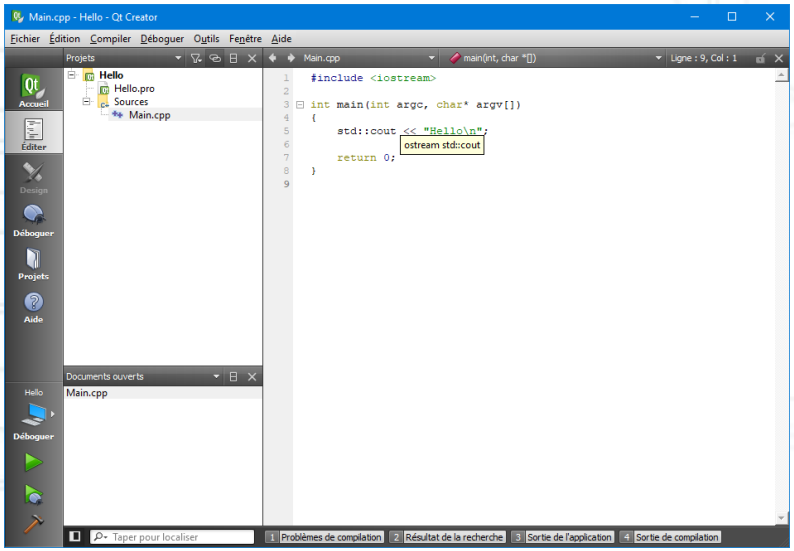
- CMake
  - En cas de problème, supprimer les fichiers générés par CMake, notamment **CMakeCache.txt**
  - Problèmes courants : CMake peut ne pas trouver le compilateur qu'on souhaite utiliser, utiliser l'option **-G** pour spécifier explicitement le « Generator » souhaité, e.g. **cmake -G "MinGW Makefiles" .** 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

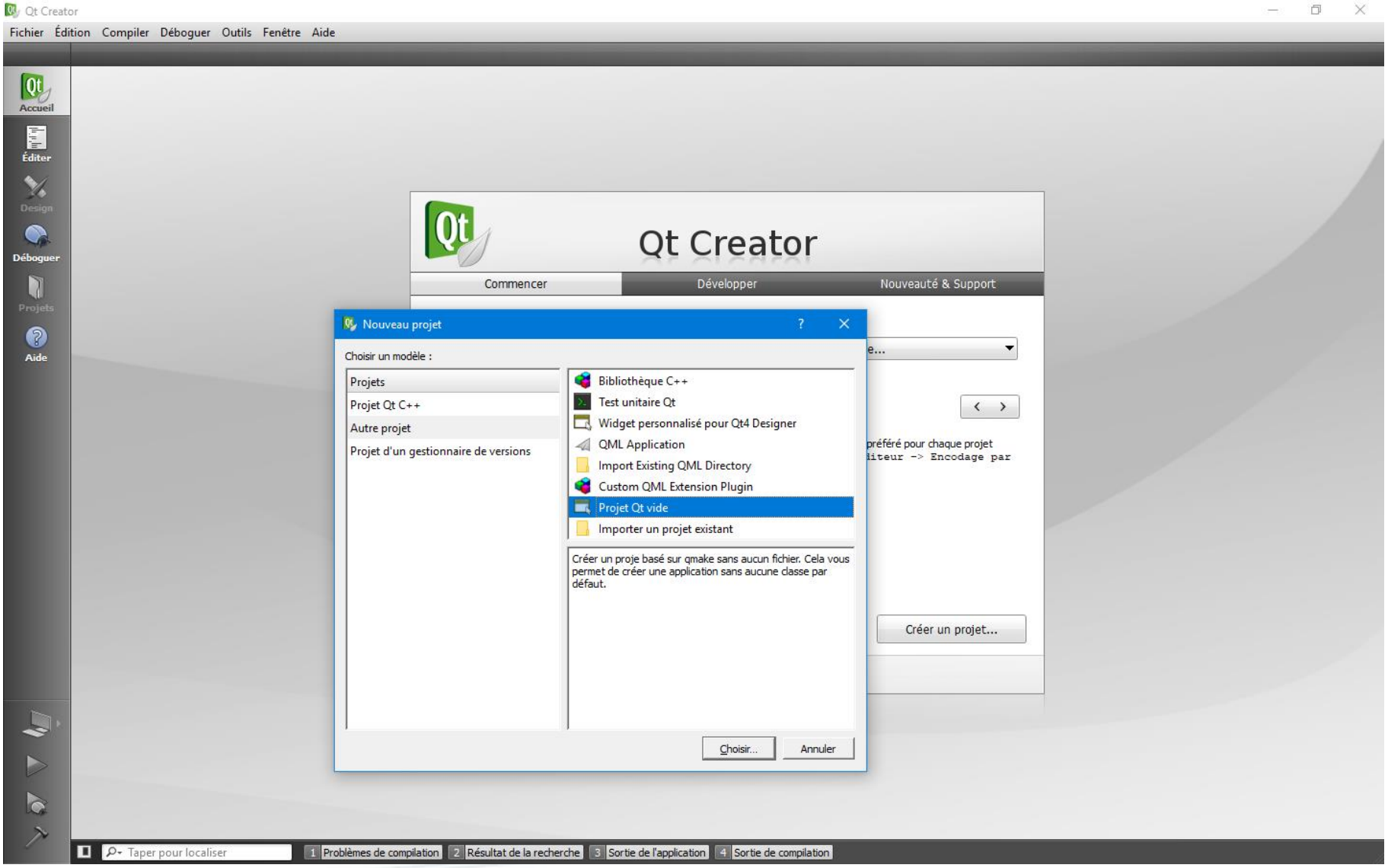
Qt Creator

- 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**

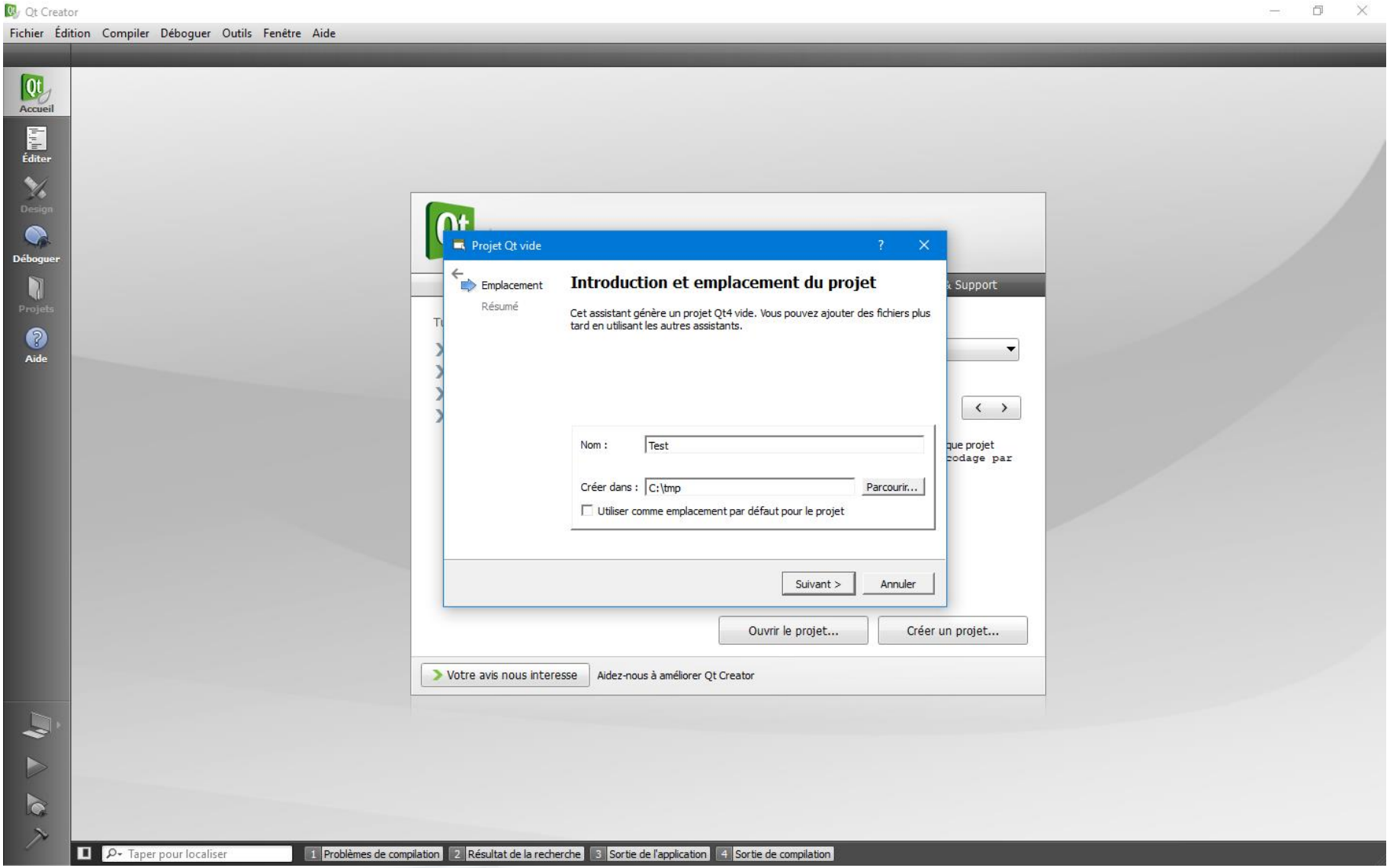
    Fichier texte **éditable** manuellement (e.g. pour pouvoir rajouter bibliothèques et options spécifiques)
  - Un fichier **CMakeLists.txt** peut aussi être utilisé en tant que projet **CMake**
  - Installation : voir e.g. https://www.ensta-bretagne.fr/lebars/Share/setup_qt_opencv.pdf



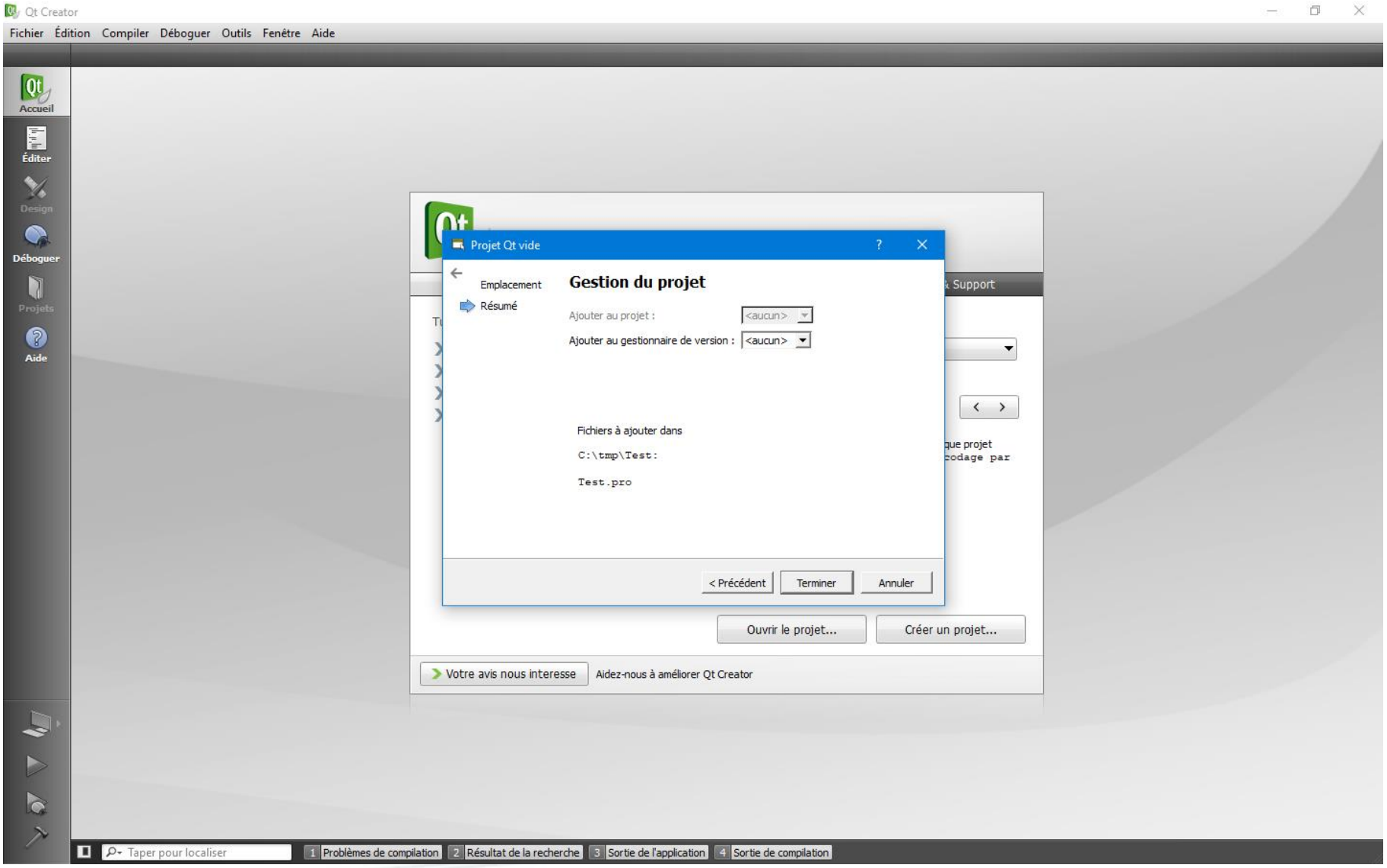
Créer un projet (non CMake) C simple avec Qt Creator



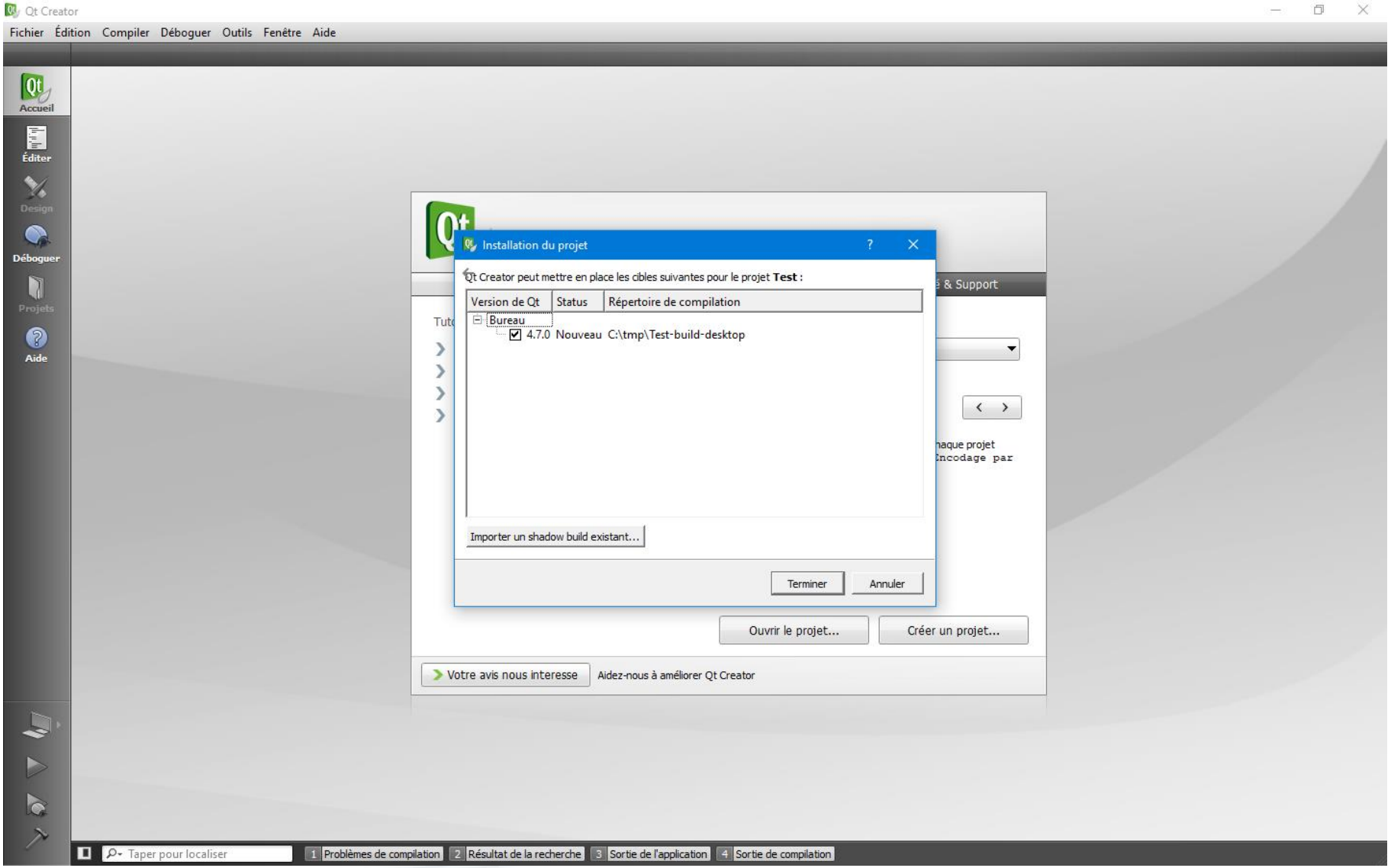
Créer un projet (non CMake) C simple avec Qt Creator



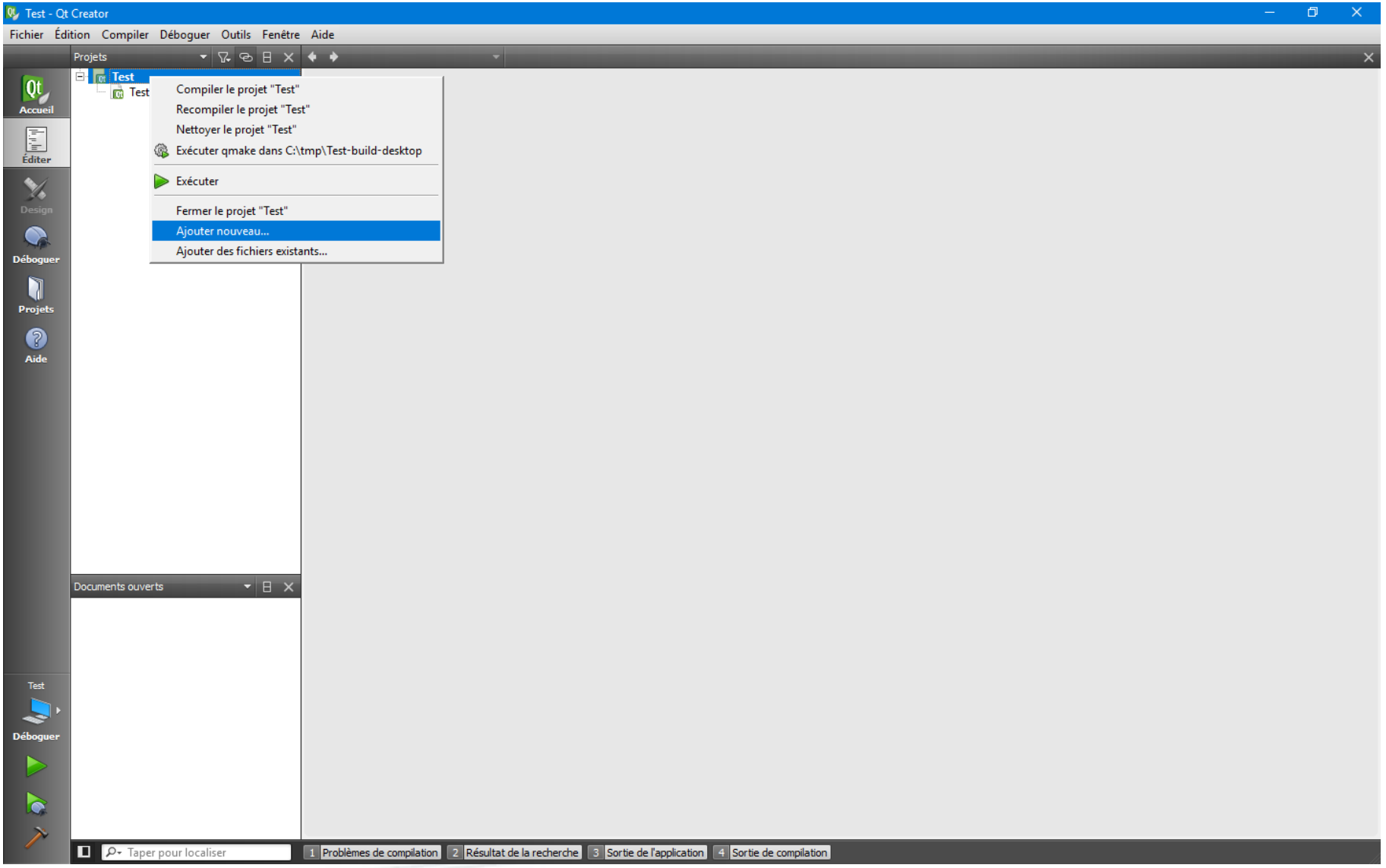
Créer un projet (non CMake) C simple avec Qt Creator



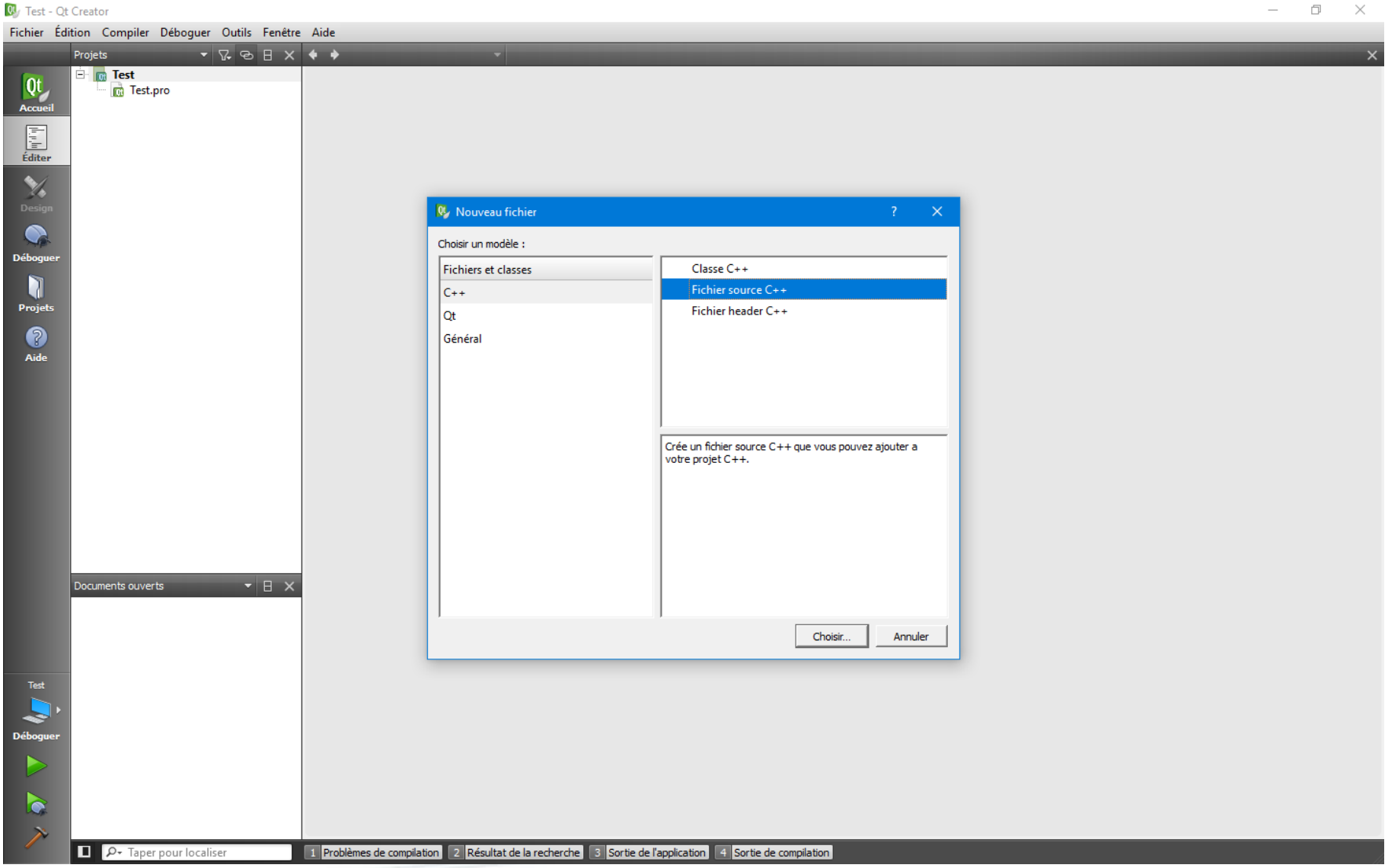
Créer un projet (non CMake) C simple avec Qt Creator



