1. Lancer QtCreator

QtCreator est accessible depuis un terminal en exécutant la commande suivante[1] :

```
/opt/qtcreator-4.9.2/bin/qtcreator
```

Un projet QtCreator regroupe un ensemble de fichiers sources pour construire un (ou plusieurs) exécutables. L'ensemble des fichiers composant un projet est regroupé dans un répertoire créé spécifiquement pour le projet et qui porte le nom du projet. De façon générale, chaque exercice se fera dans un projet différent.

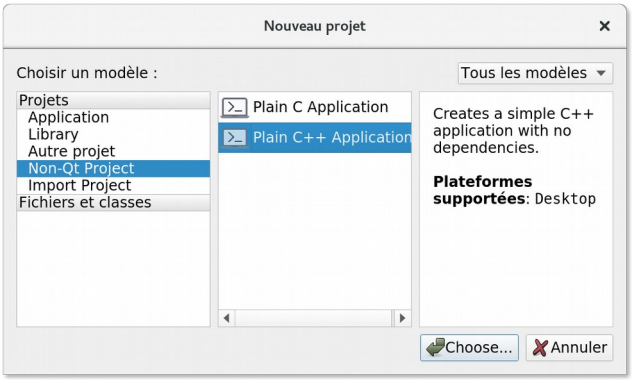
Comme tout environnement de développement, QtCreator propose de faire la construction[2] (*build*) d'un projet dans un répertoire autre que le répertoire contenant des sources, pour ne pas « polluer » le répertoire des sources avec des fichiers générés. Il n'y a aucun intérêt à sauvegarder ce répertoire de construction, puisqu'il peut être reconstruit automatiquement. Par contre, vous devez sauvegarder votre répertoire de projet (clé usb, email, outils de stockage en ligne, `scp`, `rsync`, etc.). Par défaut, QtCreator créera un répertoire de construction au même niveau que le répertoire de projet, le nom de ce répertoire commencera par `build-nomduprojet`.

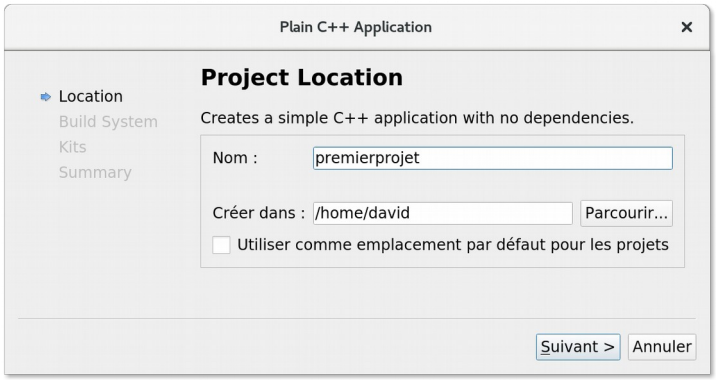
Attention Si vous choisissez de travailler directement sur une clé usb, utilisez la clé usb uniquement pour le répertoire du projet, pas pour le répertoire de construction[3]. Lors de la première construction d'un projet, il est possible de choisir dans quel répertoire la construction doit être faite (cf. ci-dessous).

2. Créer et configurer un projet

La colonne de gauche de QtCreator liste les modes d'utilisation de l'outil. L'*accueil* permet de créer un nouveau projet.

Les manipulations de cette section devront être répétées pour chaque nouveau projet. Cliquer sur le bouton *Projets* puis sur *Nouveau projet*, et choisir le modèle de projet *Non-Qt Project - Plain C++ Application* puis cliquer sur *Choose* (capture d'écran ci-contre).





Donner un nom au projet, et choisir le répertoire dans lequel le projet sera créé (capture d'écran ci-contre). **Ne pas utiliser un nom de projet ou un répertoire de projets contenant des caractères spéciaux tels que espace : contentez-vous d'utiliser des lettres, chiffres et « _ ».** Cliquer sur *Suivant* et sélectionner comme *Build system* : ***CMake***. Cliquer sur *Suivant*.

Sur la page suivante (*Kits*), différents répertoires de construction (correspondant à des configurations différentes de construction) peuvent être choisis. Dans le cadre des TP que nous allons faire, nous

1 Attention : Deux versions de QtCreator sont installées sur les postes du département informatique. Celle qui est dans le menu Ubuntu est celle qui est fournie avec Ubuntu 16.04 : il s'agit d'une version plus ancienne, moins agréable à utiliser.