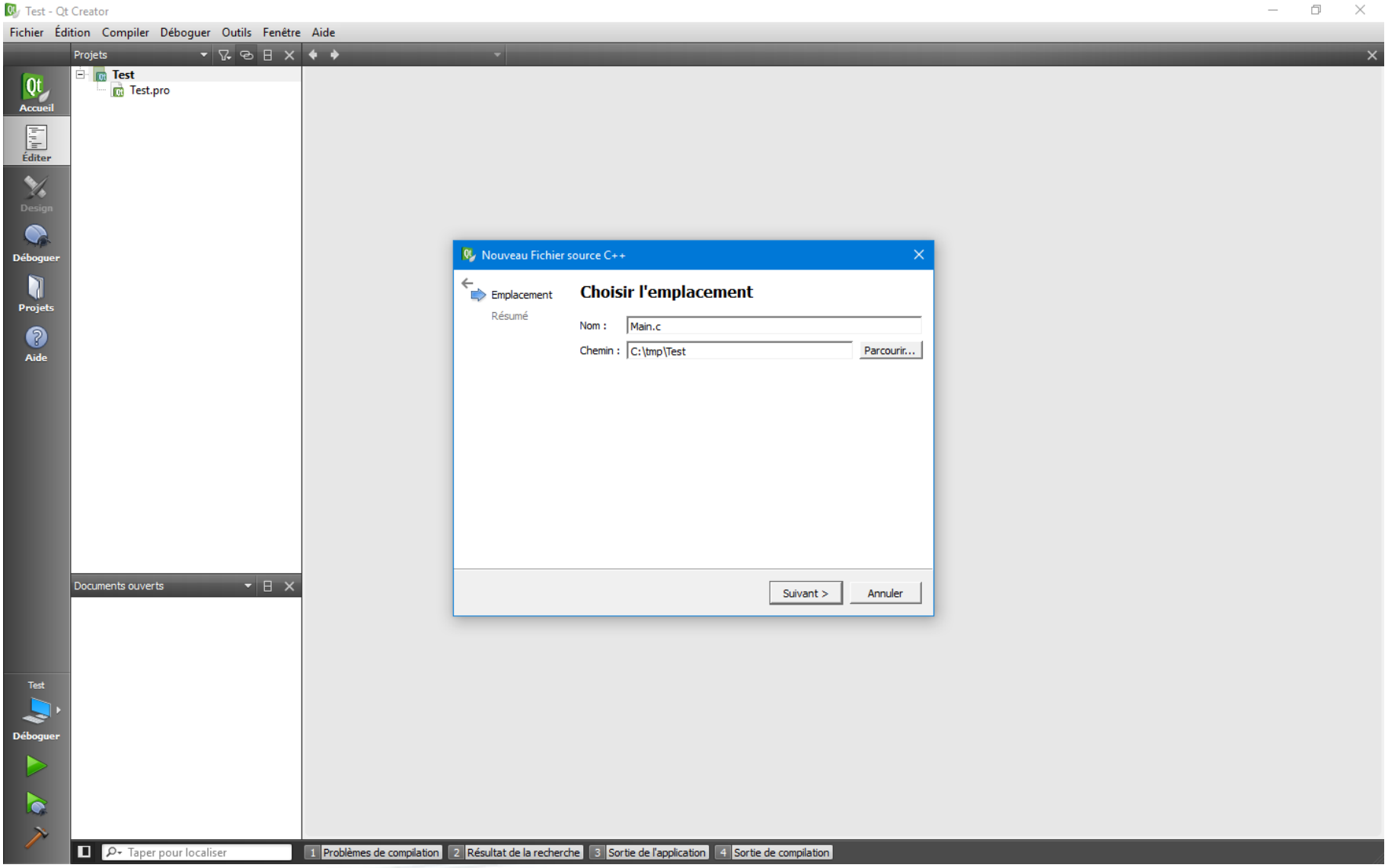
Créer un projet (non CMake) C simple avec Qt Creator



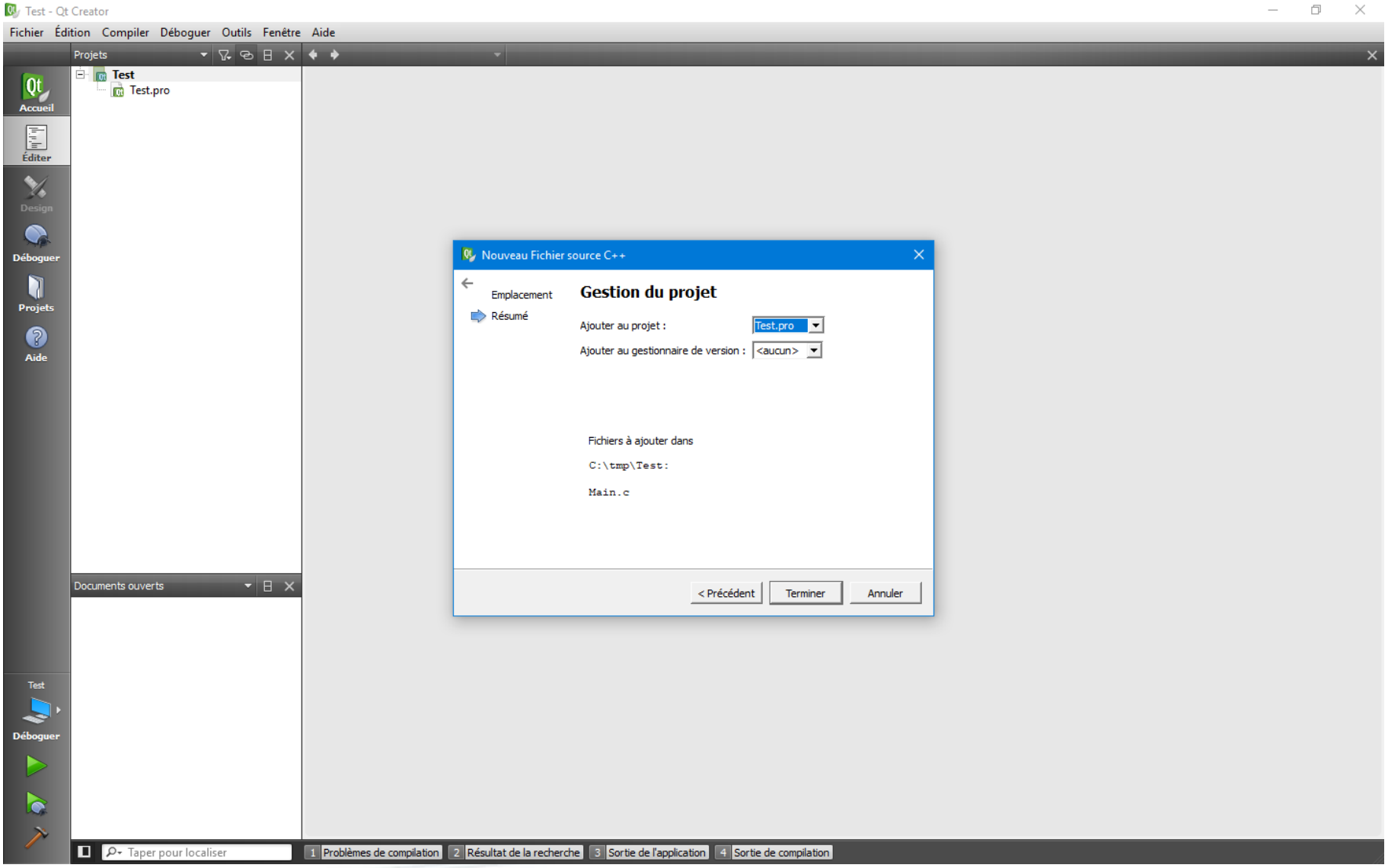
Créer un projet (non CMake) C simple avec Qt Creator



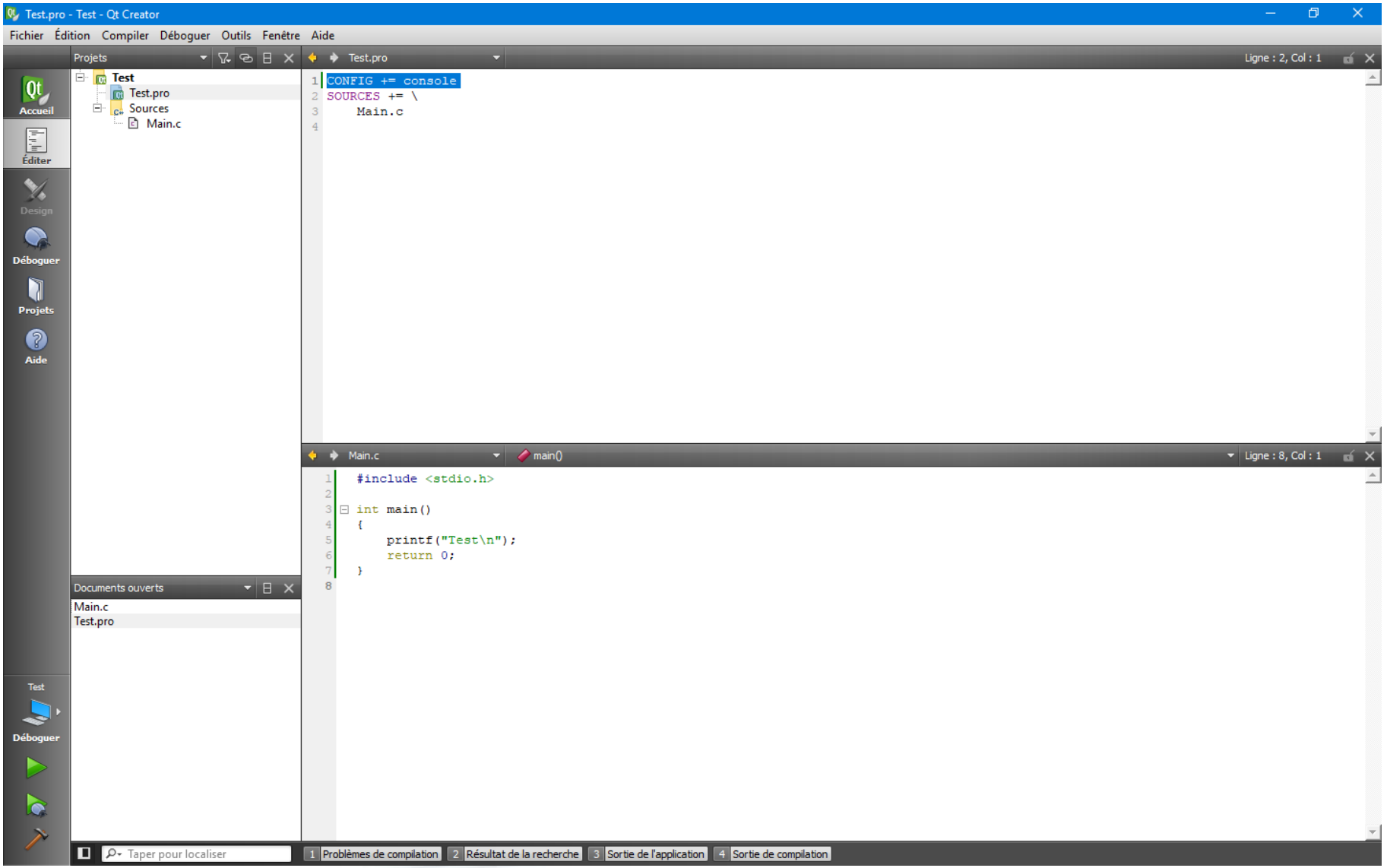
Créer un projet (non CMake) C simple avec Qt Creator



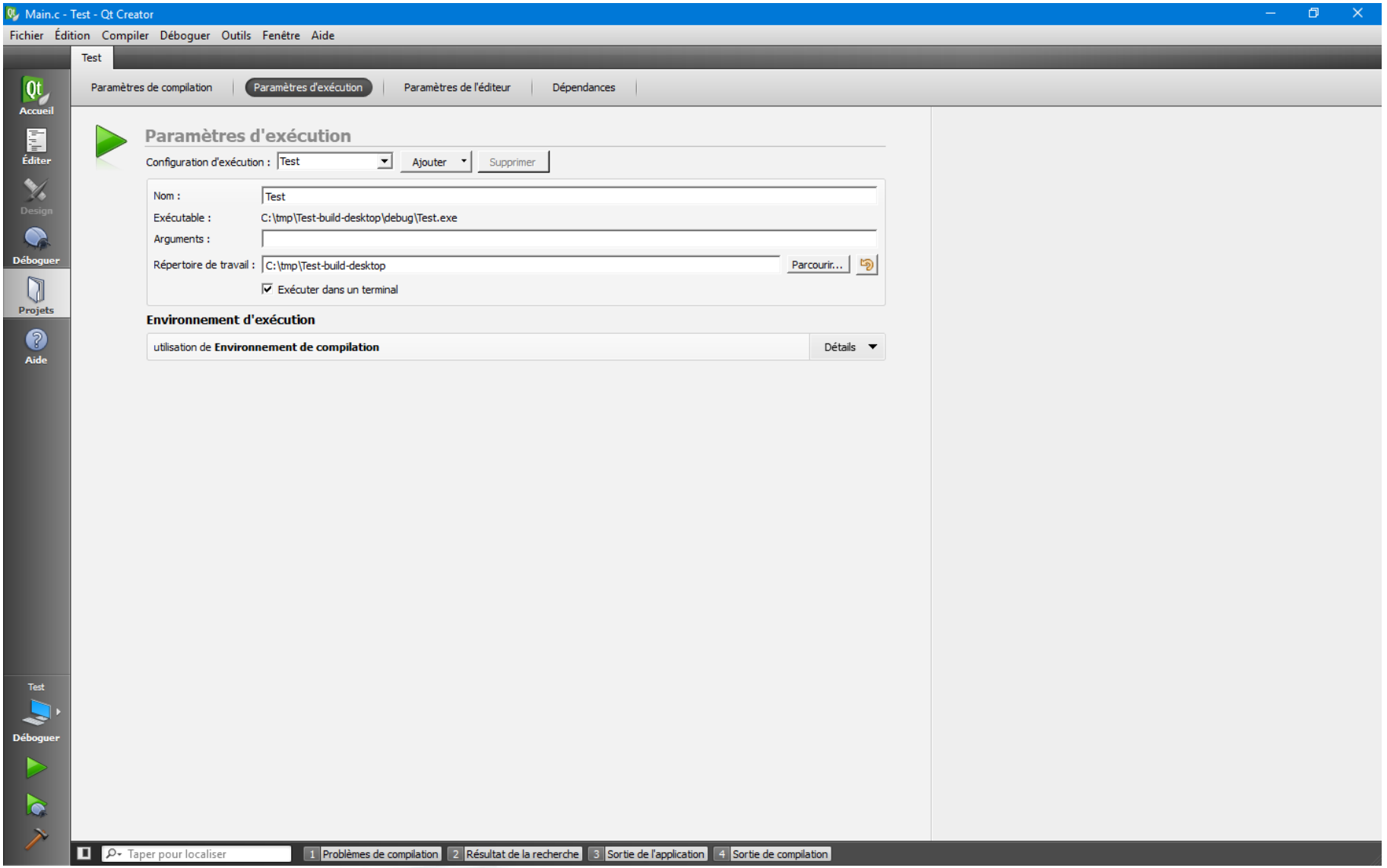
Créer un projet (non CMake) C simple avec Qt Creator



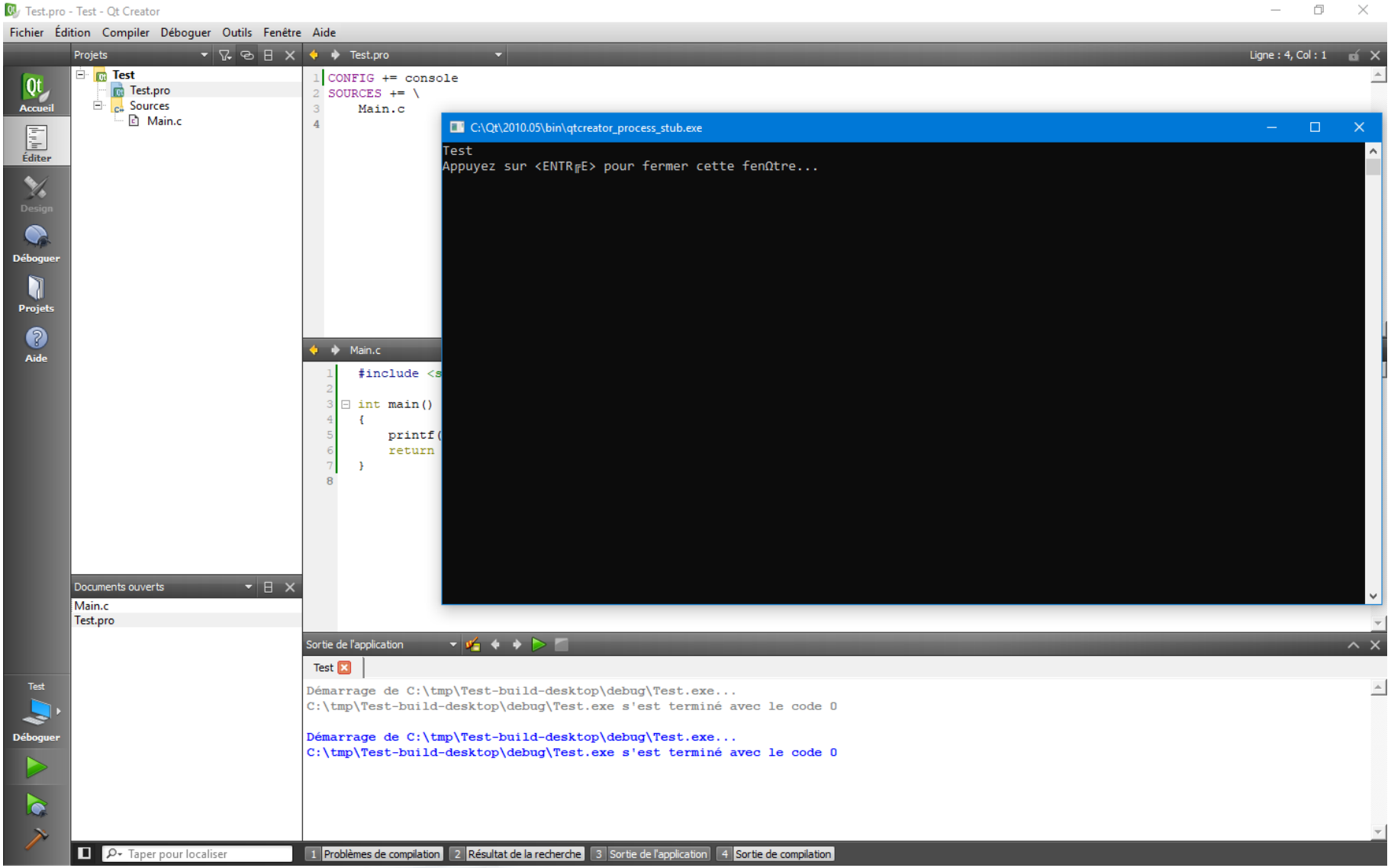
Créer un projet (non CMake) C simple avec Qt Creator



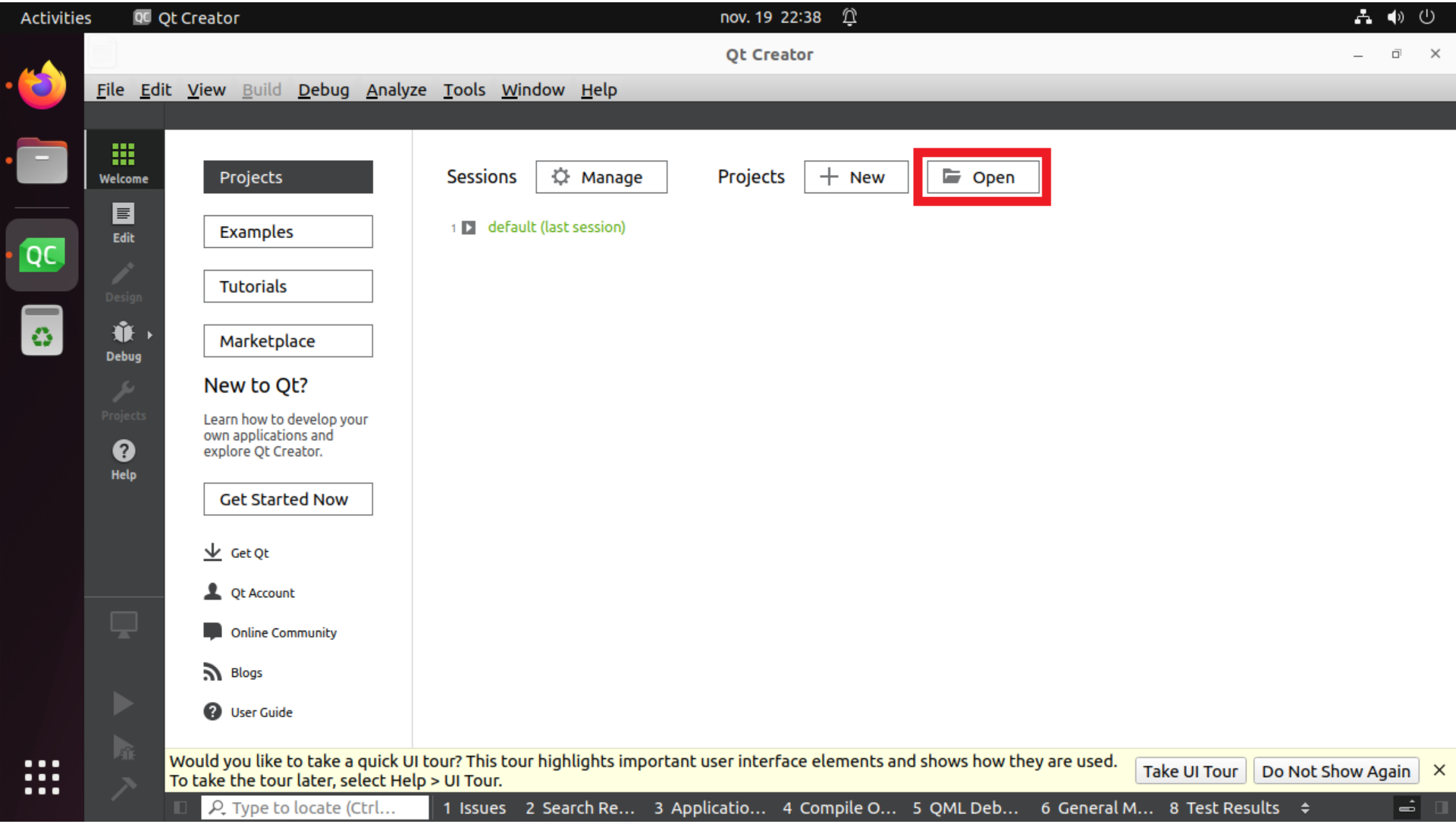
Créer un projet (non CMake) C simple avec Qt Creator



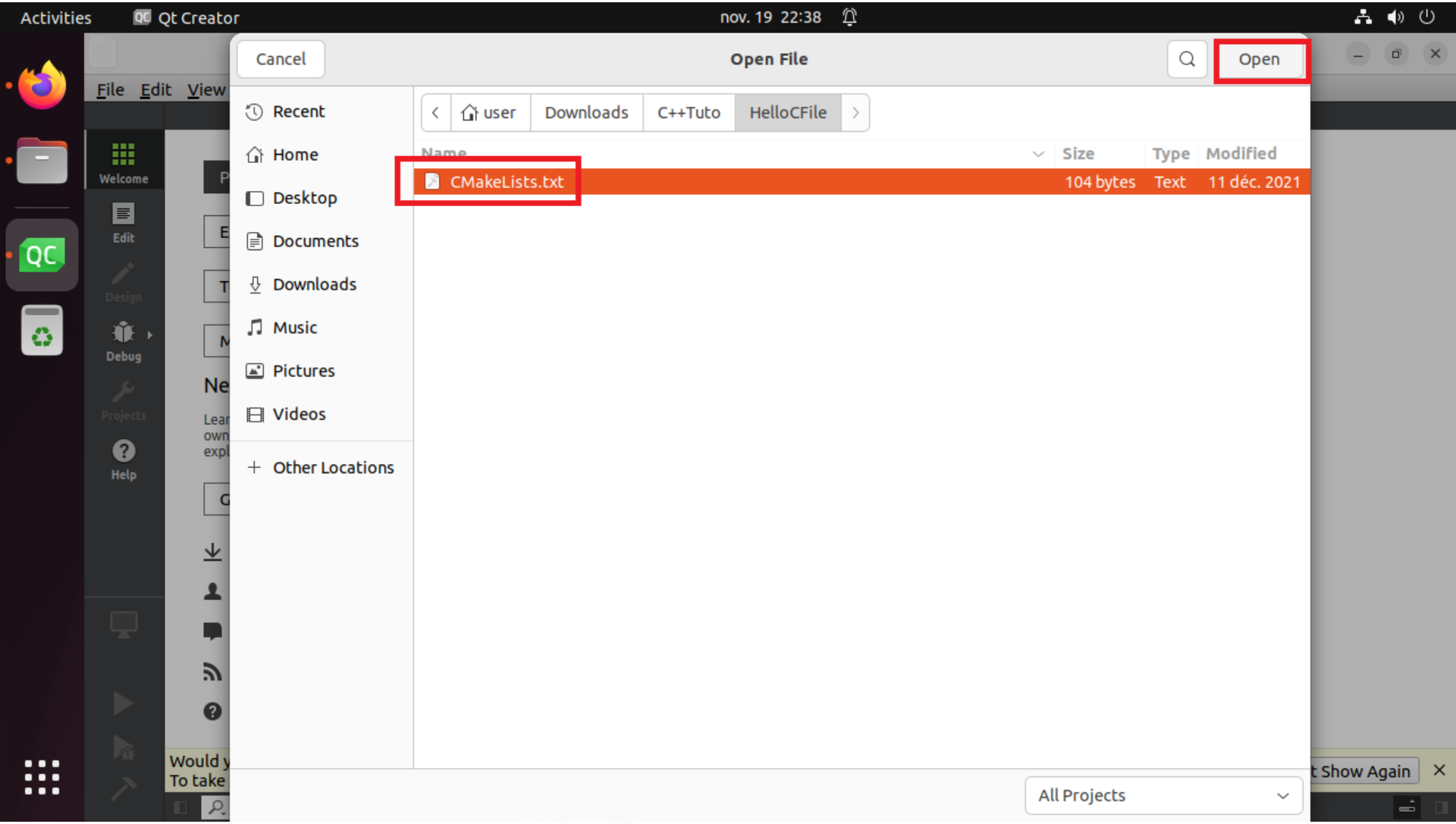
Créer un projet (non CMake) C simple avec Qt Creator



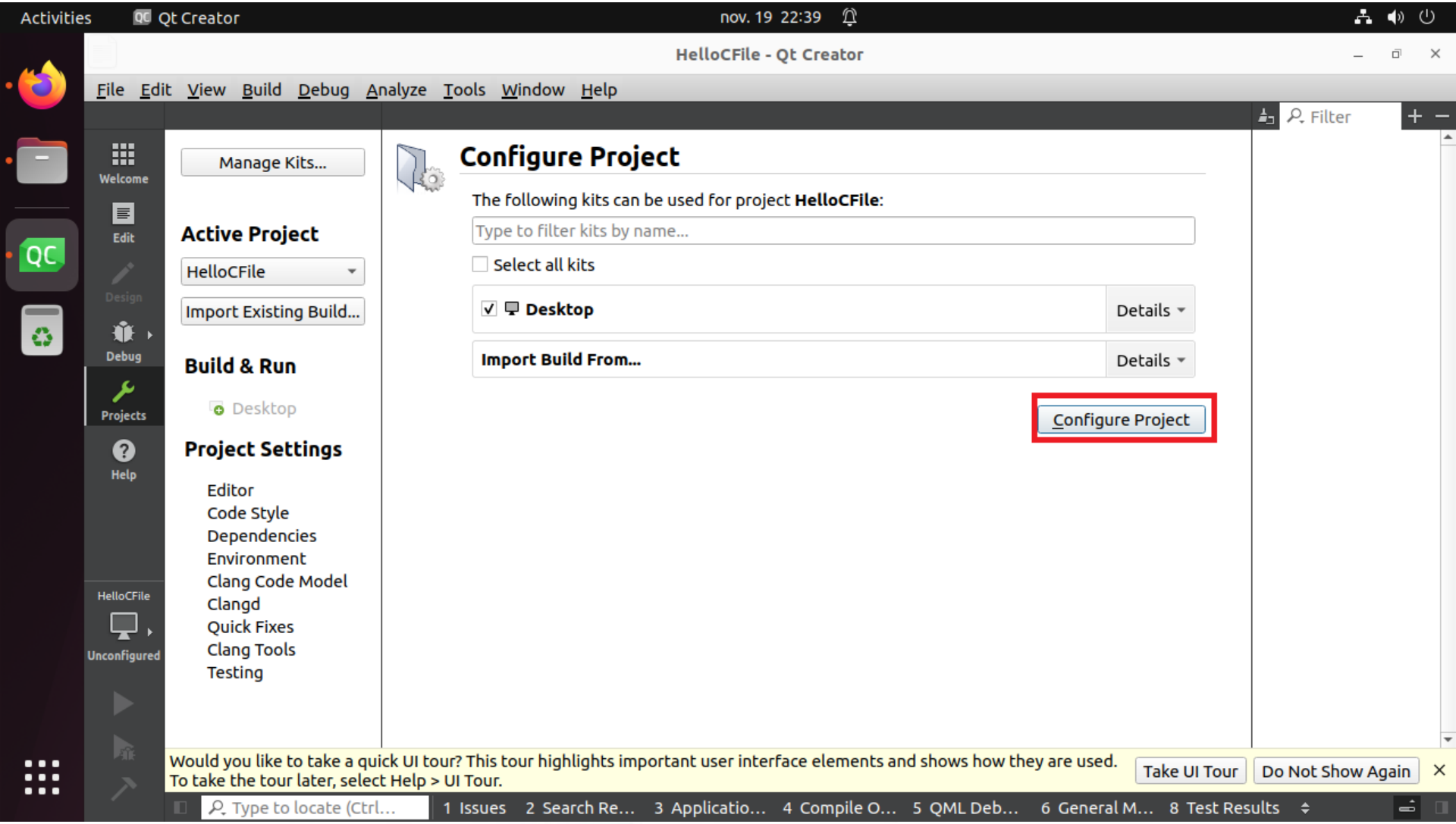
Opening and debugging a CMake project using Qt Creator



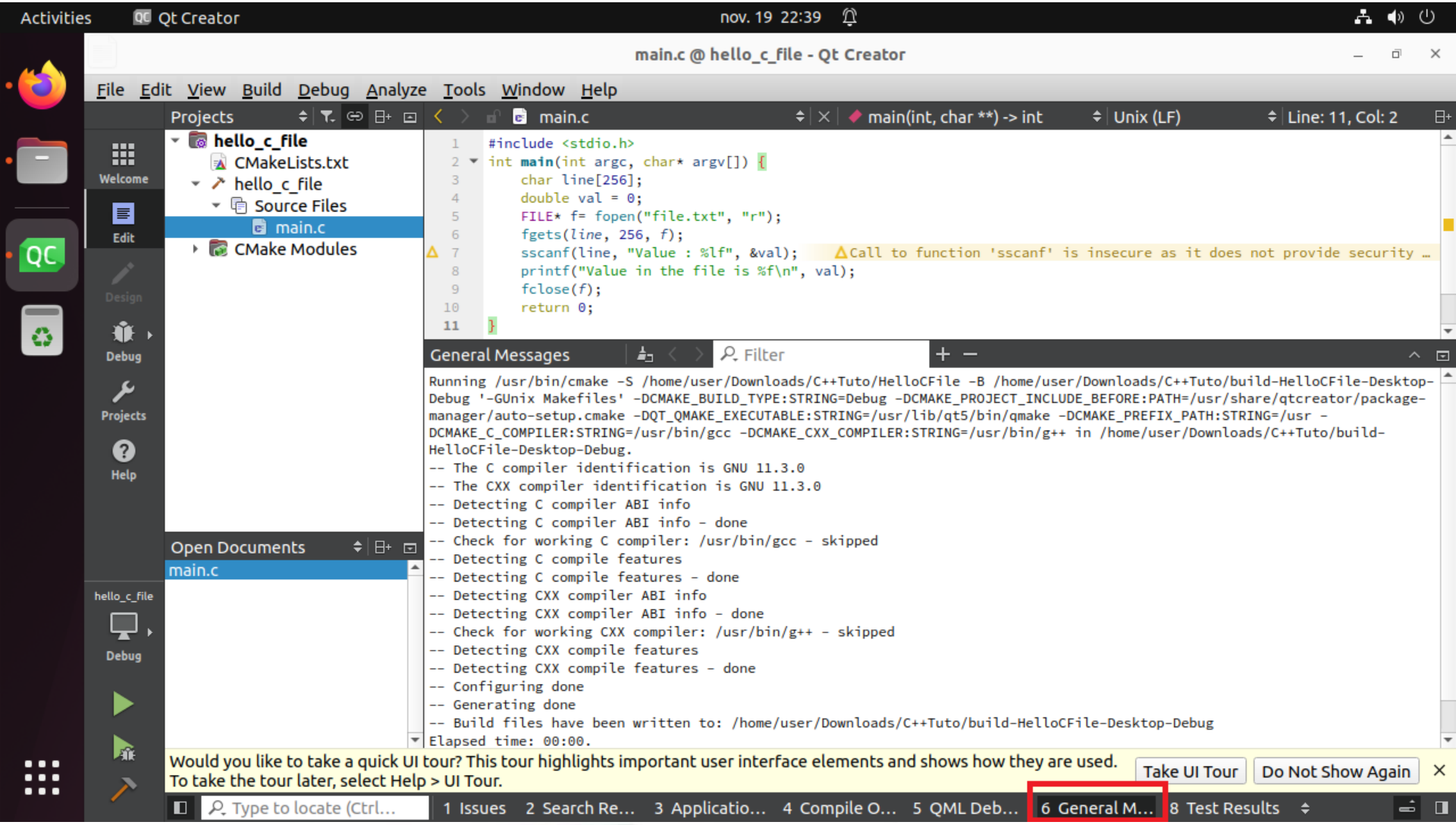
Opening and debugging a CMake project using Qt Creator



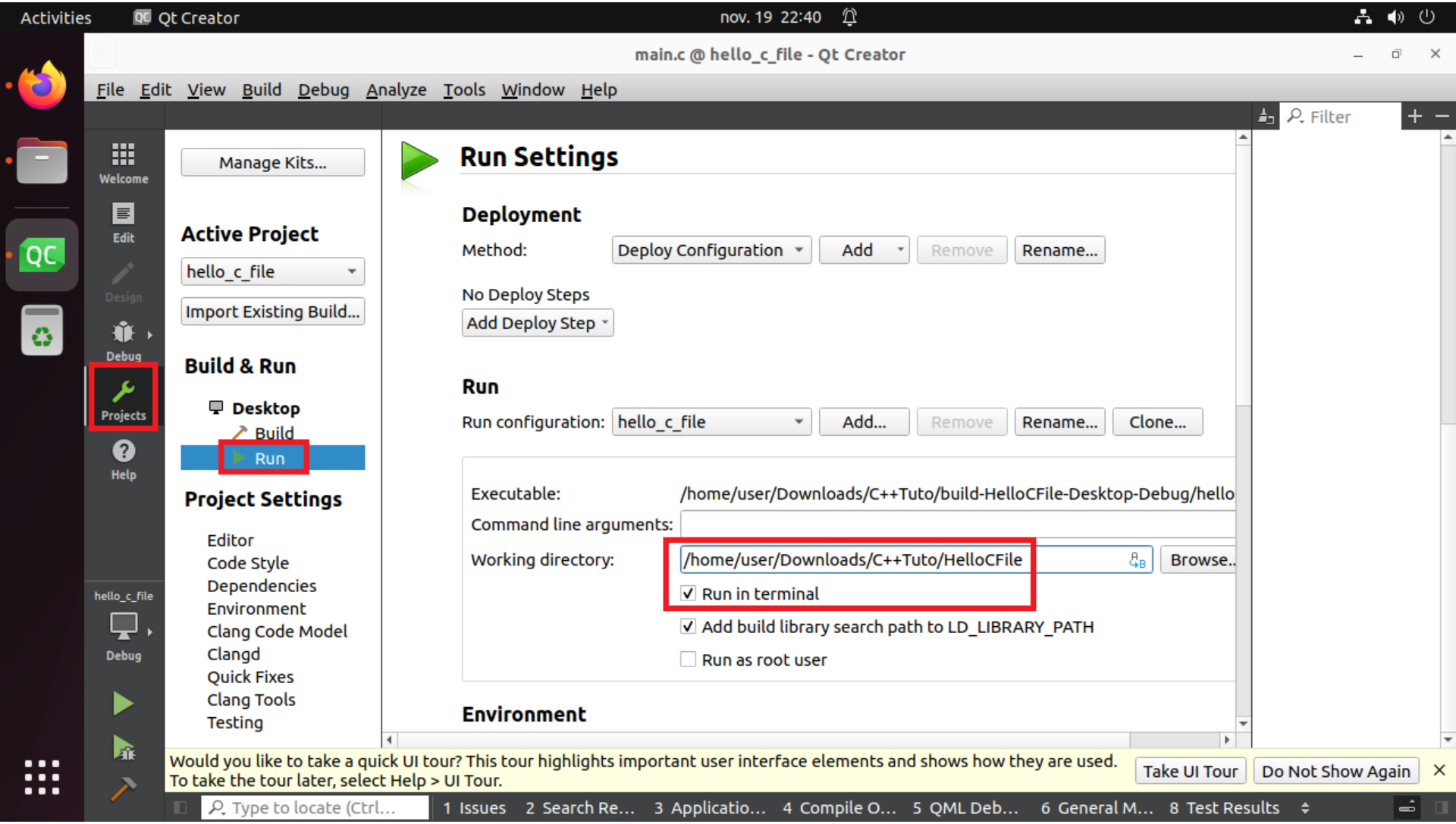
Opening and debugging a CMake project using Qt Creator



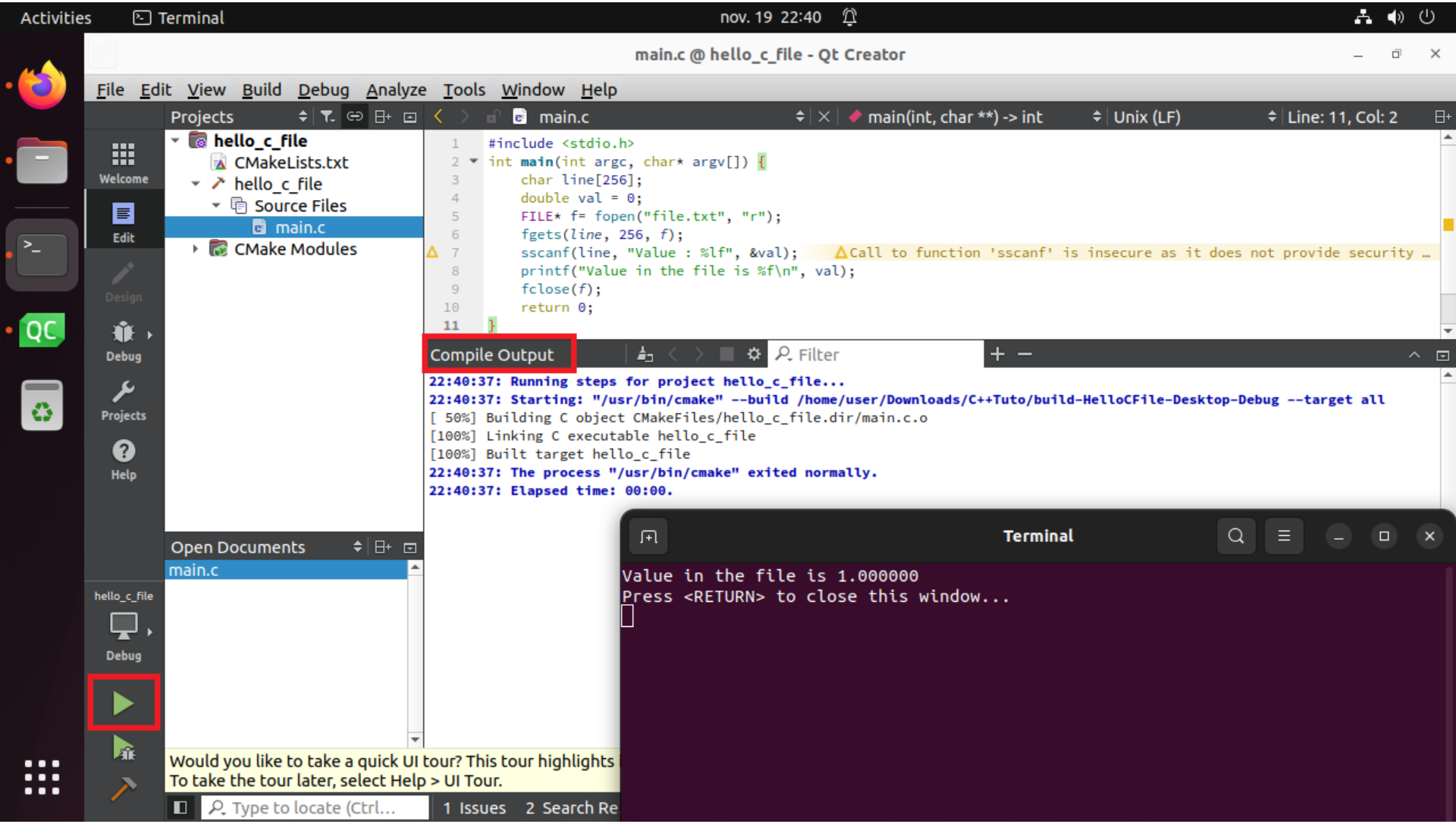
Opening and debugging a CMake project using Qt Creator



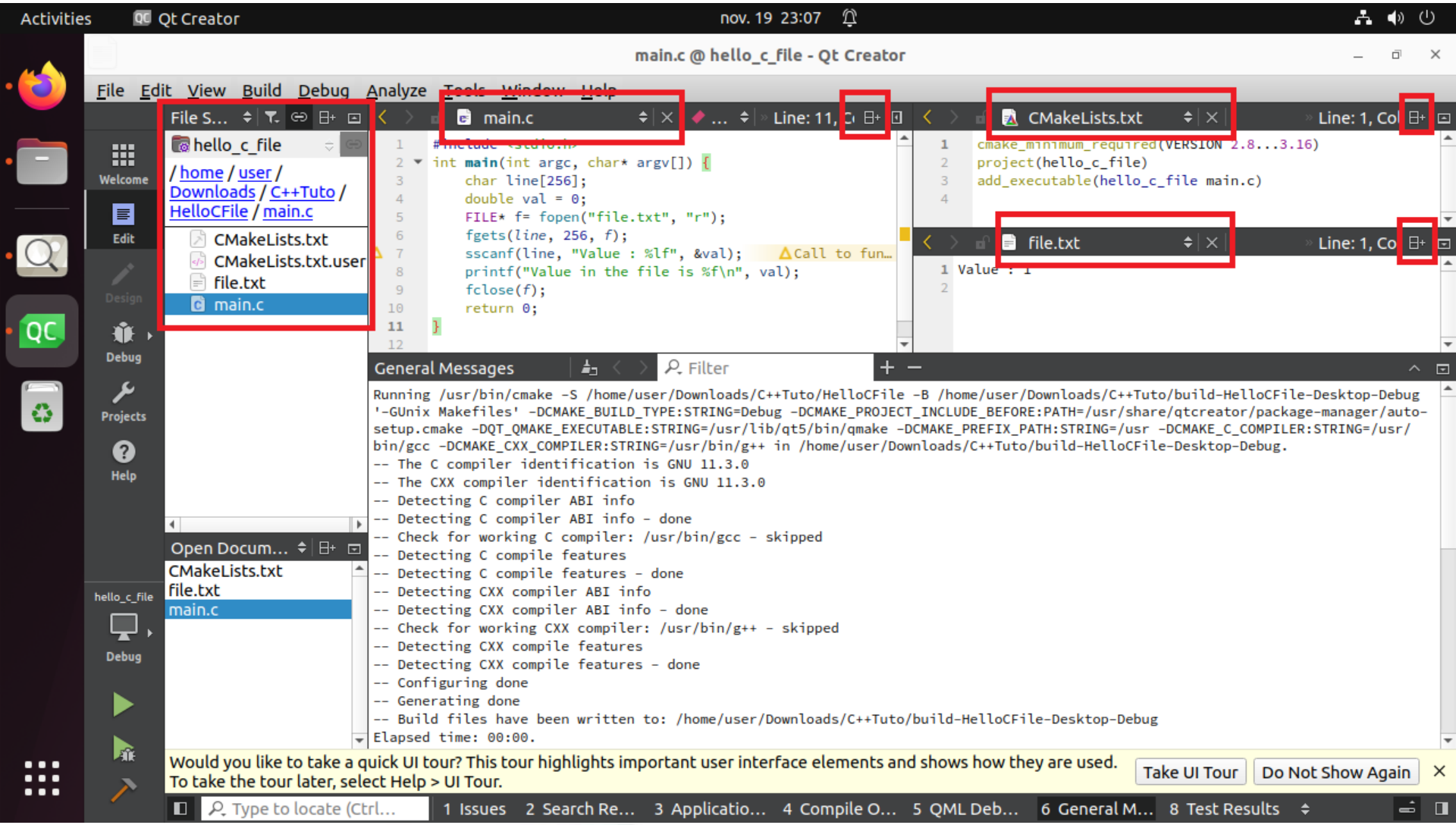
Opening and debugging a CMake project using Qt Creator