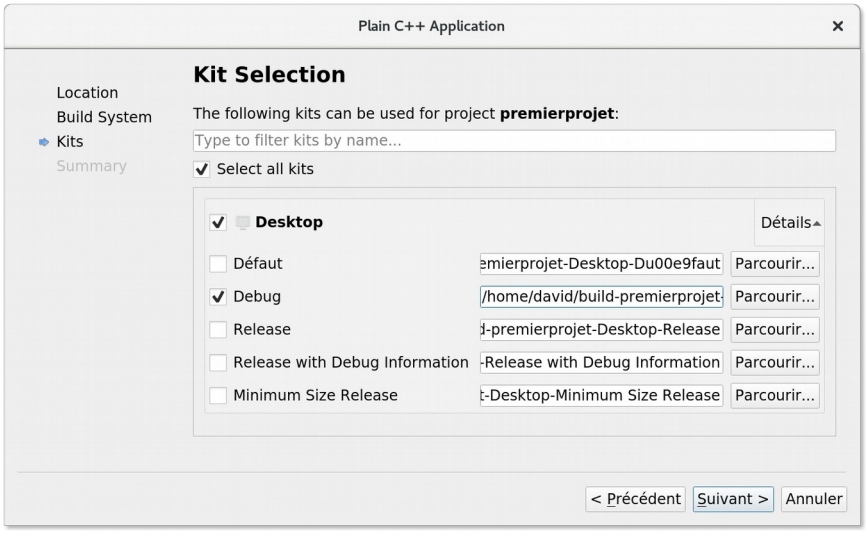
2 C'est-à-dire la génération des fichiers compilés « objet » et exécutables.

3 Les clés usb ne sont pas conçues pour des écritures répétées.

n'allons utiliser qu'une seule configuration de construction : *Debug*. Cliquer sur *Détails*, et faire en sorte que seule la configuration *Debug* soit sélectionnée (cases à cocher à gauche, cf. capture d'écran ci-contre).

Si vous utilisez une **clé usb**, c'est ici que vous devez changer le répertoire de construction de la configuration *Debug*, et choisir par exemple `/tmp/build-nomduprojet-etc`.

Cliquer sur *Suivant*, et sur la dernière page (*Project Management*) cliquer sur *Terminer*.



Le projet est créé, et contient un fichier `main.cpp` contenant un simple `main`, et un fichier `CmakeLists.txt`. L'EDI passe en mode *Éditer* (Barre d'icônes à gauche). Par défaut, le fichier `CMakeLists.txt` généré automatiquement par QtCreator ne lance pas la compilation en mode C++ 14 (mais en C++ 03), il faut donc le modifier pour cela selon les indications vues en cours : Double-cliquer sur le fichier `CMakeLists.txt` dans la liste des fichiers du projet (colonne de gauche) et le modifier de la façon suivante : (en gras les lignes à rajouter)

```
project(premierprojet)
cmake_minimum_required(VERSION 3.1.0)
set(CMAKE_CXX_STANDARD 14)
set(CMAKE_CXX_STANDARD_REQUIRED on)
set(CMAKE_CXX_EXTENSIONS off)
if(CMAKE_CXX_COMPILER_ID MATCHES "GNU")
  add_compile_options(-Wall -Wpedantic)
endif()
add_executable(${PROJECT_NAME} "main.cpp")
```

Sauvegarder les modifications par *Fichier / Save CMakeLists.txt* Ctrl-S.

3. Compiler un projet

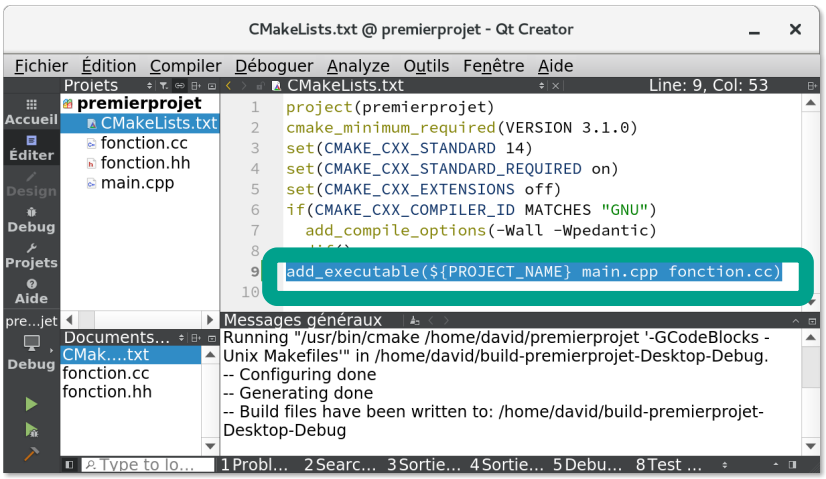
Cliquer dans le menu *Compiler* sur *Compiler le projet nomduprojet* (ou Ctrl-B[1]). Si des erreurs de compilation sont rencontrées, elles sont affichées dans la zone du bas (onglet *Problèmes*). Un double-clic sur l'erreur positionne l'éditeur sur la ligne en question. F6 passe à l'erreur suivante et Shift-F6 à l'erreur précédente. La zone du bas *sortie de compilation* contient la sortie exacte du compilateur / éditeur de liens.