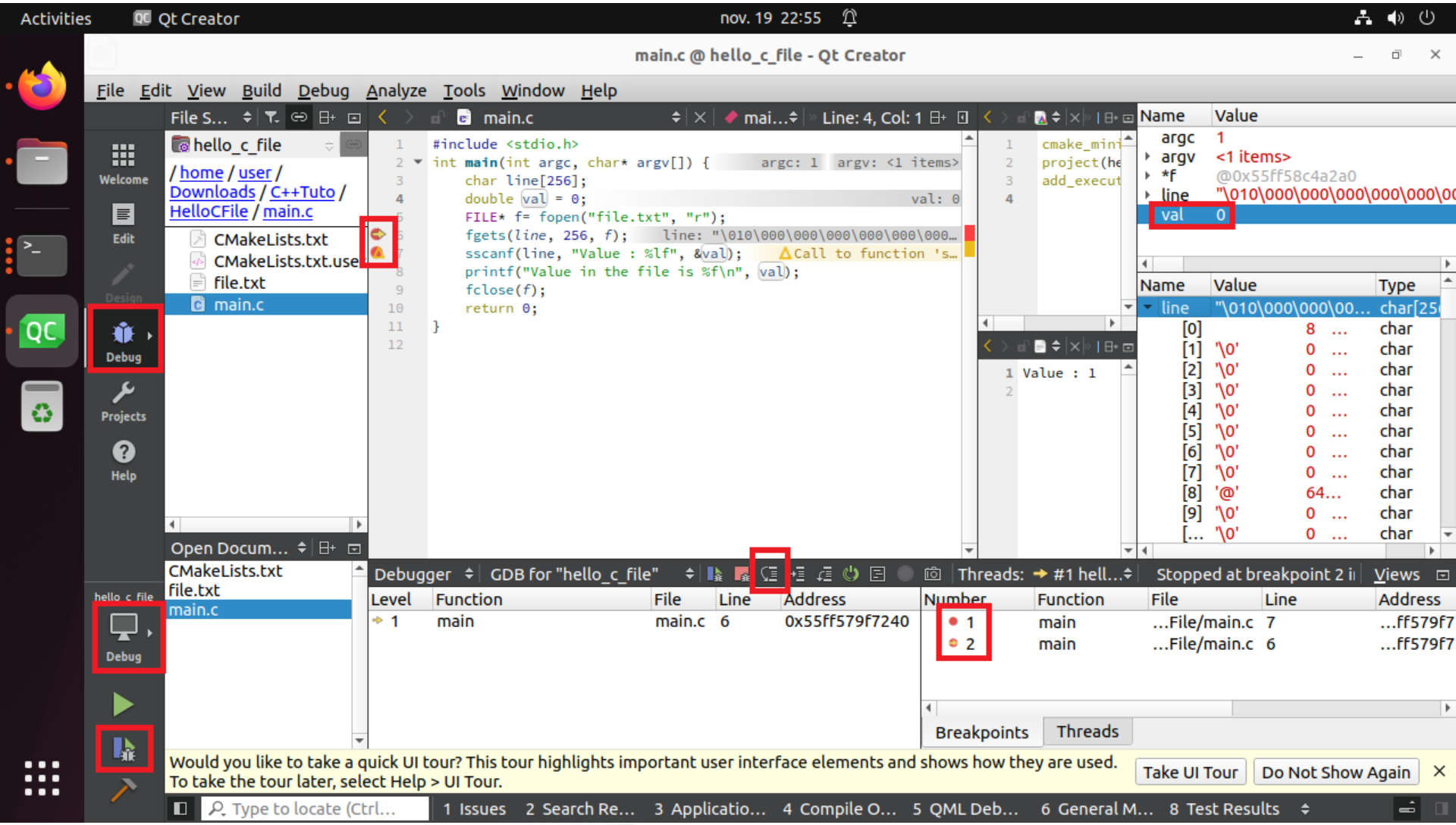
Opening and debugging a CMake project using Qt Creator



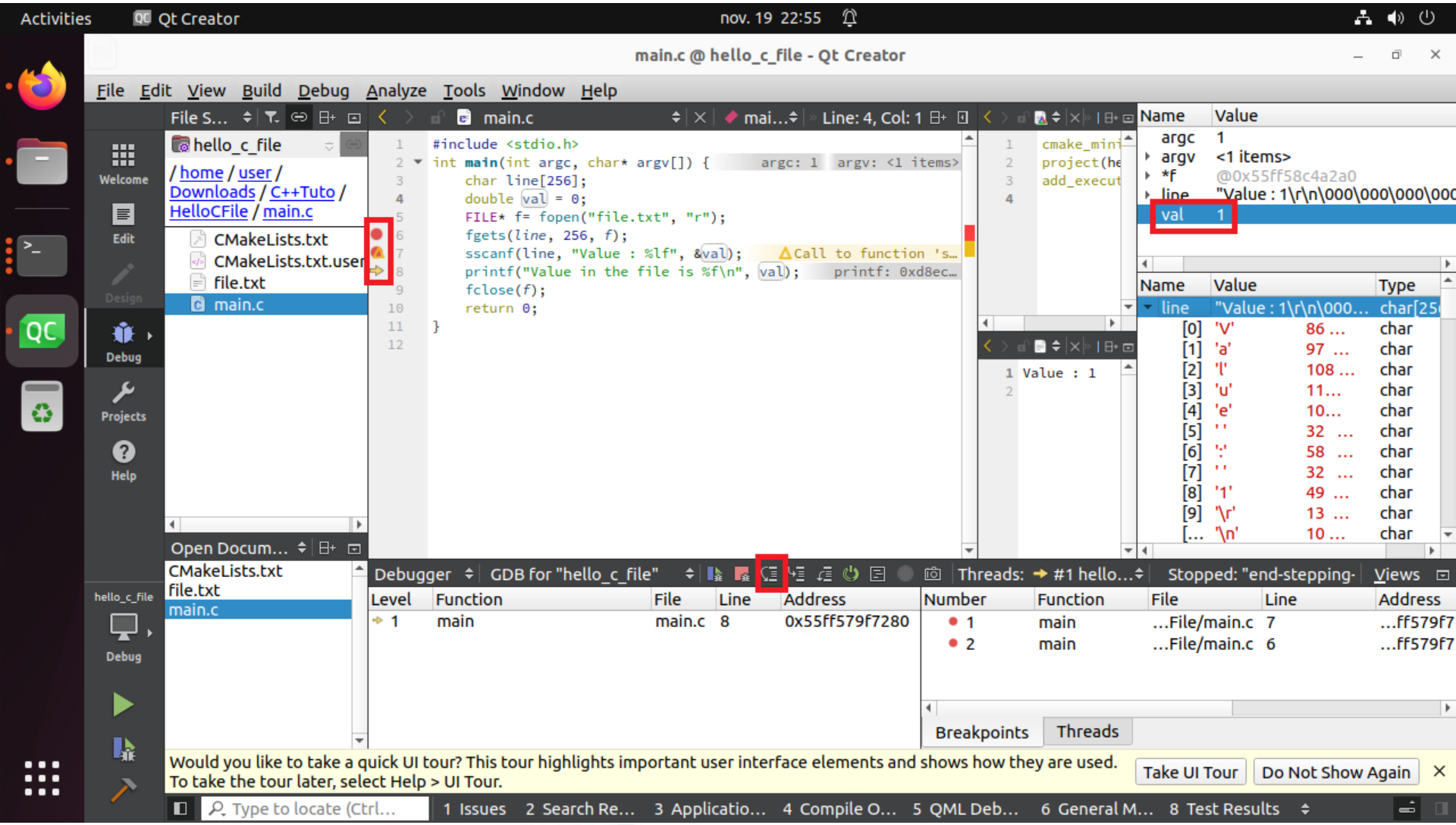
Opening and debugging a CMake project using Qt Creator



Opening and debugging a CMake project using Qt Creator



Opening and debugging a CMake project using Qt Creator



Tips Qt Creator

- The **current directory of the application** is set by default to the folder that ends with **-build-desktop**
- 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 **.pro.user** and the folder that ends with **-build-desktop** 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
  - 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

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

Visual Studio

- Visual Studio
  - Organisation

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

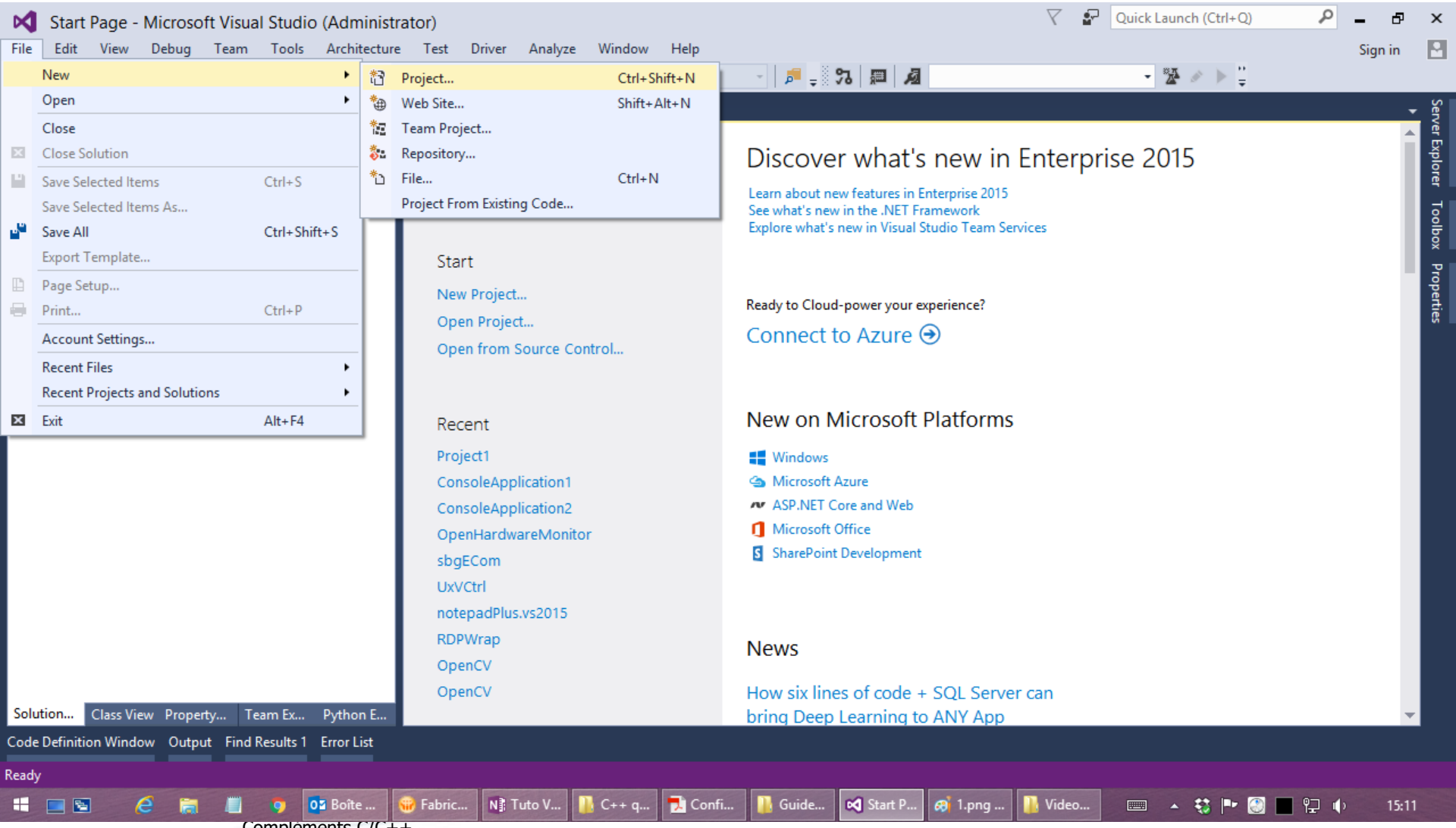
    Projet (fichiers .dsp/.vcproj/**.vcxproj+.vcxproj.filters**) : ensemble de fichiers (.c, .cpp, .h…) et informations nécessaires à la génération d'un exécutable ou bibliothèque

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

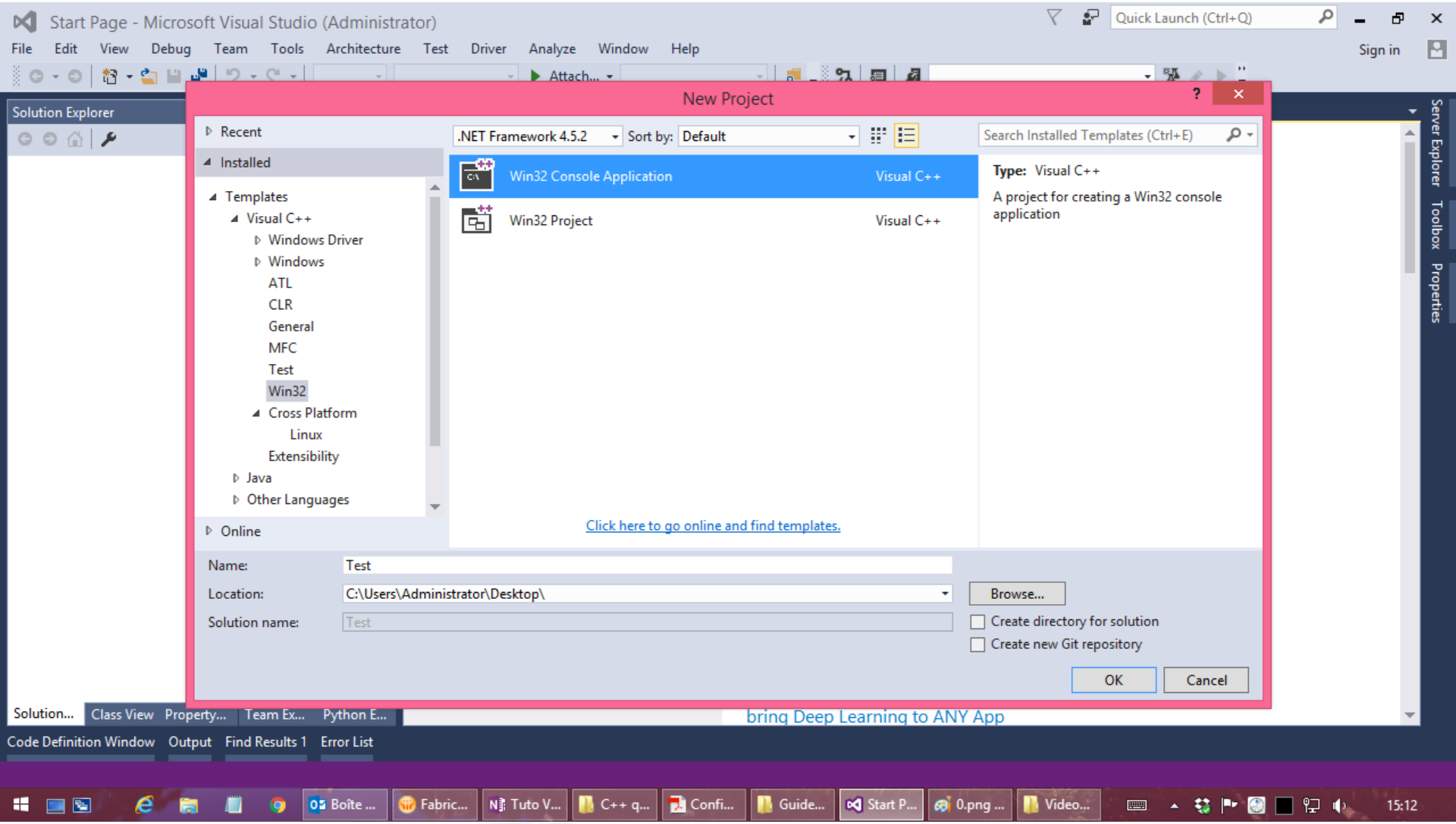
Créer un projet C simple avec Visual Studio

- 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**

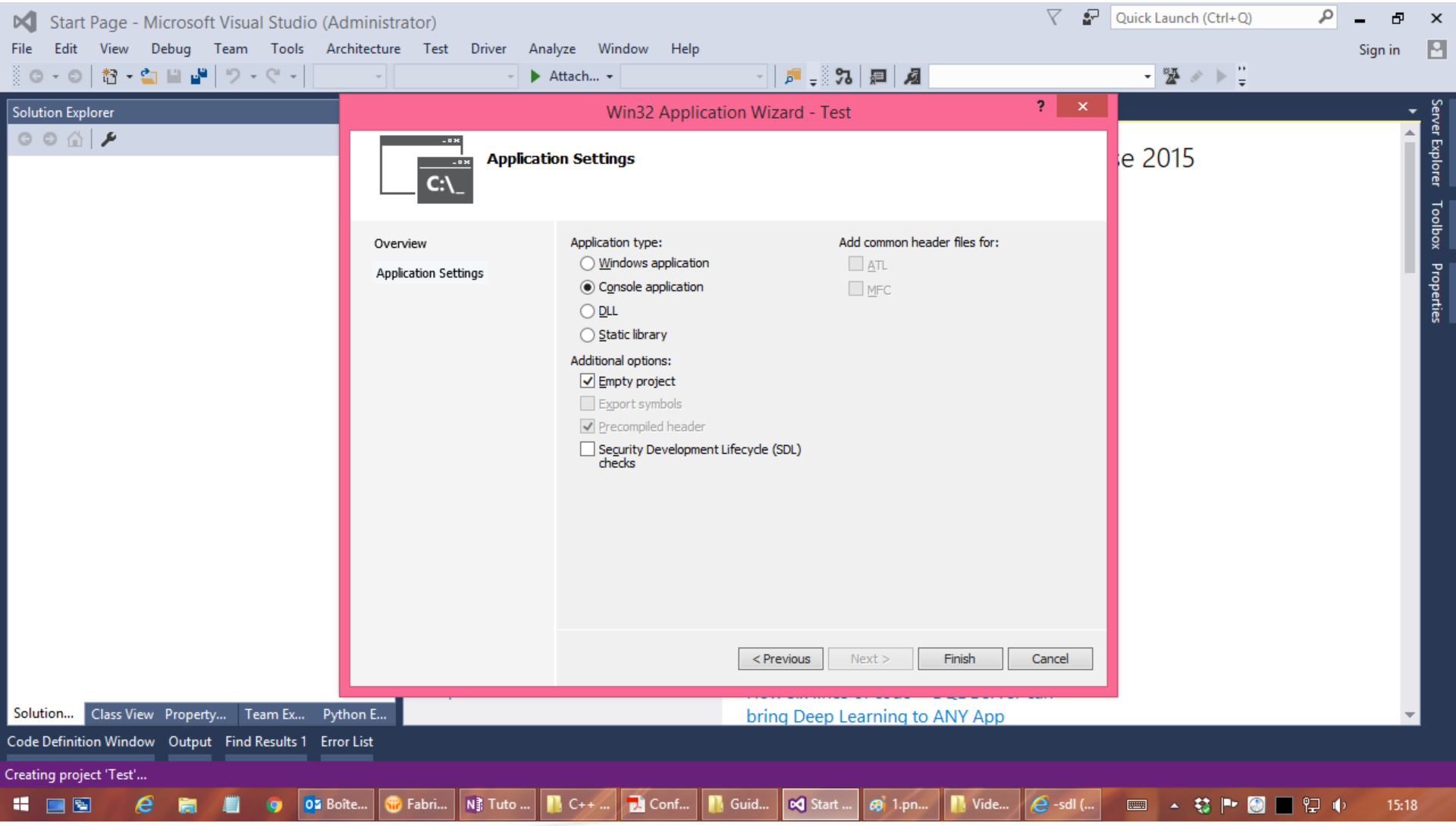
Créer un projet C simple avec Visual Studio



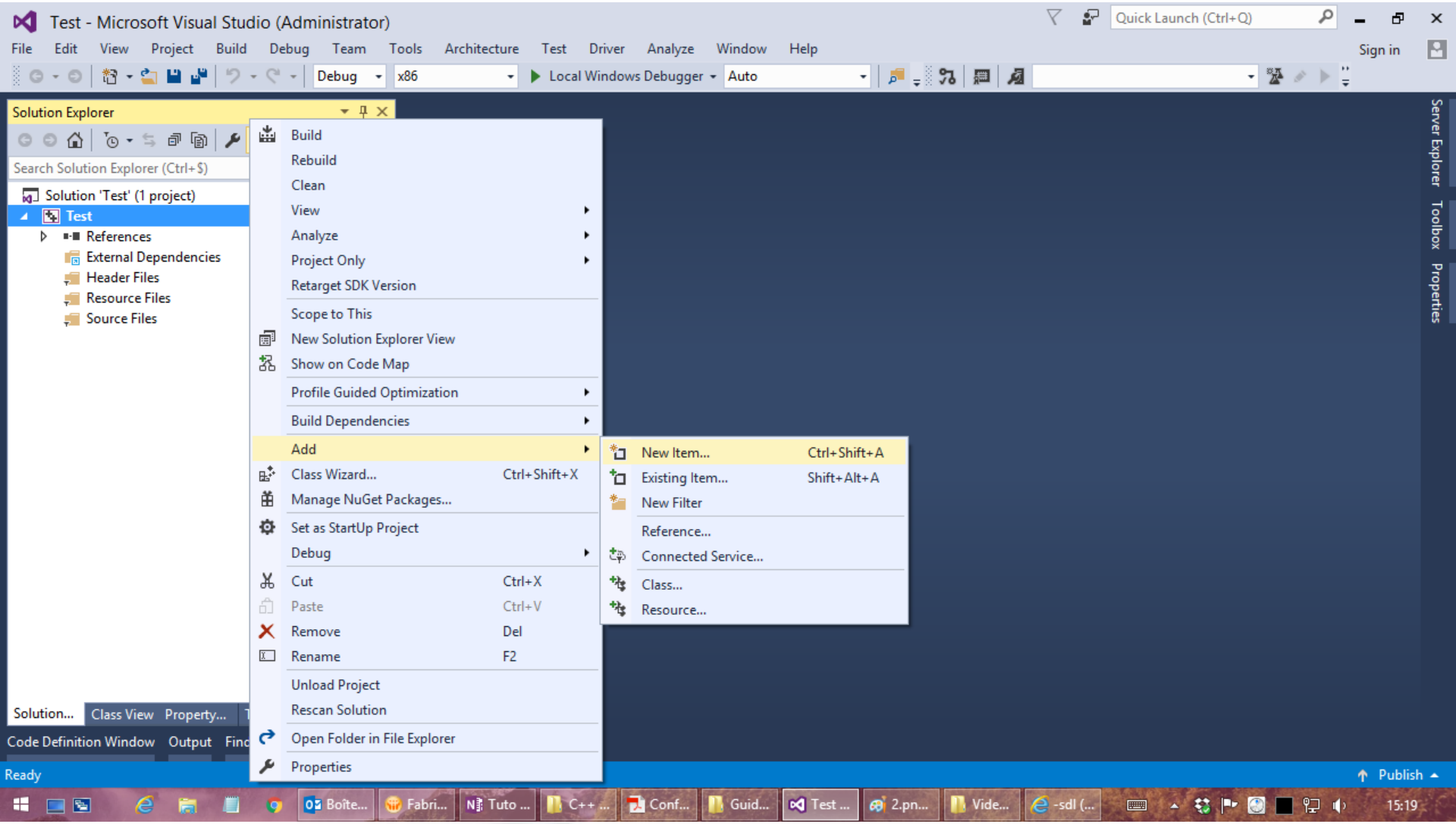
Créer un projet C simple avec Visual Studio



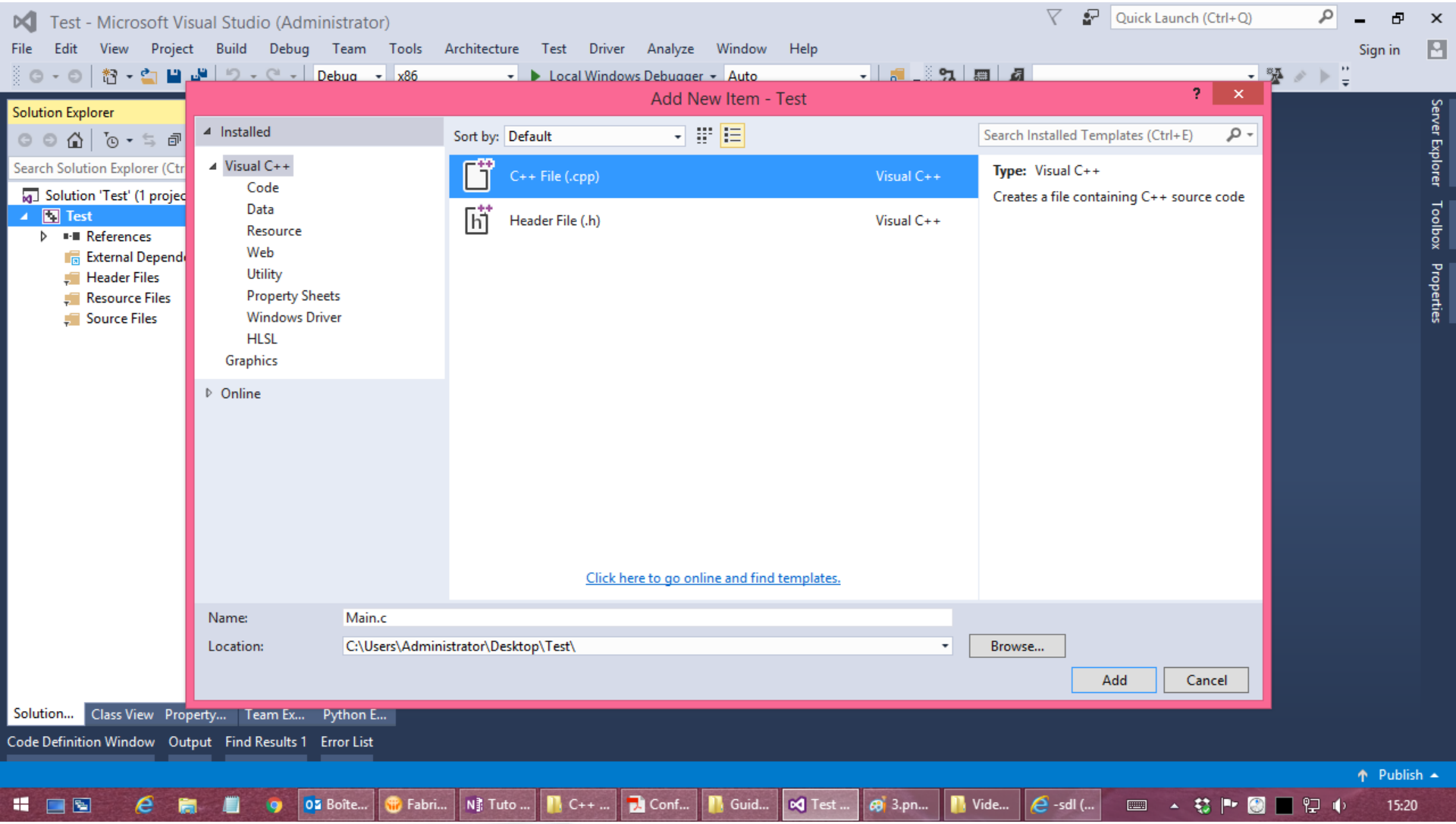
Créer un projet C simple avec Visual Studio



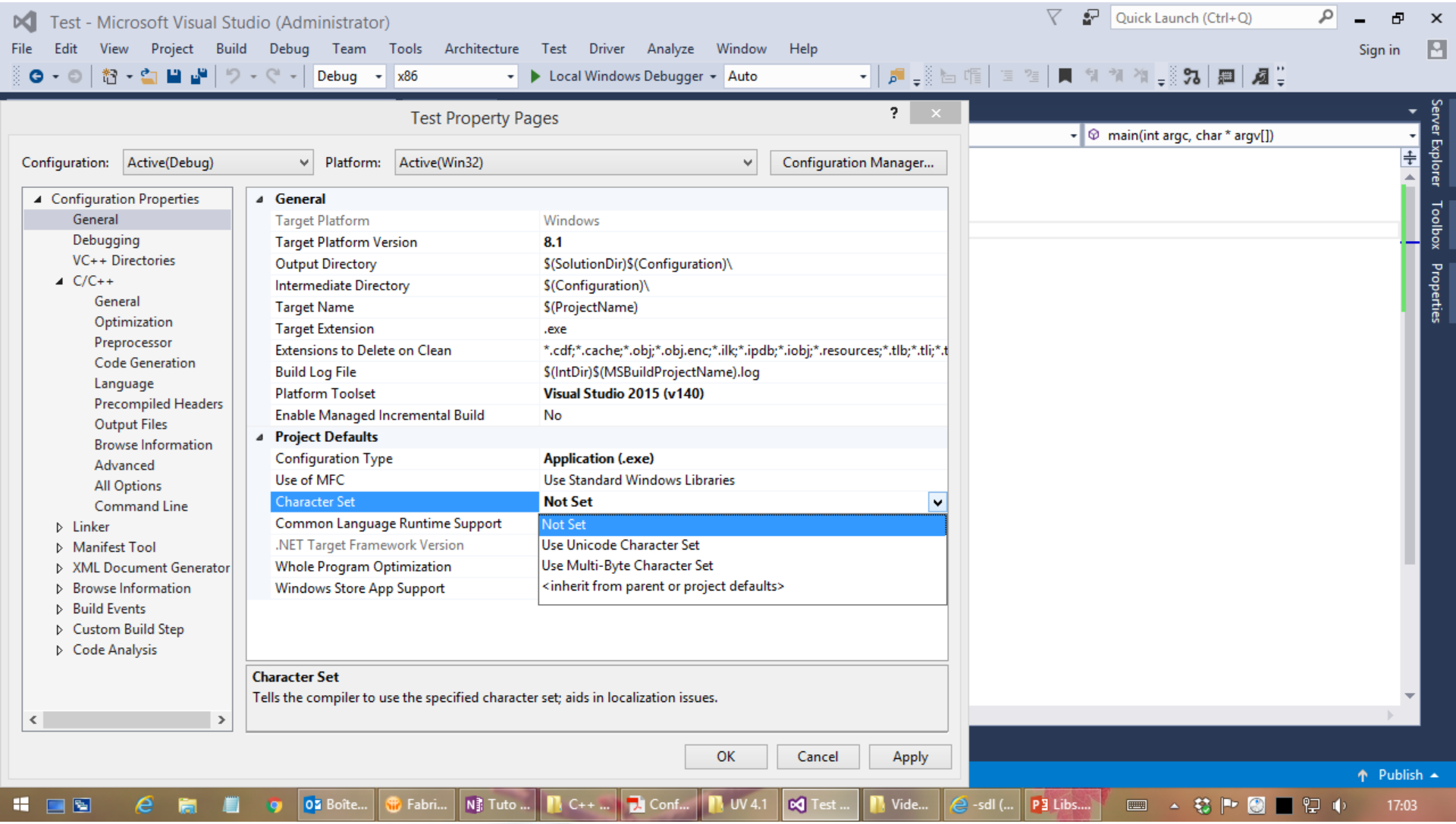
Créer un projet C simple avec Visual Studio



Créer un projet C simple avec Visual Studio



Créer un projet C simple avec Visual Studio



Créer un projet C simple 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**

Tips Visual Studio

- 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

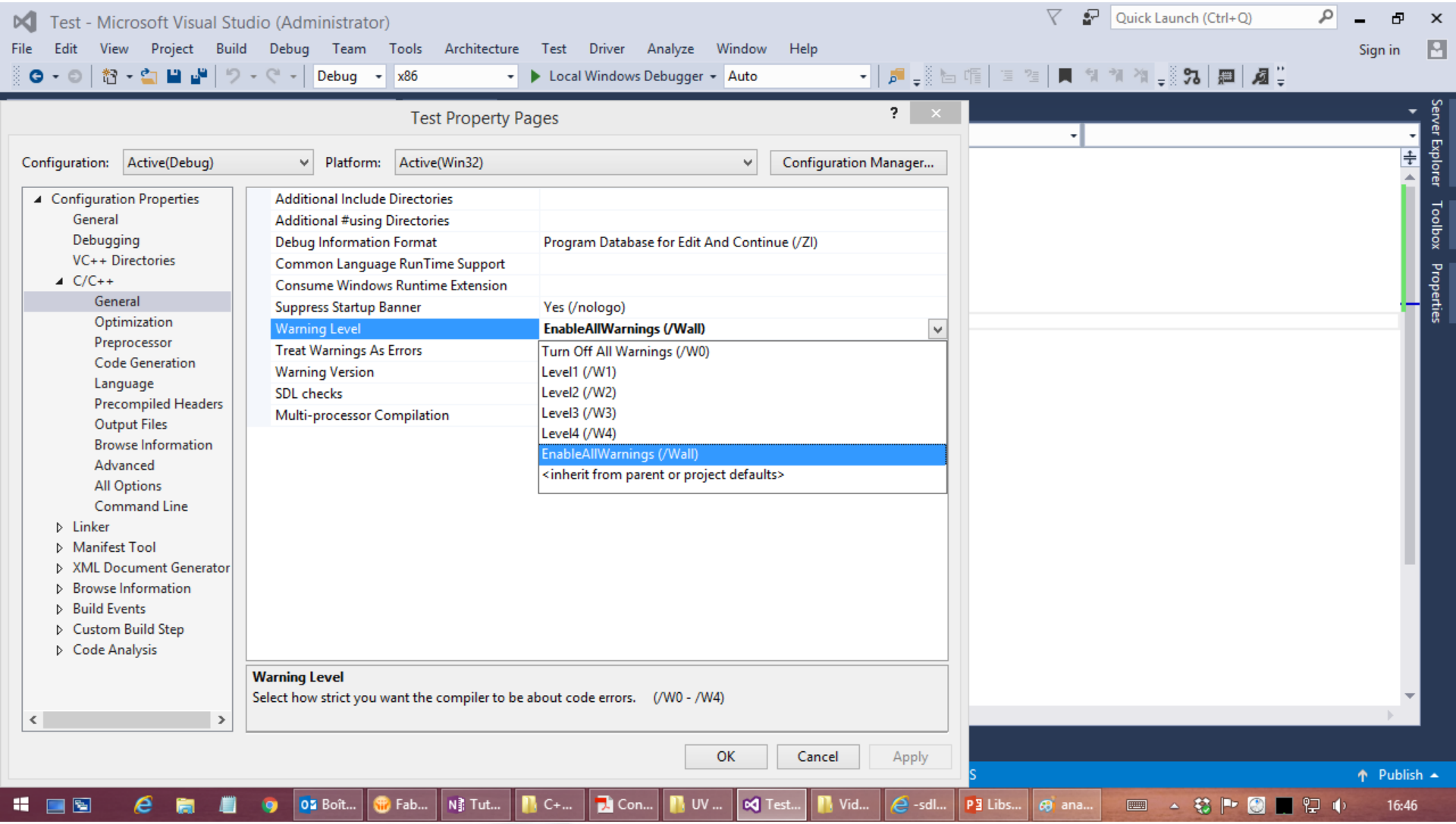
- See also http://www.ensta-bretagne.fr/lebars/tutorials

Débugger rapidement avec Visual Studio

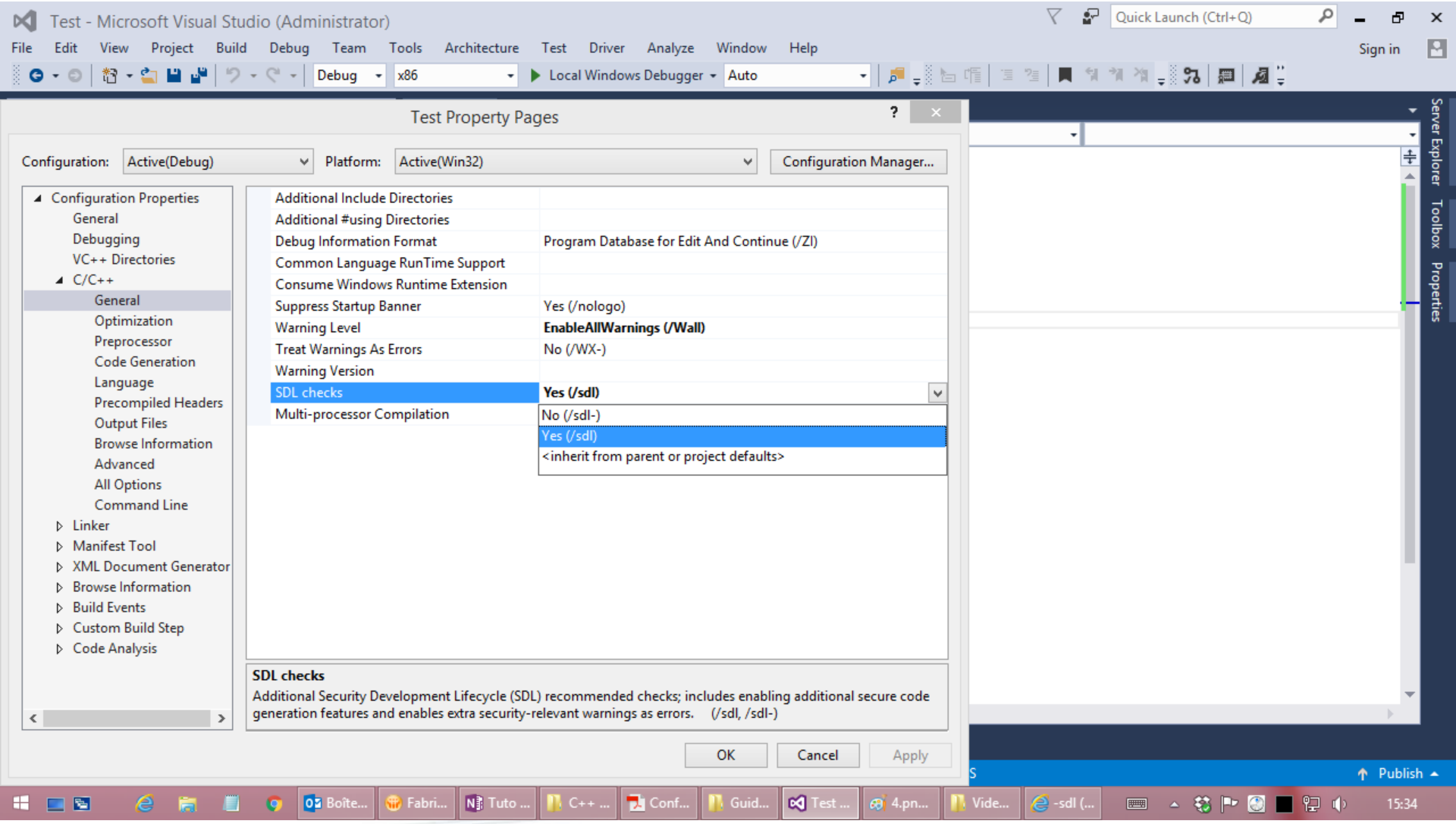
- Analyse statique de code :
  - Activer tous les avertissements (**/W4** ou même **/Wall**)
  - Enable SDL checks (**/sdl**) : makes some aggressive checks and turns some warnings into errors, therefore might need to be **disabled for compatibility** with some libraries, however can be good to enable 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.

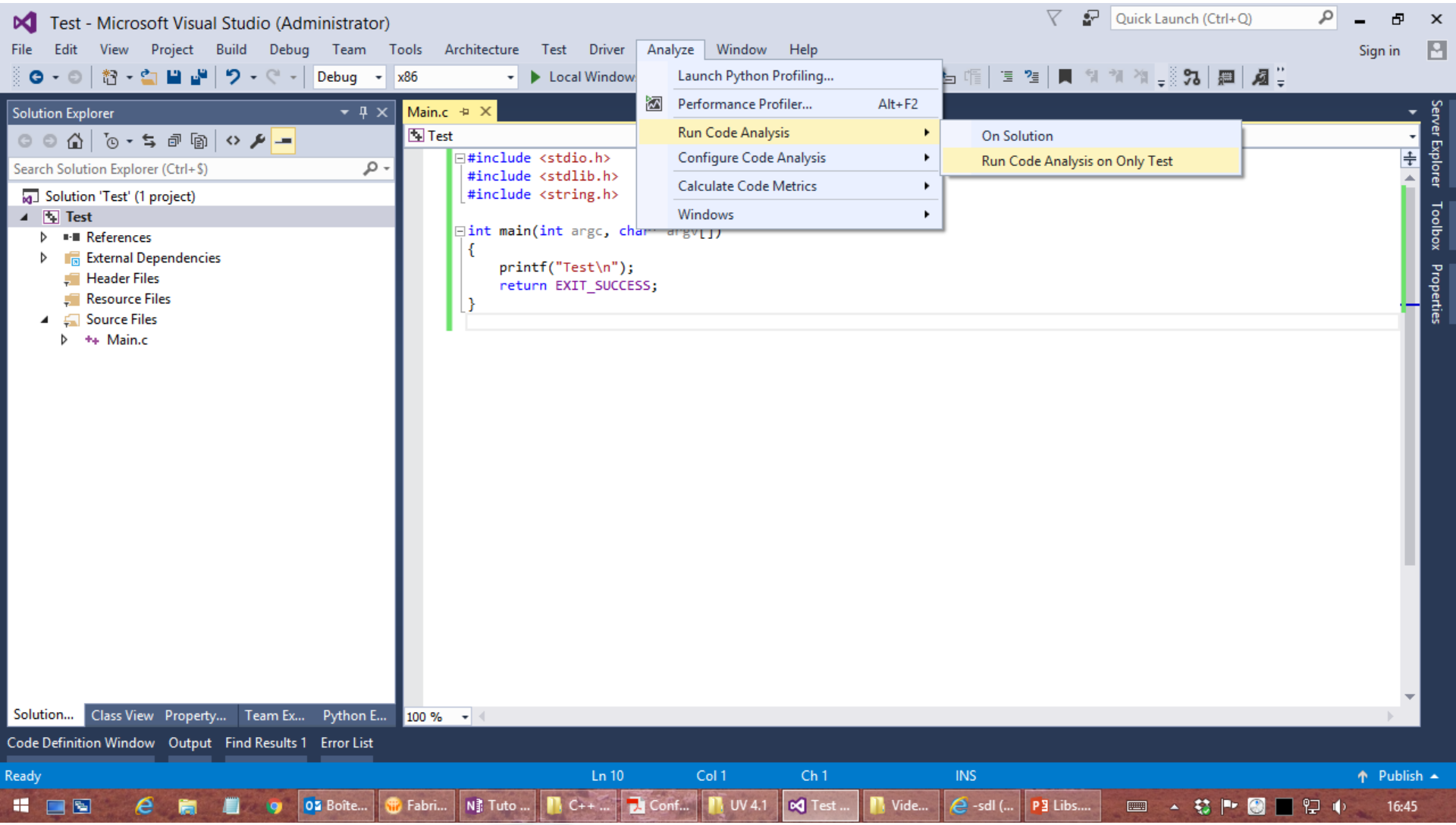
Débugger rapidement avec Visual Studio



Débugger rapidement avec Visual Studio



Débugger rapidement avec Visual Studio



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() :