4. Ajouter un fichier à un projet

Pour rajouter un nouveau fichier à un projet, cliquer dans *Fichier* sur *Nouveau fichier ou projet.* Choisir comme modèle *Fichiers et classes / C++*, puis *C++ Source File* pour créer un fichier contenant des définitions de classes ou fonctions (extension `.cc`) ou *C++ Header File* pour créer un fichier d'entêtes (extension `.hh`), puis cliquer sur le bouton *Choose*. N'utilisez pas *C++ Class*, on est là pour apprendre. Saisir le nom du fichier **avec l'extension .cc ou .hh**, puis sur *Suivant*. Cliquer sur *Terminer*. Dans le cas d'un fichier d'entêtes, QtCreator fournit une garde (`#ifndef ... #define ... #endif`) que vous pouvez conserver ou **remplacer** par une garde comme vu en cours (`#pragma once`).

1 Pour s'en souvenir : *B* comme build.

Une fois que ceci est fait, le fichier a effectivement été créé et vous pouvez l'éditer, **mais** il n'apparaît pas encore comme étant rattaché au projet : QtCreator rattache au projet les fichiers sources présents dans le fichier `CMakeLists.txt`. Les fichiers d'entête n'ont pas besoin d'être rajoutés dans le `CMakeLists.txt`, ils seront ajoutés au projet automatiquement quand le fichier source associé (fichier `.cc` ayant le même nom que le fichier `.hh`). Par exemple, après avoir ajouté deux fichiers `fonction.hh` et `fonction.cc`, le fichier `CMakeLists.txt` doit être édité comme ci-contre, et une fois que ceci est fait (et le fichier `CMakeLists.txt` sauvegardé), `fonction.hh` et `fonction.cc` apparaissent dans la liste des fichiers du projet (colonne de gauche).



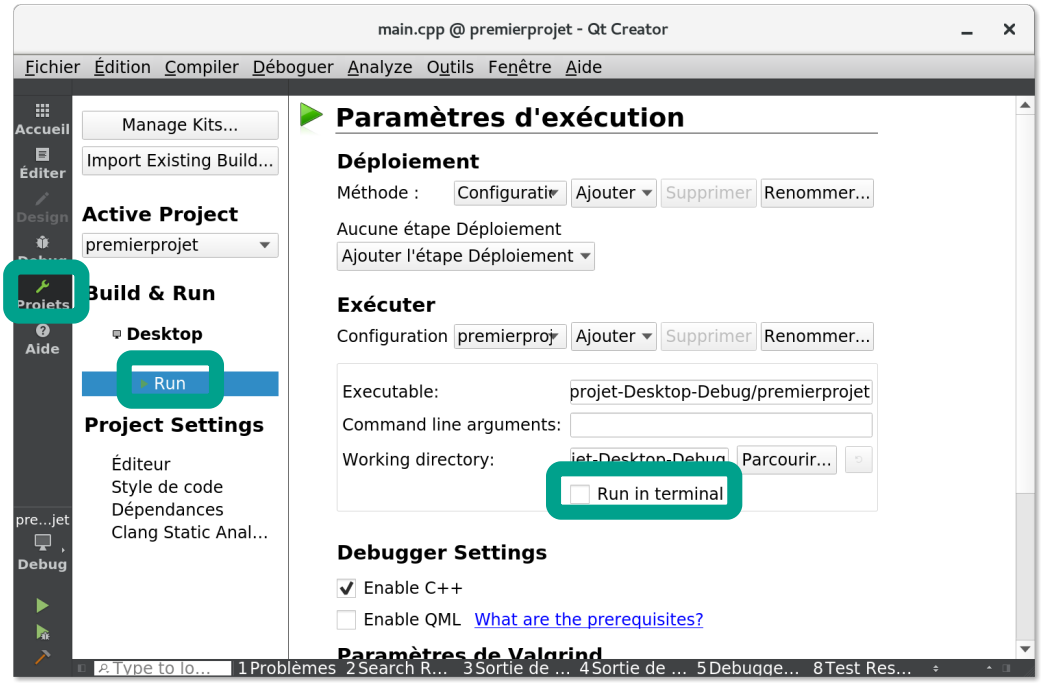
Notez qu'un projet peut contenir plusieurs exécutables[1] : pour cela, il faut éditer le fichier `CMakeLists.txt` et ajouter autant de lignes `add_executable` que d'exécutables, et faire suivre le nom de chaque exécutable des noms des fichiers sources nécessaires à la création de l'exécutable :

```
add_executable(executable1 mainexe1.cpp autresource.cc)
add_executable(executable2 mainexe2.cpp autresource.cc)
```

5. Exécuter

Pour lancer l'exécutable, cliquer dans *Compiler* sur *Exécuter* Ctrl-R[2]. La sortie d'exécution s'affiche dans la zone du bas *Sortie de l'application*.