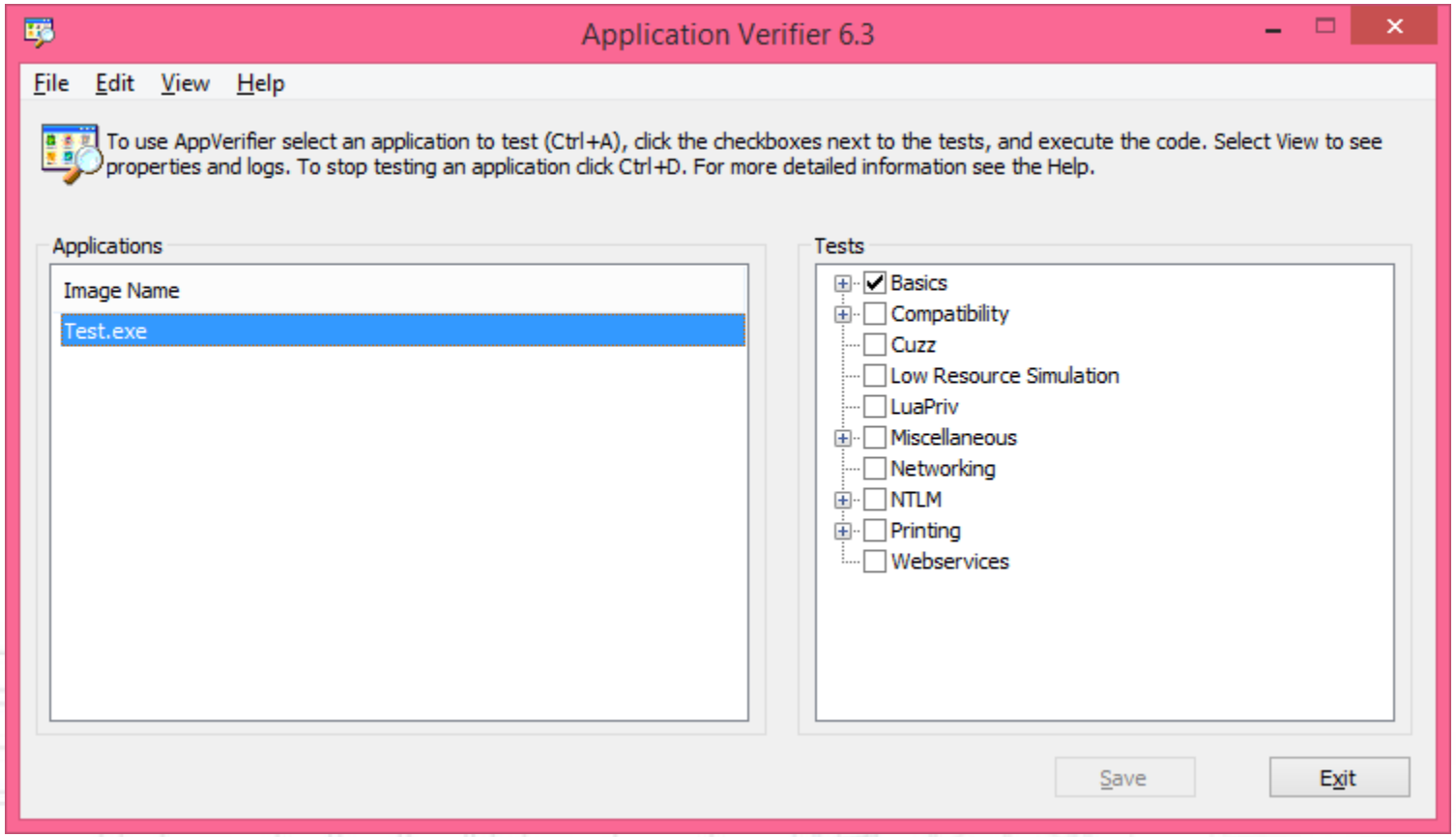
```
_CrtSetDbgFlag(_CRTDBG_ALLOC_MEM_DF |
  _CRTDBG_LEAK_CHECK_DF);
```

  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

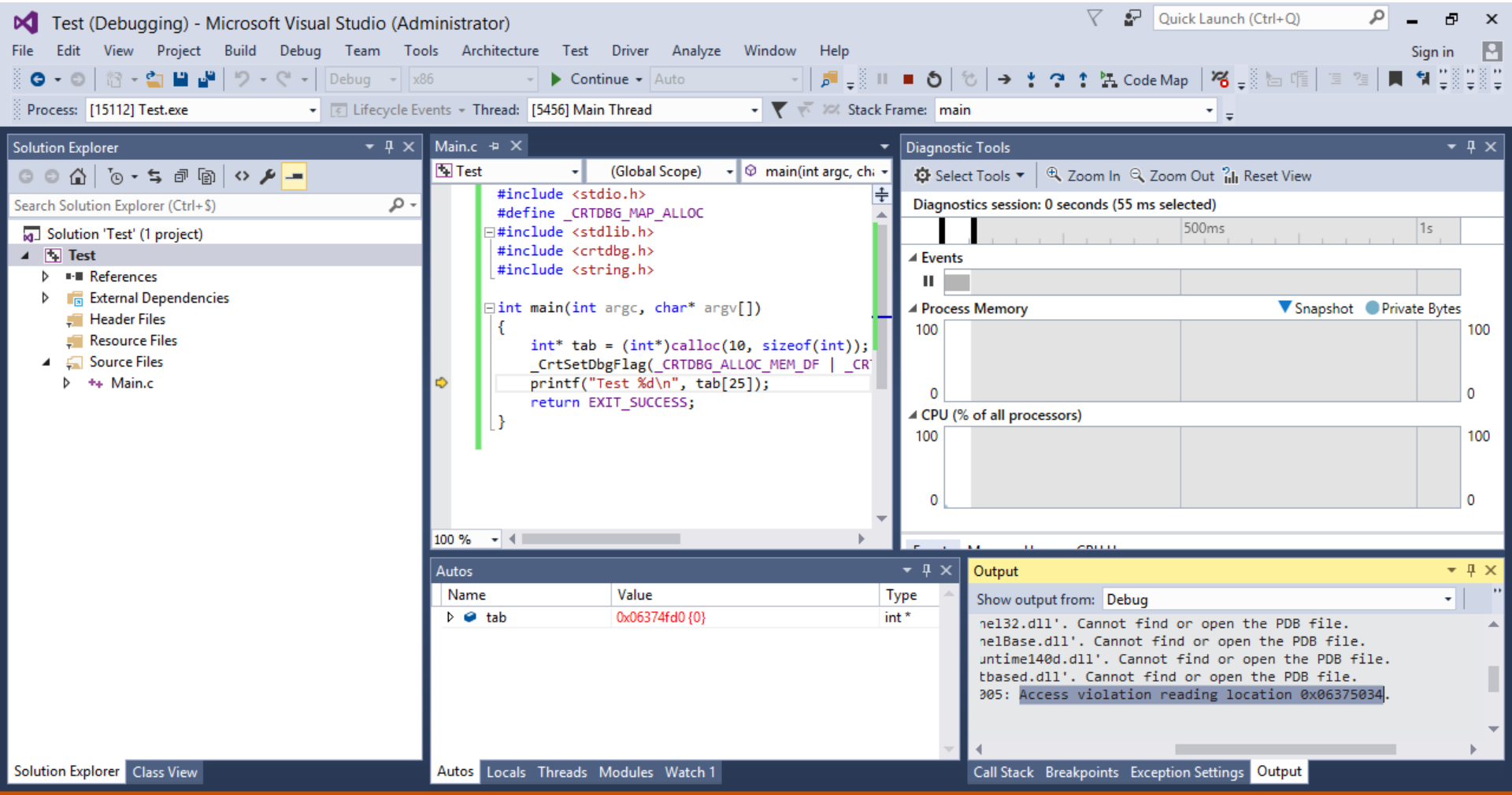
Débugger rapidement avec Visual Studio

- Analyse dynamique de code :
  - Exemple : programme Test.exe qui **alloue un tableau de 10 entiers**, essaye d'**afficher la valeur du 25ème** 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)…

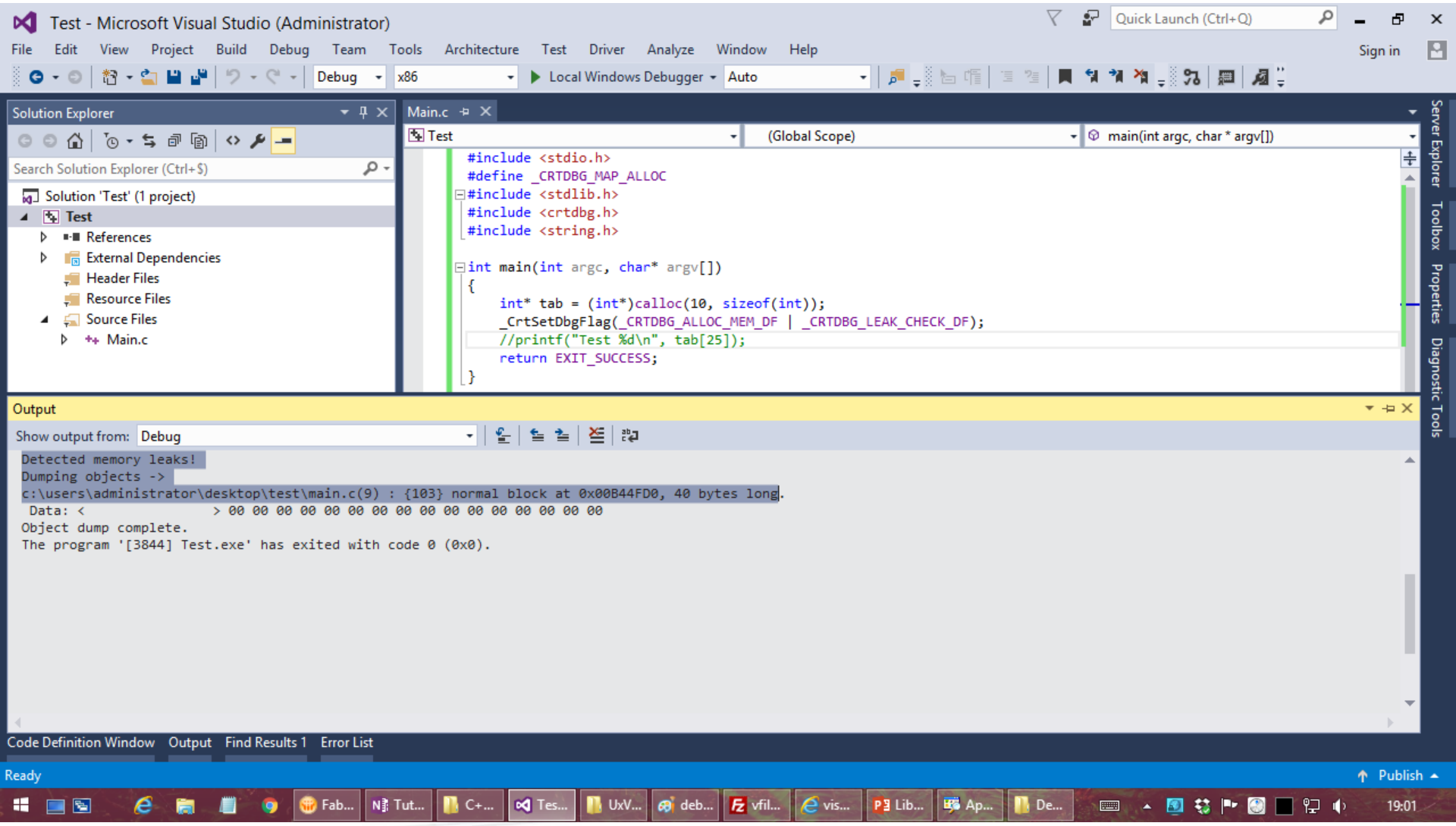
Débugger rapidement avec Visual Studio



Débugger rapidement avec Visual Studio



Débugger rapidement avec Visual Studio



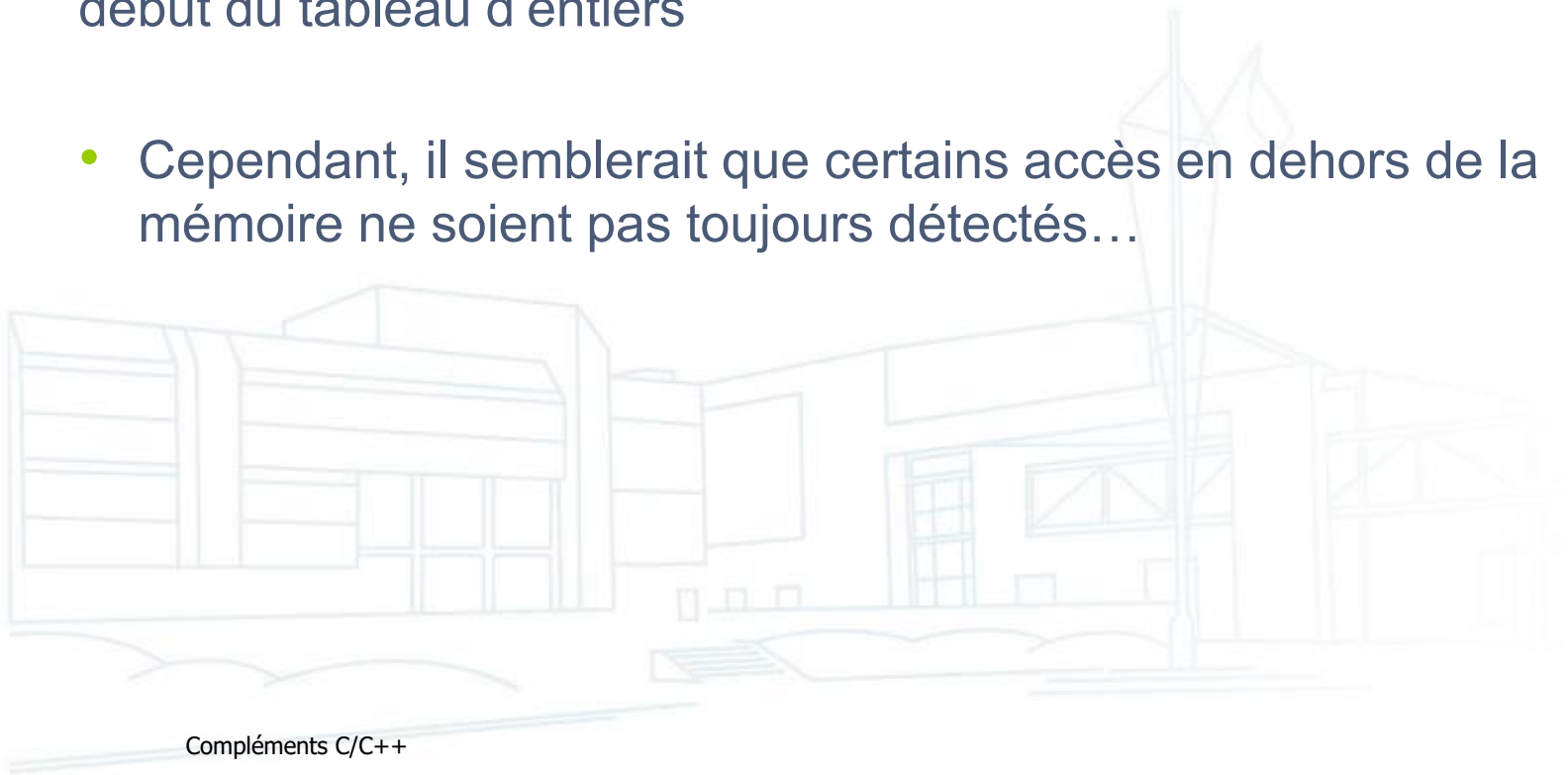
Débugger rapidement avec Visual Studio

- Analyse dynamique de code :
  - On vérifie bien que dans l'exemple

    0x06375034-0x06374fd0=0x64->100->25*sizeof(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 make wget
    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**

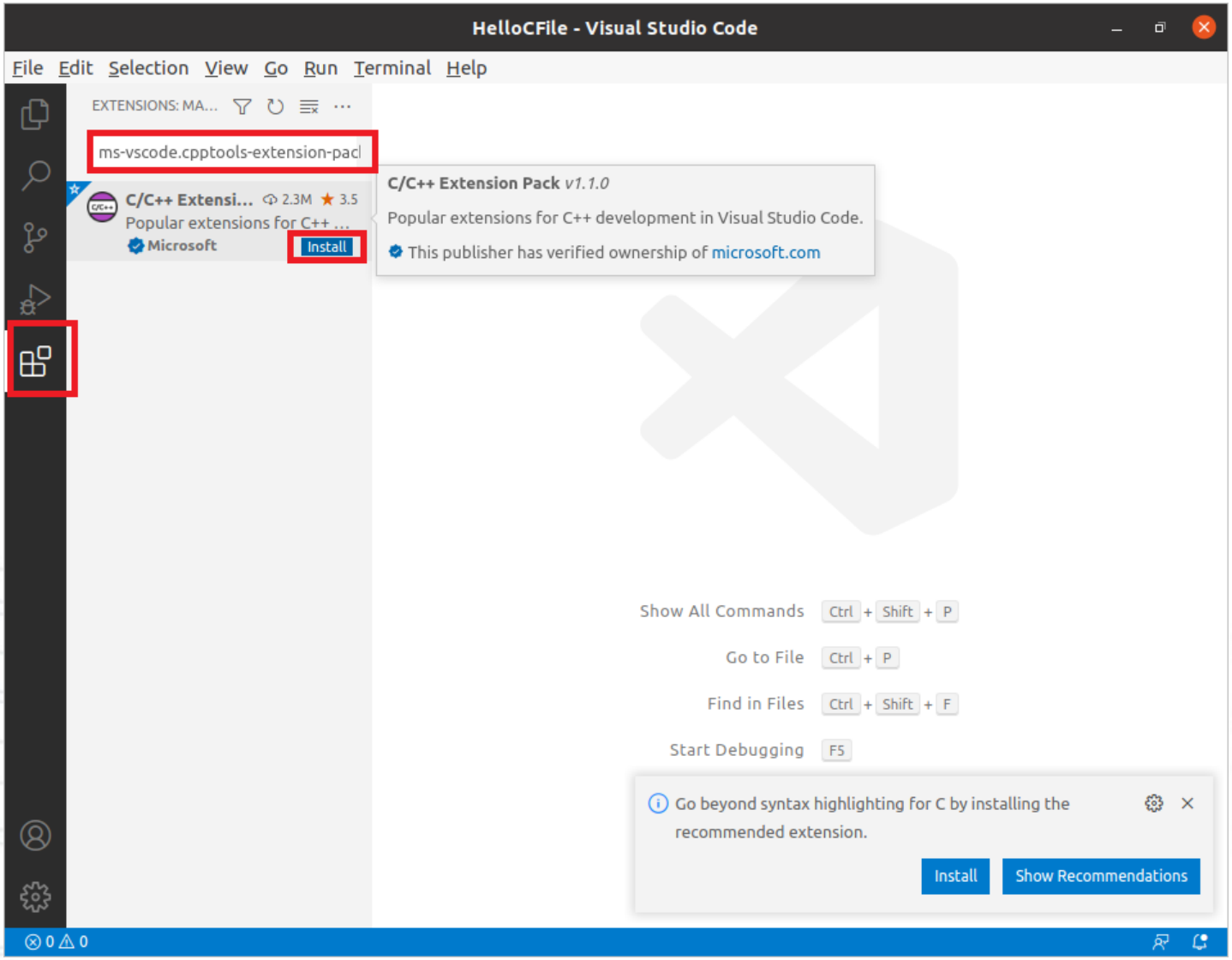
Visual Studio Code

- Visual Studio Code CMake support
  - Tips

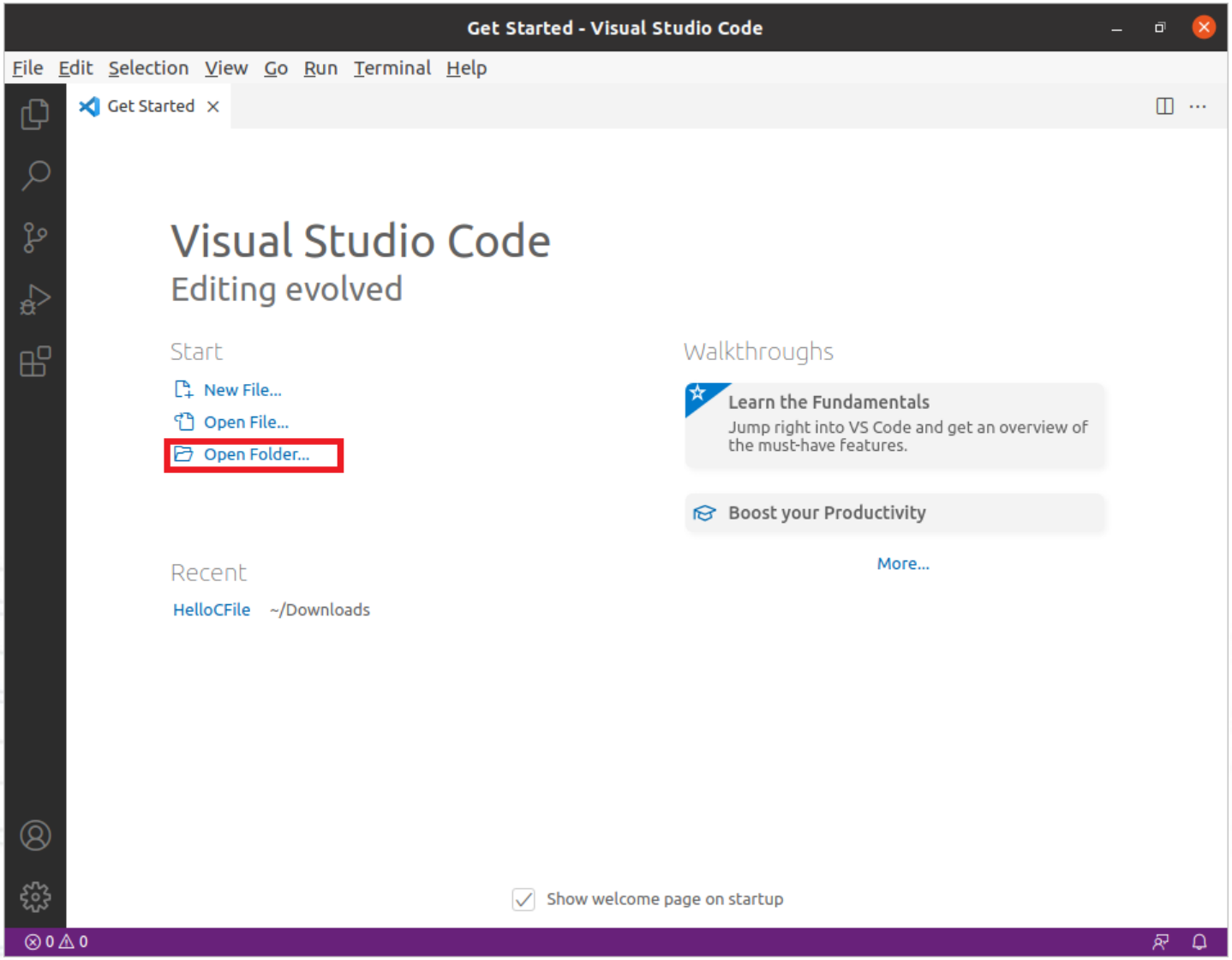
    Pour changer le dossier courant qui est par défaut "${workspaceFolder}/build" lors de l'exécution en mode débuggage, créer dans le dossier du projet **.vscode/settings.json** avec dedans e.g. (si le programme à débugguer cherche à ouvrir des fichiers dans le dossier où se trouve CMakeLists.txt) :

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

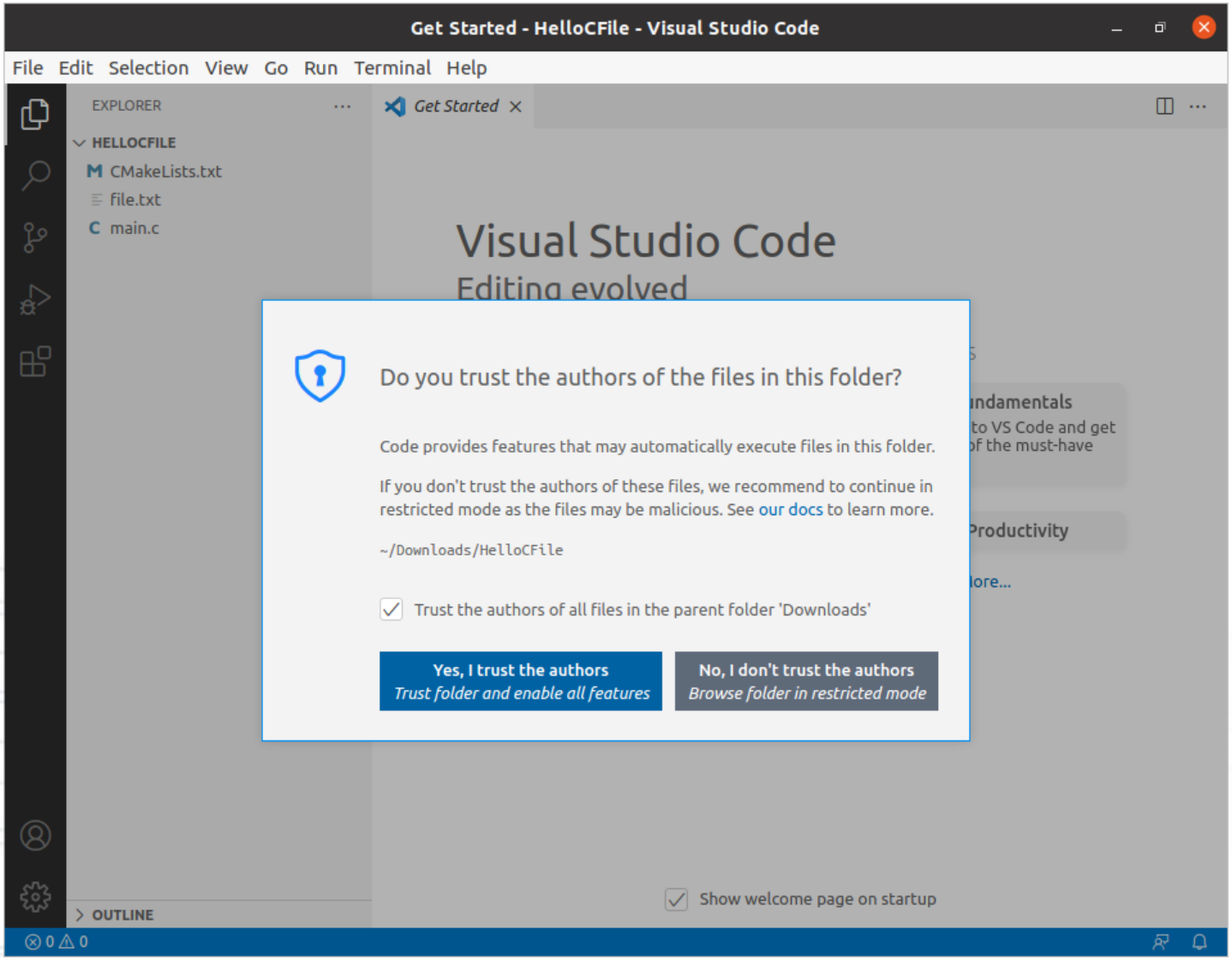
Visual Studio Code



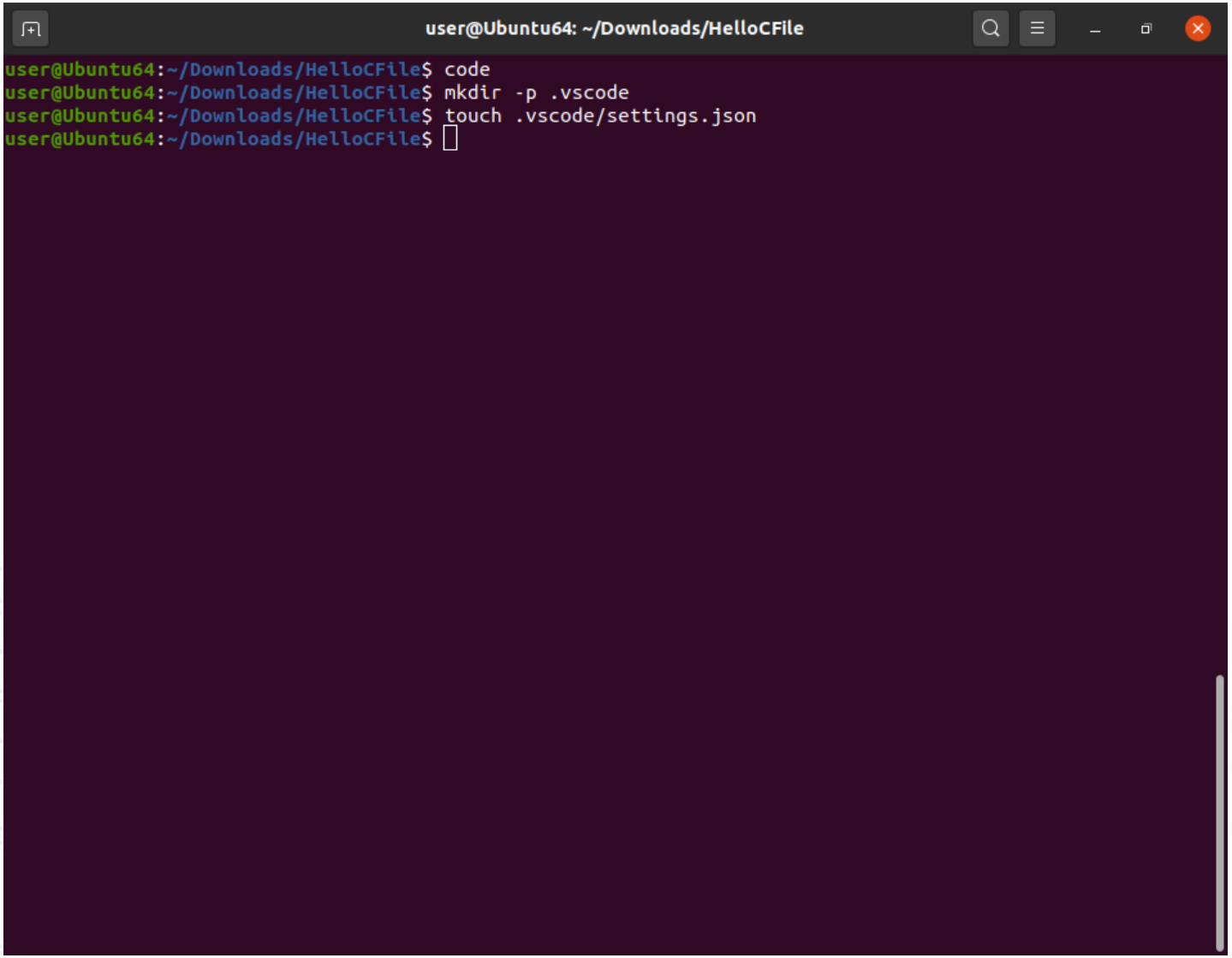
Visual Studio Code



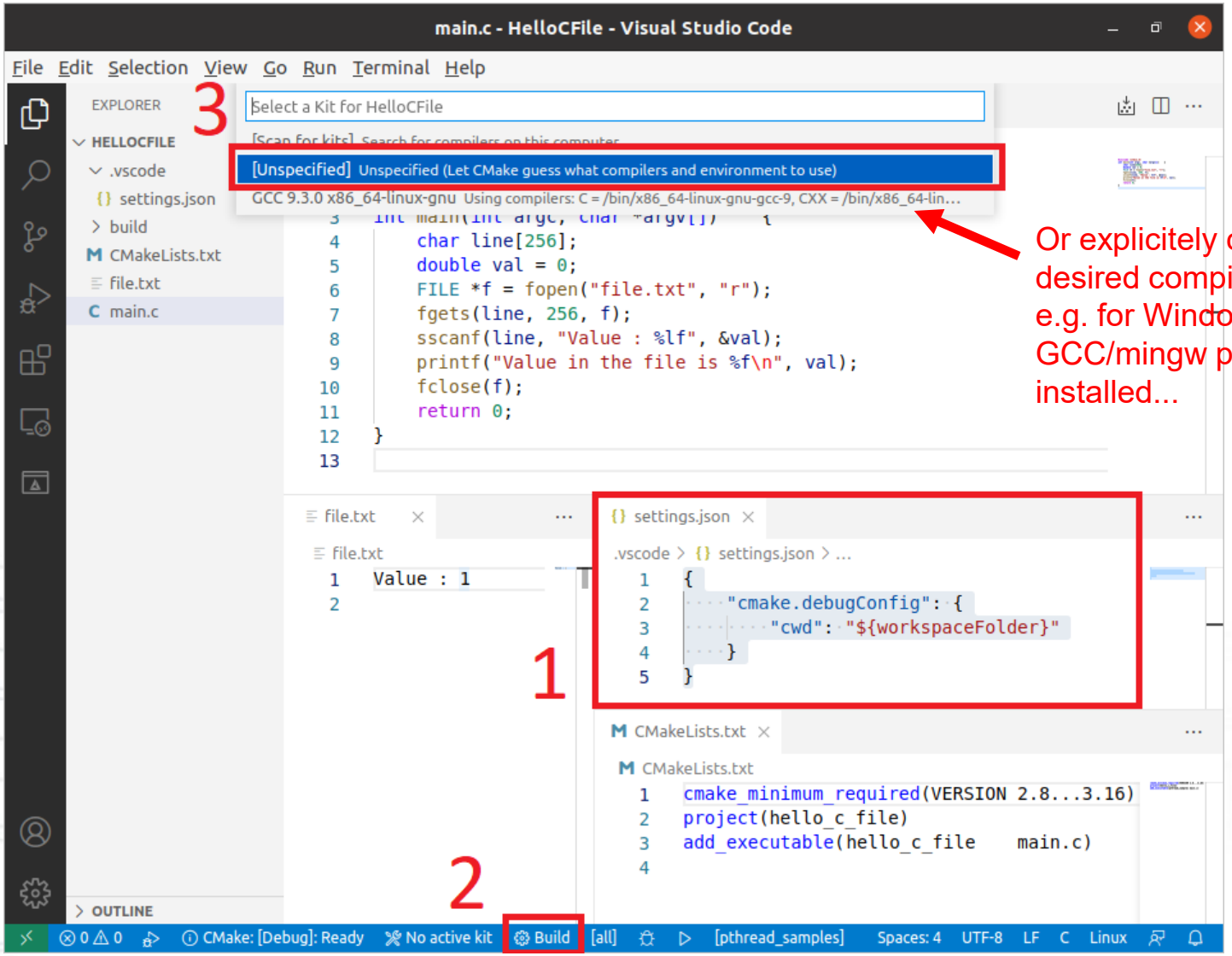
Visual Studio Code



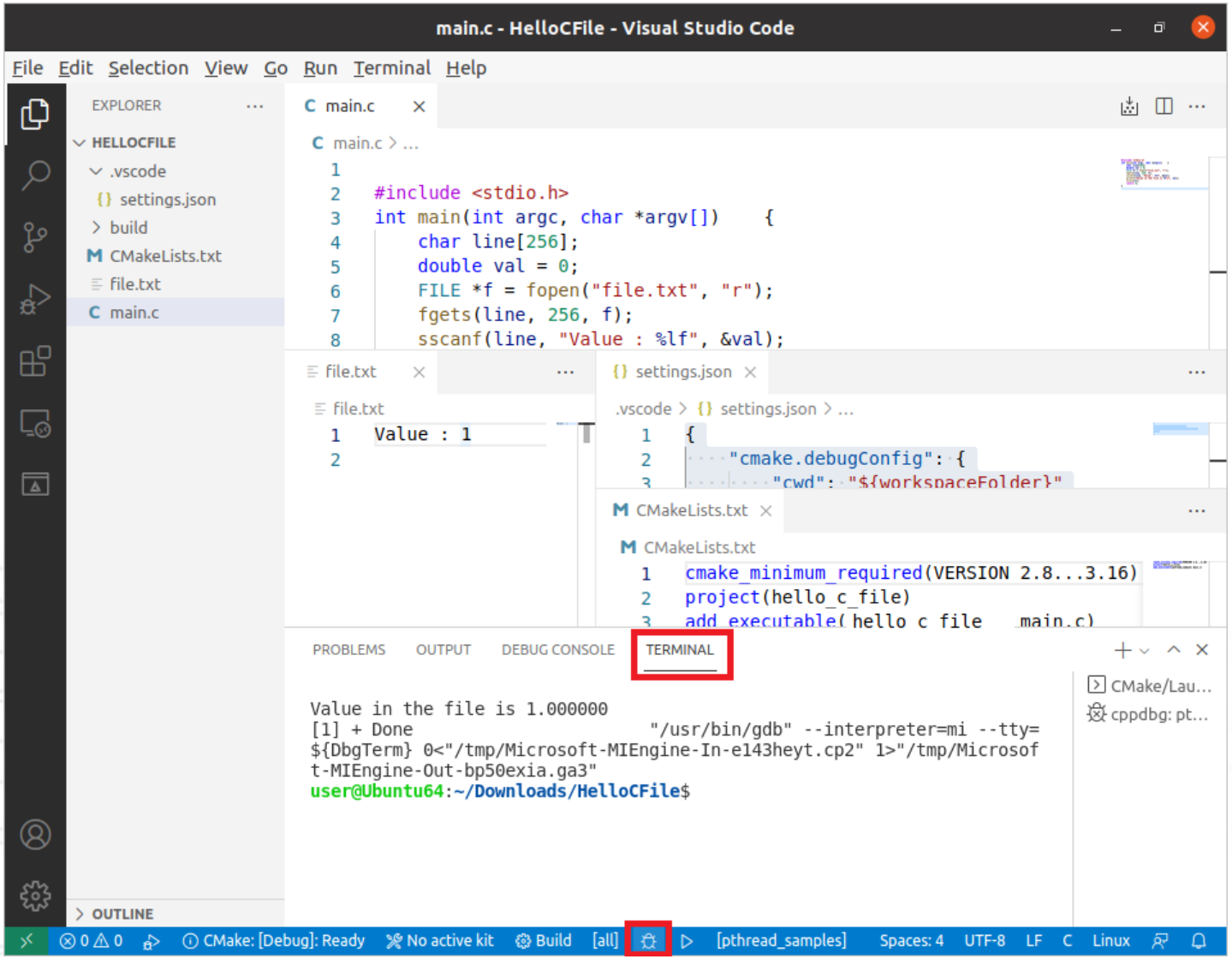
Visual Studio Code



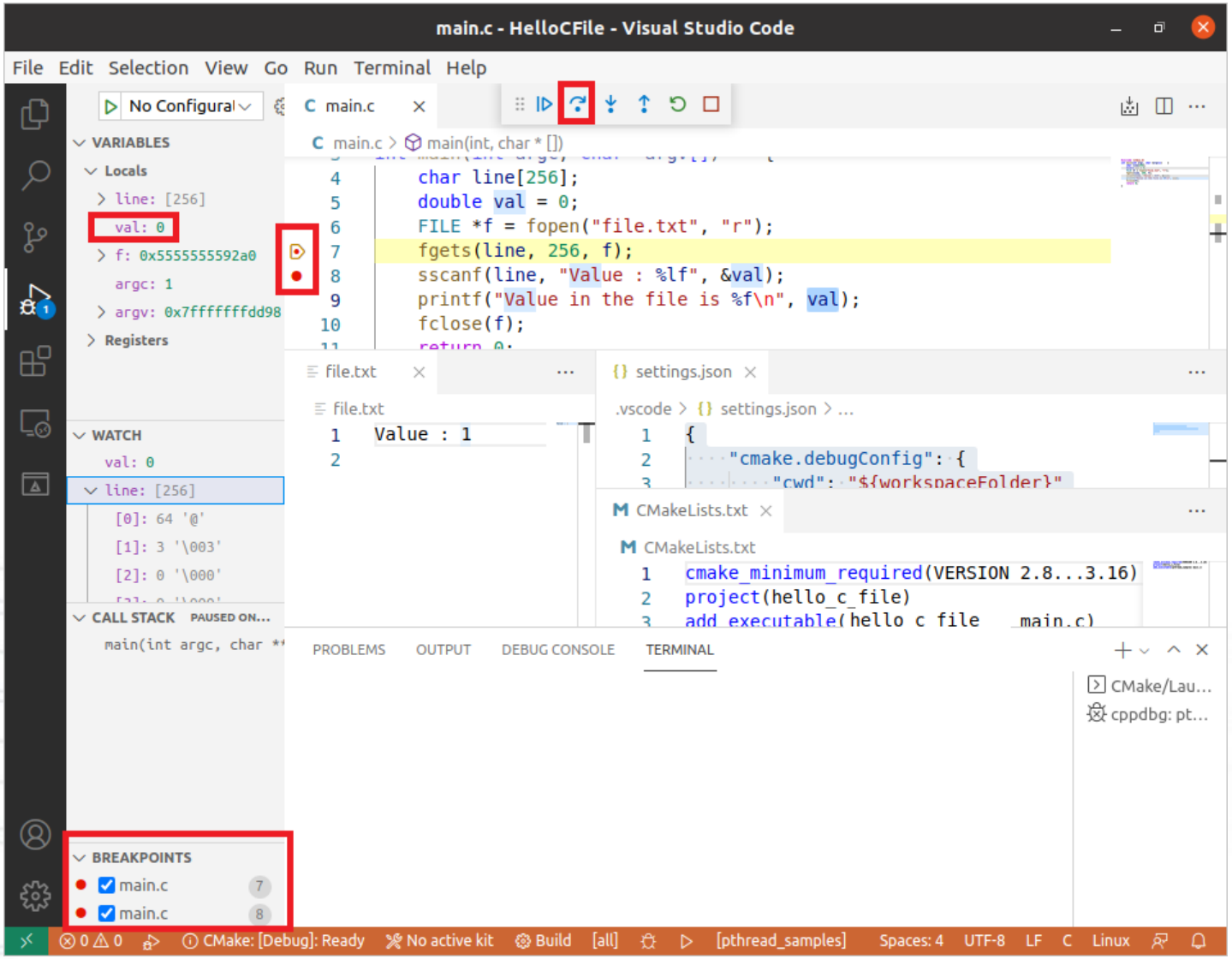
Visual Studio Code



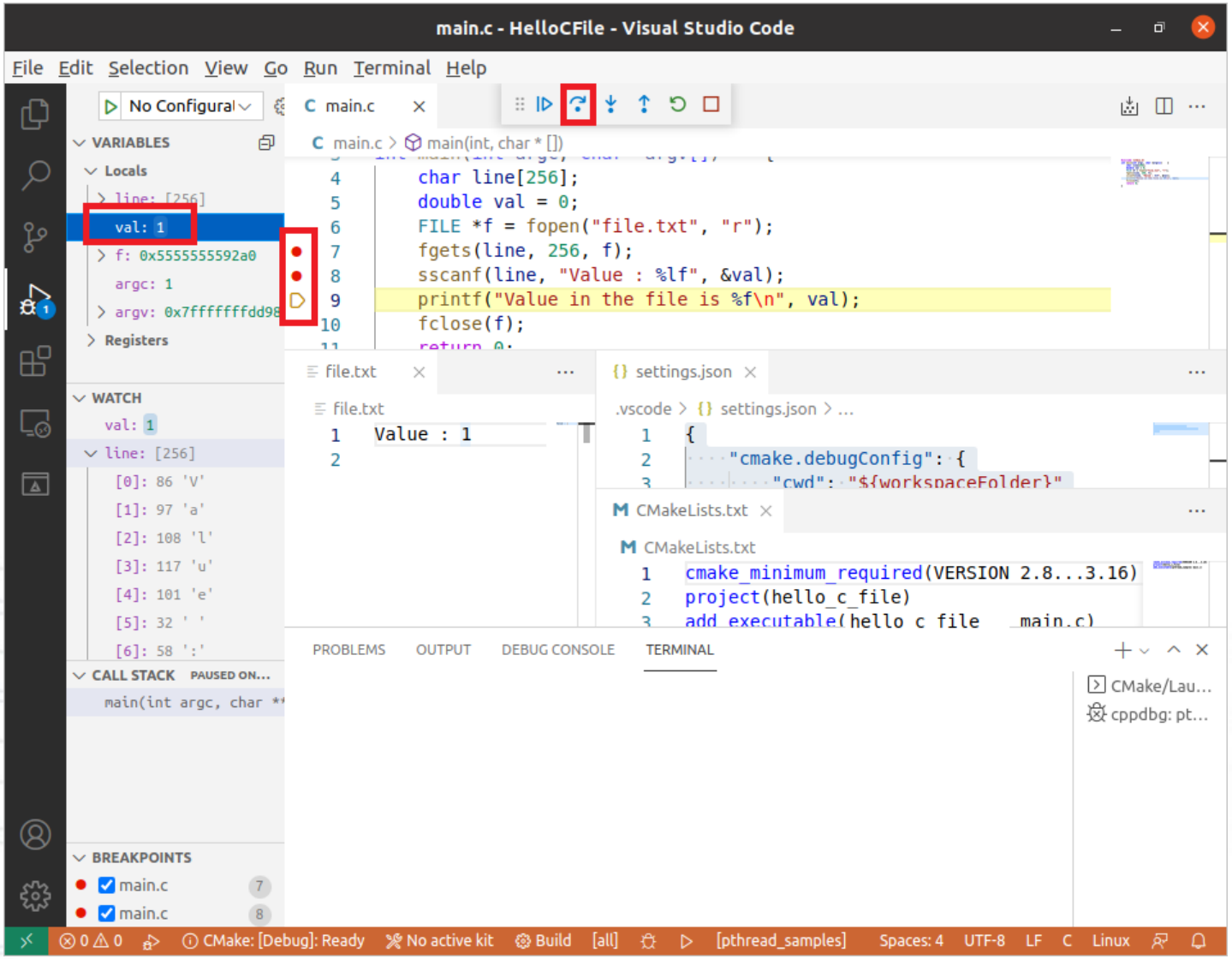
Visual Studio Code



Visual Studio Code



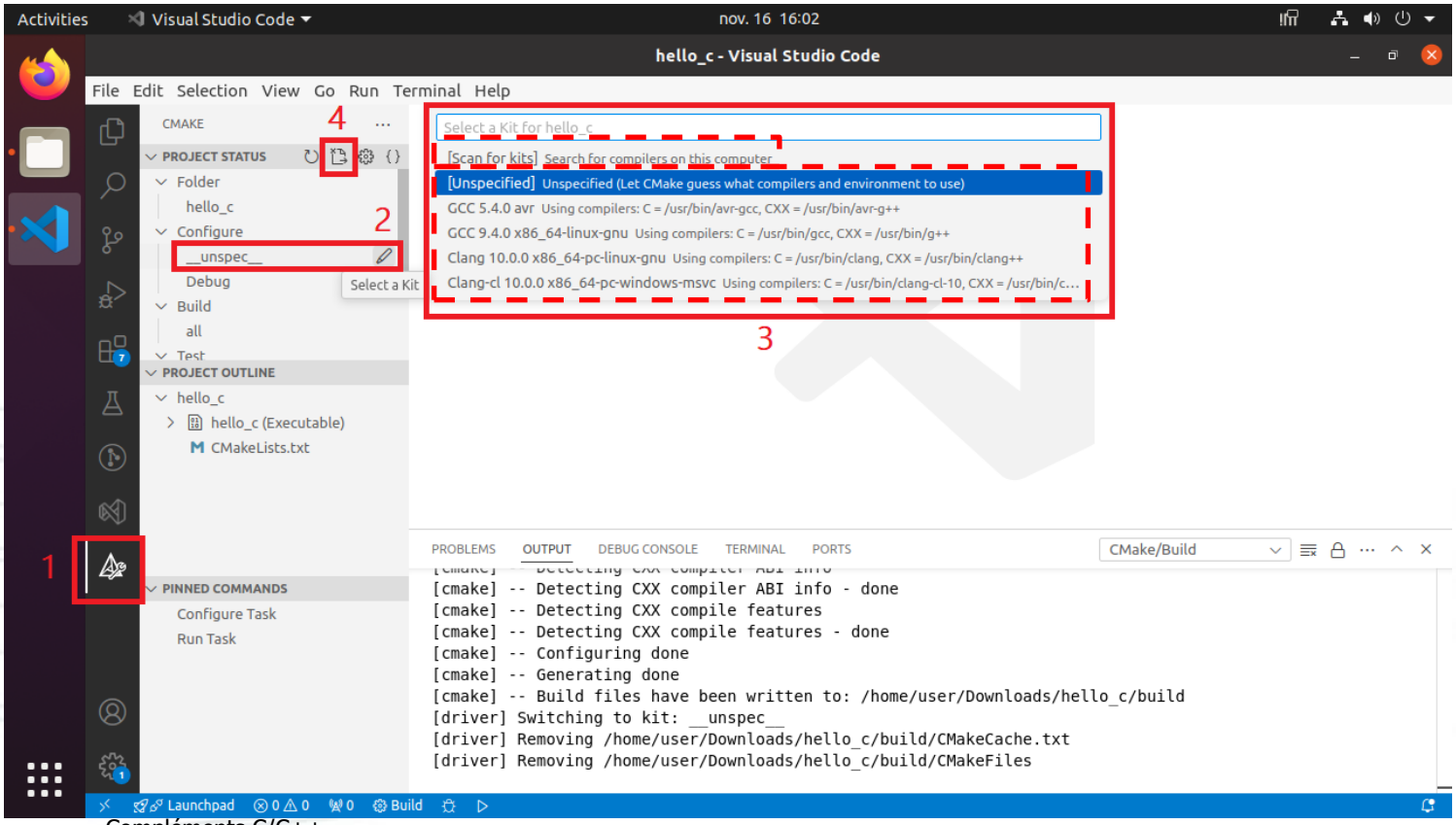
Visual Studio Code



Visual Studio Code

- Visual Studio Code CMake support
  - Tips

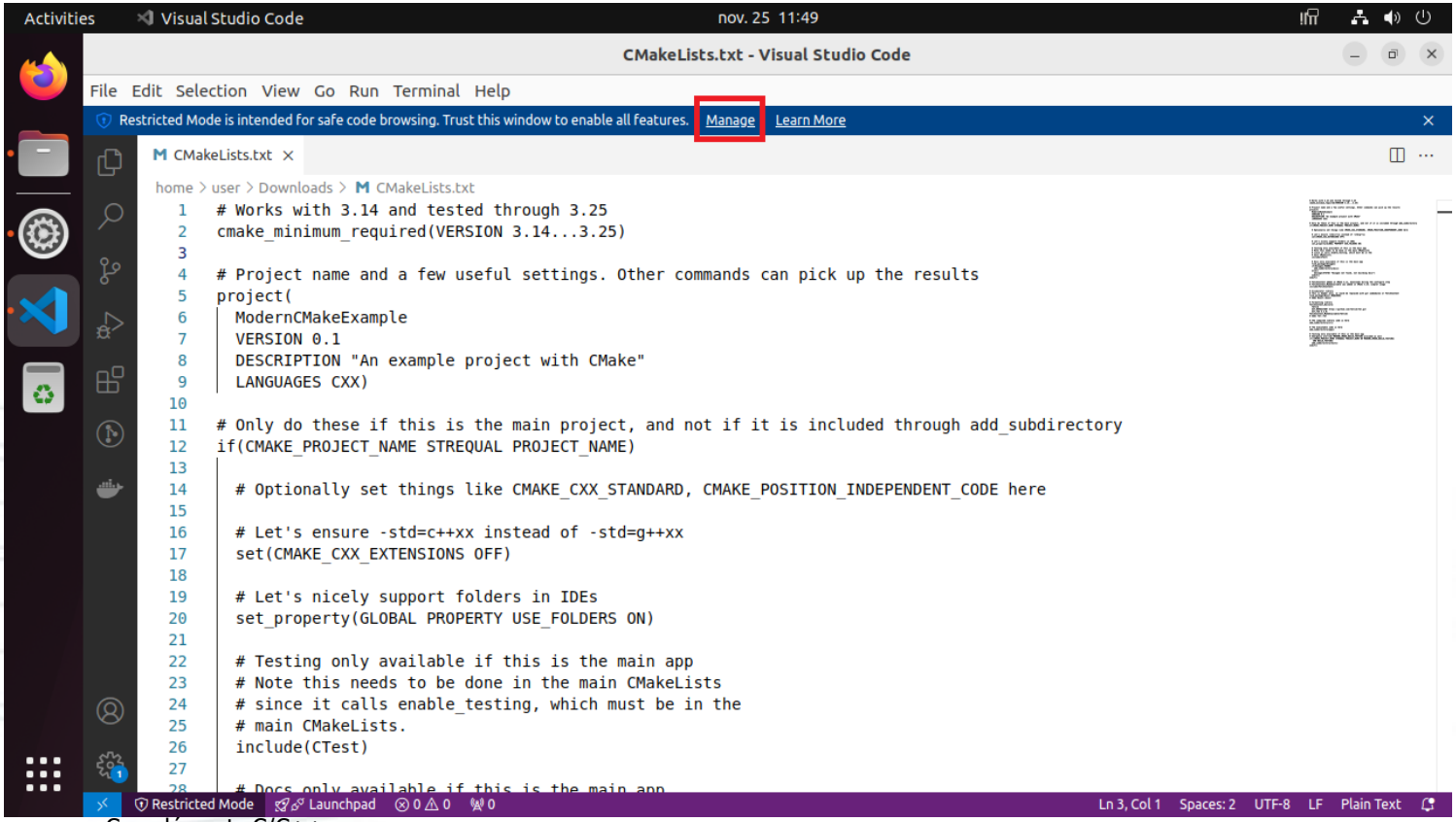
    Changing the compiler used by CMake:



Visual Studio Code

- Visual Studio Code CMake support
  - Tips

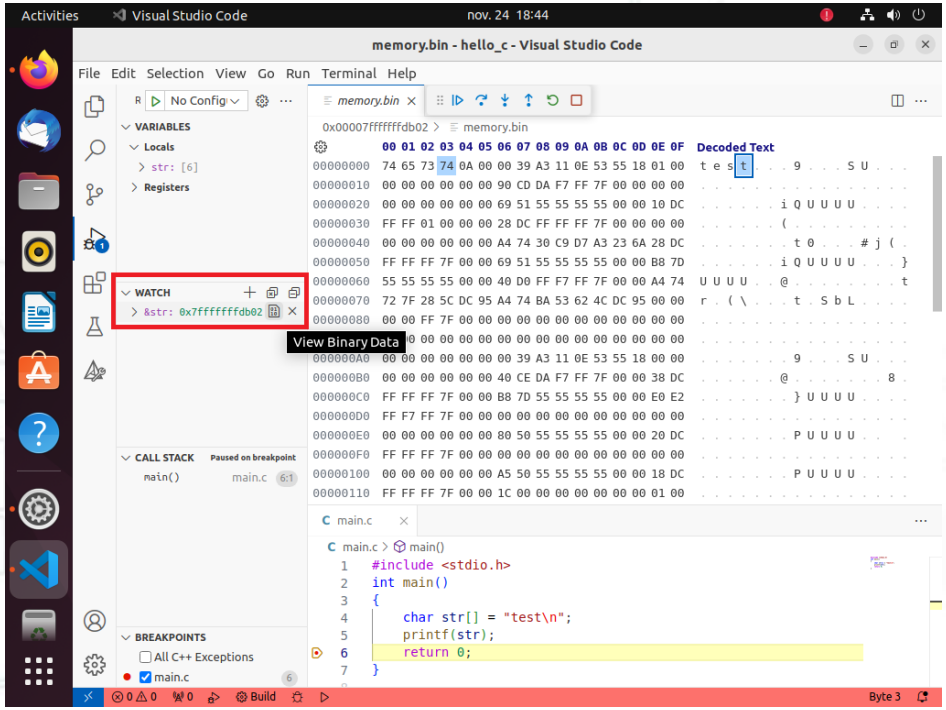
    Check the **Restricted Mode** bar if e.g. missing syntax highlighting:



Visual Studio Code

- 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

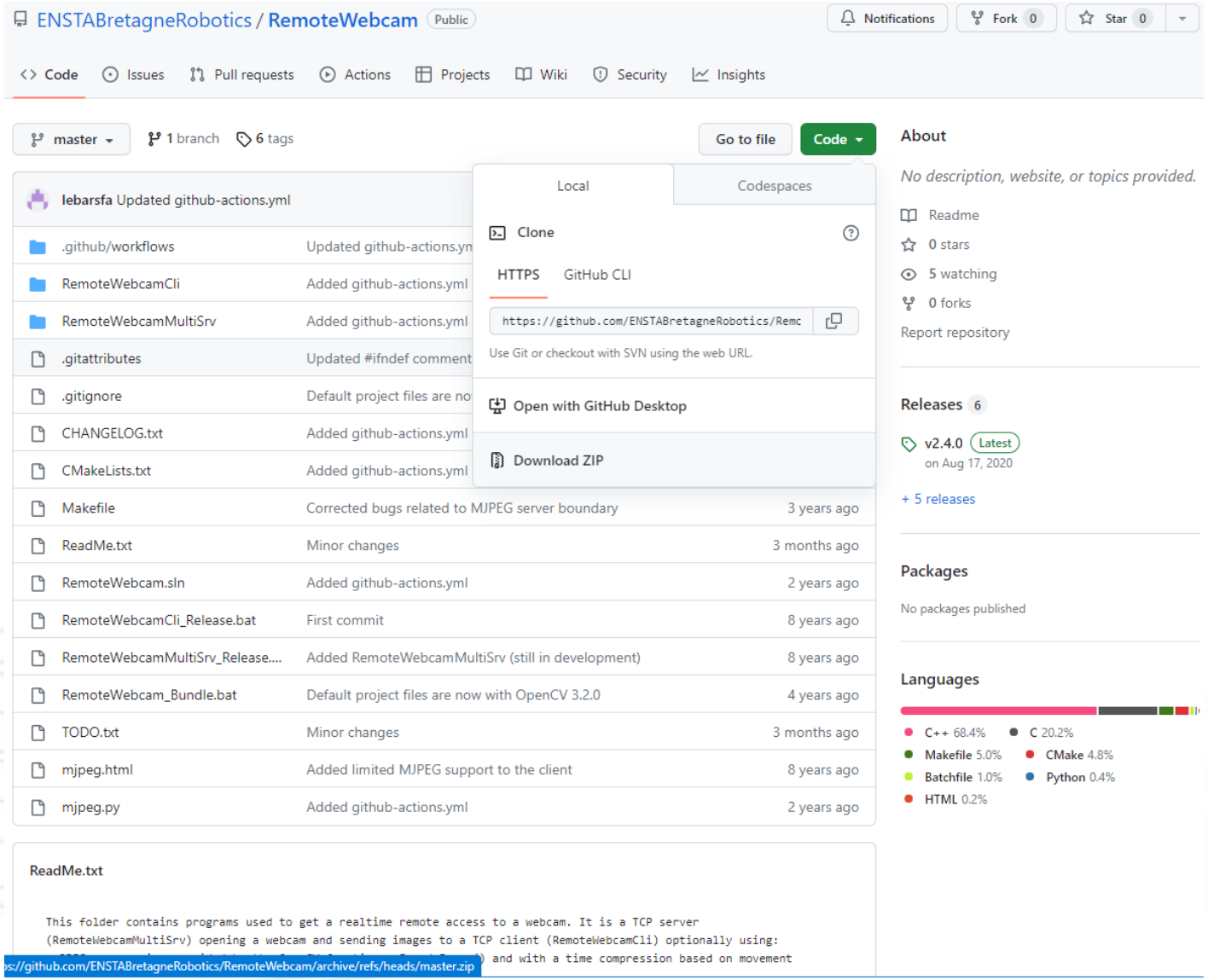
    Vertical selection: **SHIFT+ALT+click** and hold the left mouse button at the start of the text you want to select, then drag the mouse down to the end of the text

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
    - Eventuellement utiliser **Code\Download ZIP** pour télécharger rapidement le code sans l'historique des versions