Attention Si l'exécutable a besoin de saisies au clavier (`std::cin >> ...`), il restera définitivement en attente car la zone *Sortie de l'application...* n'est qu'une sortie, et pas une entrée. L'exécution peut toutefois être interrompue par le bouton stop (carré rouge) dans la zone de sortie. Pour exécuter une application faisant des saisies au clavier, il faut l'exécuter dans une fenêtre externe : Dans le mode *Projets*, catégorie *Run*, cocher *Run in terminal*. La configuration de l'exécution permet aussi de choisir les paramètres passés en ligne de commande (`argv` du `main`) à l'exécutable (*Command line arguments*). Plusieurs configurations d'exécution peuvent être créées à partir de cet écran de configuration (bouton *Ajouter*).



1 Ce ne sera pas le cas dans ce premier TP.

2 *R* comme Run.

6. Déboguer

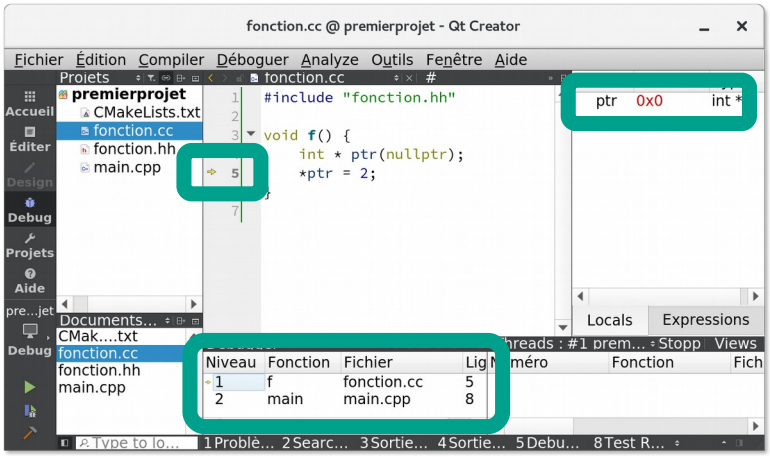
Attention Une application exécutée dans un terminal ne peut être déboguée dans QtCreator[1].

Rajouter dans un fichier source cette fonction (manifestement incorrecte) et ajouter un appel vers cette fonction depuis le `main`.

```
void f() {
    int * ptr(nullptr);
    *ptr = 2;
}
```

Quand l'application est exécutée, elle se termine évidemment par une erreur : `Le programme s'est terminé subitement`. Si le code source est long, il est difficile de savoir quelle est la cause d'une erreur d'exécution. Le débogage permet de mettre au point une application et nous aide à trouver et corriger les erreurs d'exécution. Pour pouvoir déboguer une application il faut que la construction soit faite en mode *Debug* (ce qui est le cas si vous avez suivi les consignes de la section 2).

Pour commencer une session de débogage, plutôt que de lancer une exécution Ctrl-R il faut cliquer sur *Déboguer* / *Commencer le débogage* / *Start debugging of startup project* F5. L'exécution est alors lancée dans le débogueur.



L'exécutable se termine toujours par une erreur (évidemment), on apprend maintenant que c'est un signal `SIGSEGV` (erreur de segmentation). La zone du bas affiche la pile d'appels, ici on voit que c'est le `main` qui a appelé `f` qui a causé l'erreur. La ligne courante est précisée par une flèche sur le source, et les valeurs des variables locales sont affichées (à droite). Passer la souris sur un nom de variable fait apparaître une bulle d'aide avec sa valeur.

Le mode *Debug* permet aussi de faire une exécution pas à pas : F10 pour passer à la ligne suivante, F11 pour passer à la ligne suivante en allant dans la fonction appelée. On peut aussi placer des points d'arrêt (*breakpoints*) en cliquant sur la zone à gauche des numéros de ligne : au cours de l'exécution, quand une ligne avec un point d'arrêt est rencontrée, l'exécution s'arrête (avant l'exécution de la ligne), ce qui peut nous permettre de vérifier facilement si les valeurs des variables sont conformes à ce qui était attendu à ce moment-là, puis de continuer l'exécution F5 jusqu'au prochain point d'arrêt ou continuer en mode pas à pas.

1 Ce qui n'est pas très gênant : en phase de débogage, il n'y a aucune raison de faire des saisies au clavier, or, c'est la seule raison d'exécuter l'application dans un terminal.