    - Penser à regarder les **branchs** et **forks** des projets et l'onglet **Insights\Network** pour voir ce qui est amélioré par rapport à la branche principale

Utilisation de code existant trouvé sur Internet



Utilisation de code existant trouvé sur Internet

ENSTABretagneRobotics / razor_imu_9dof Public

forked from robotictang/razor_imu_9dof

Notifications | Fork 138 | Star 90

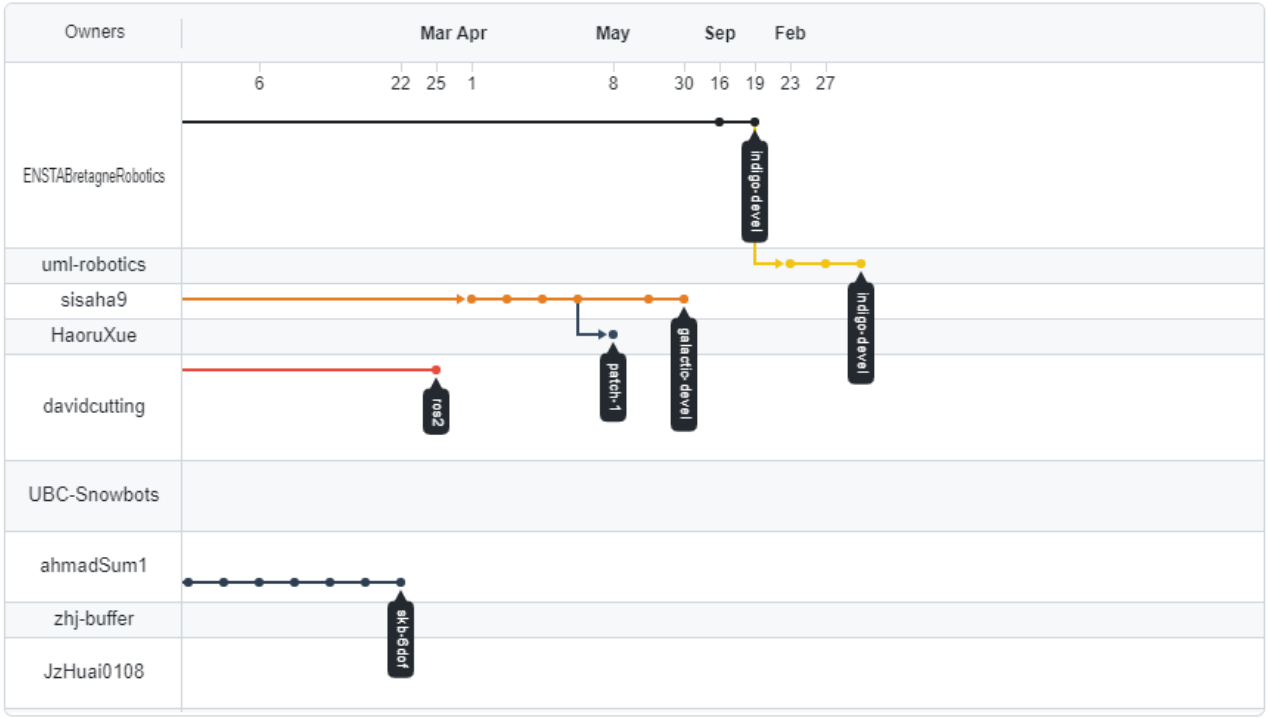
Code | Issues 18 | Pull requests 2 | Actions | Projects | Security | Insights

- Pulse
- Contributors
- Commits
- Code frequency
- Dependency graph
- Network
- Forks

Network graph

Timeline of the most recent commits to this repository and its network ordered by most recently pushed to.

The repository network shows the 100 most recently pushed forks.



Utilisation de code existant trouvé sur Internet

- 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**)…

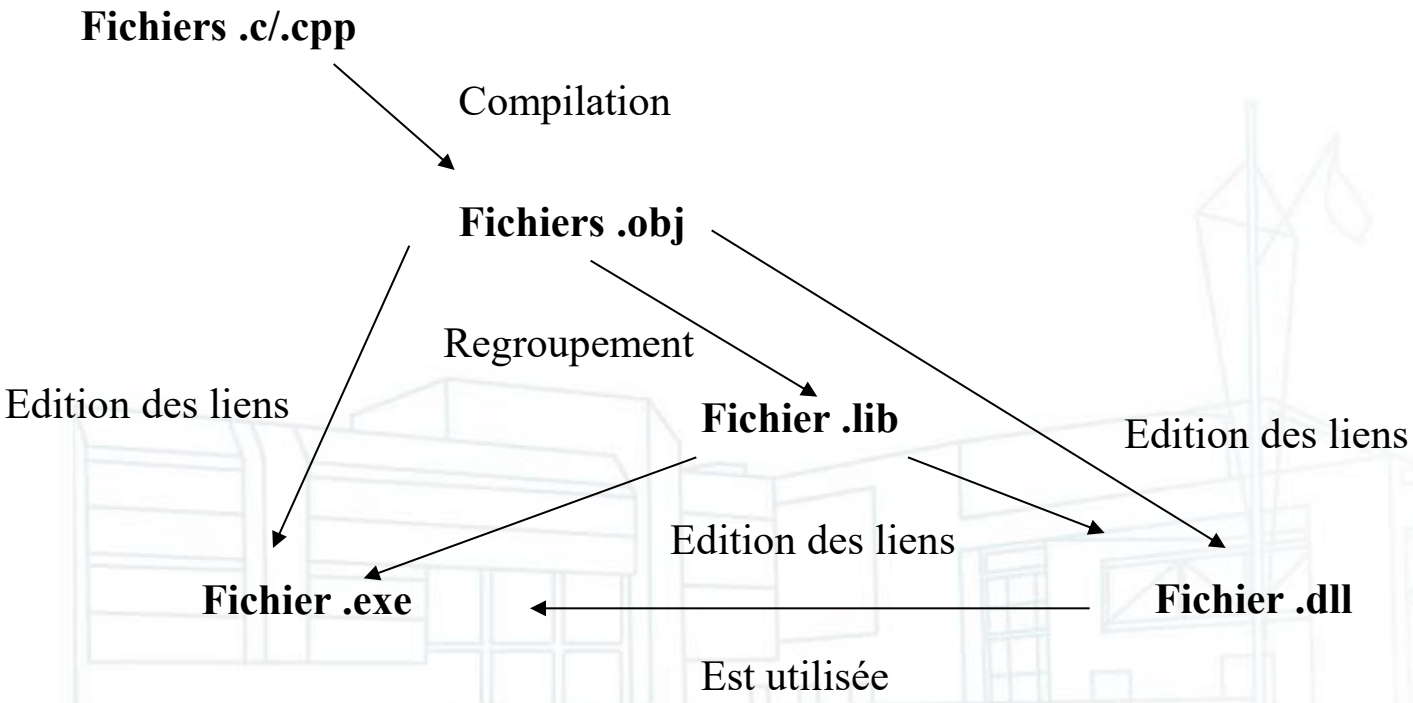
Utilisation de bibliothèques de fonctions externes

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

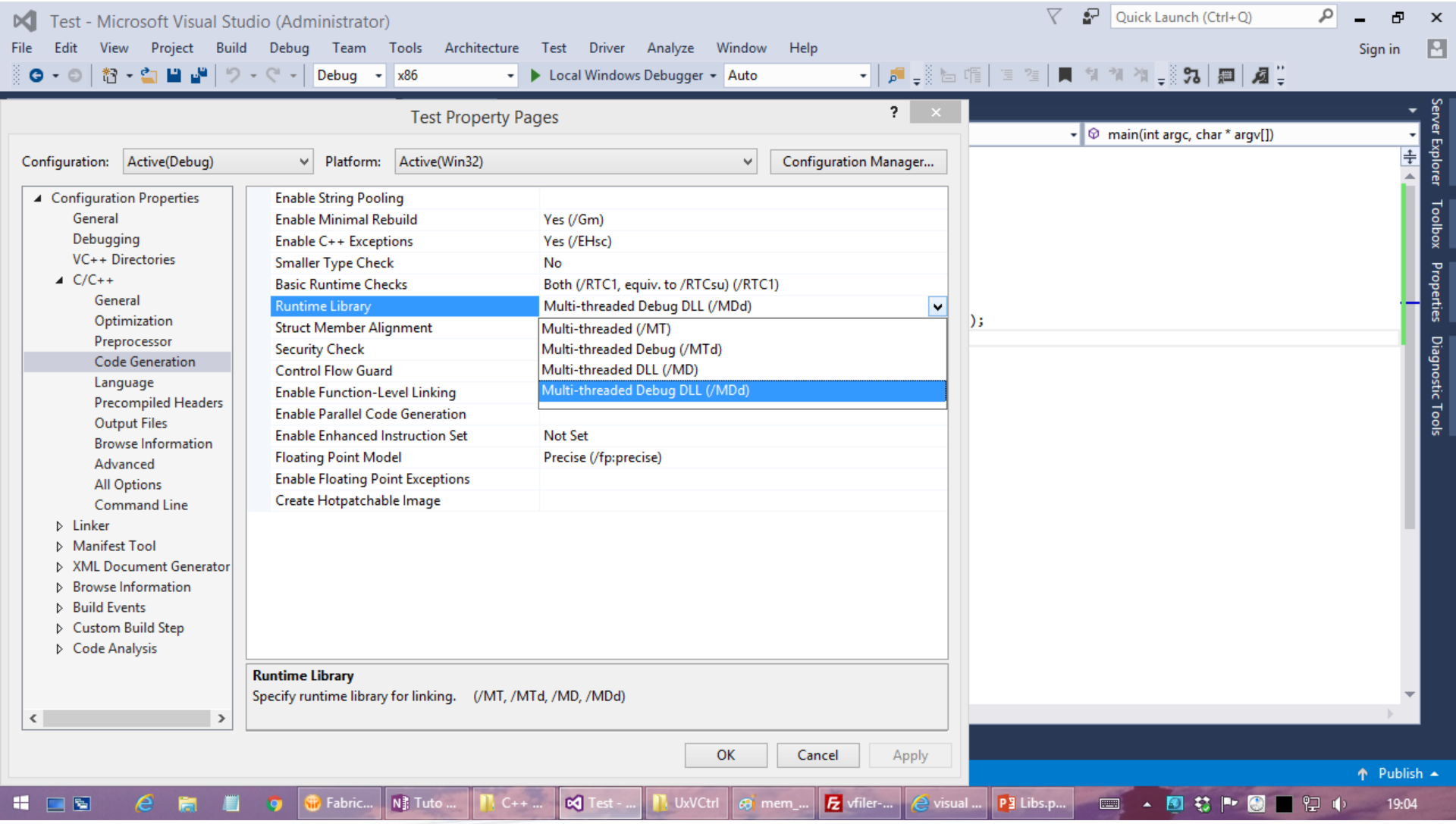
    to generate **Lib.lib**

  - 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 **Plateform** (**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



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++", categorie "**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 librairies 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