Qt Quickstart

Mathias M. Vege

October 10, 2018

Abstract

A short guide to getting started with Qt Creator.

Contents

1 Installing Qt

In order to get Qt Creator up and running as an IDE, there are a few requirements. The following is needed,

- **Qt**. The Qt core package containing the core functionality and libraries used by Qt Creator. Contains the kits needed to compile and build projects, as well as libraries for building e.g. Android and iPhone apps.
- **Qt Creator**. The IDE used for programming.
- **A C++compiler**. Needed in order to compile C++files in Qt Creator. The specific compiler needed depend on the OS.

NOTE: it is possible to update your Qt platform after installing it through the Qt Maintenance tool in case you are missing any of the Qt utilities, such as a proper Qt version.

1.1 Mac

1.1.1 Prerequisite

- brew. See `https://brew.sh/` for a guide to installing it.
- Xcode. Download from app-store. Through Xcode you should get a compiler for C++.

1.1.2 Installing Qt

Go to `https://www.qt.io/download`, and follow the instructions. Make sure to select a Qt version under the tab *Qt*, typically you select the newest version Qt.11.1(as of October 10, 2018). For me that is *macOS*. Qt Creator should be automatically selected under *Tools*.

1.2 Ubuntu

1.2.1 Prerequisite

Make sure you have a proper compiler. To install the essential tools for C++, type

```
$ sudo apt install build-essential
```

in the terminal. See this page for details on how to install a `gcc` or `g++` compiler.

1.2.2 Installing Qt

There are two options for installing Qt on Ubuntu/Linux.

- Go to `https://www.qt.io/download`, and follow the instructions. Make sure to select a Qt version under the tab *Qt*, typically you select the newest version Qt.11.1(as of October 10, 2018). Qt Creator should be automatically selected under *Tools*.
- Simply type

  ```
  $ sudo apt-get install qtcreator
  $ sudo apt-get install qt5-default
  ```

  and you should have everything you need.

1.3 Windows

1.3.1 Prerequisite

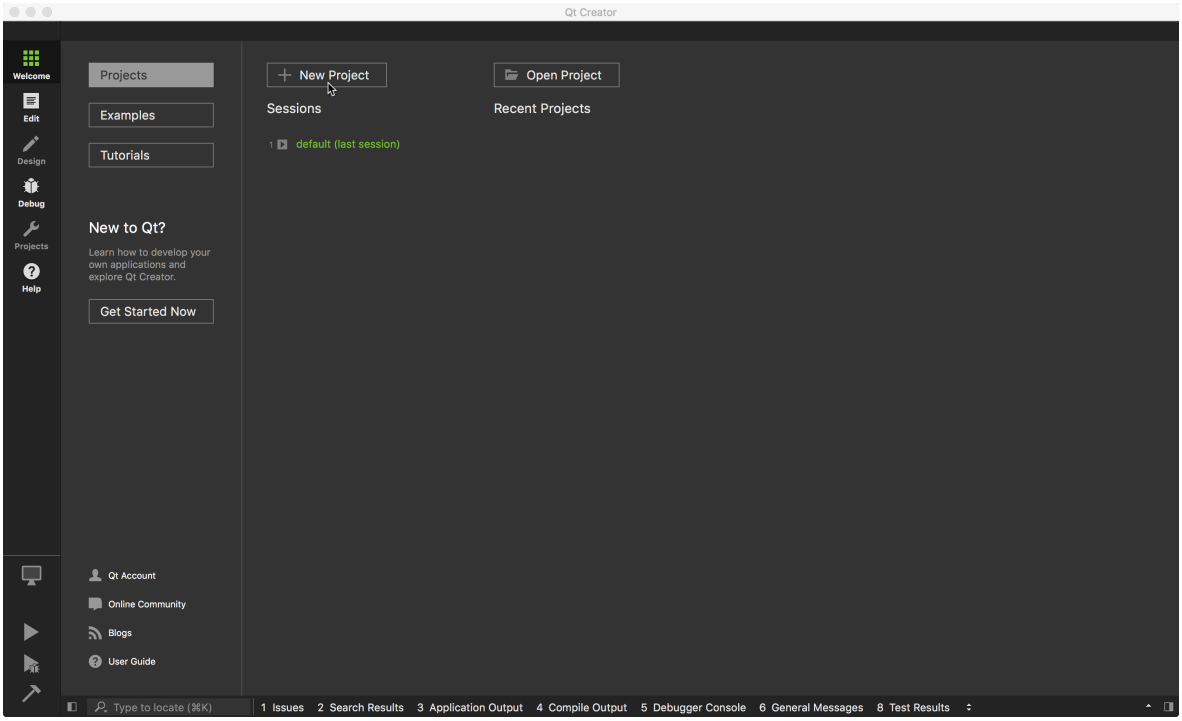
Make sure you got a C++compiler. For a quick and easy install, look up the MinGW 64 bit compiler.

1.3.2 Installing Qt

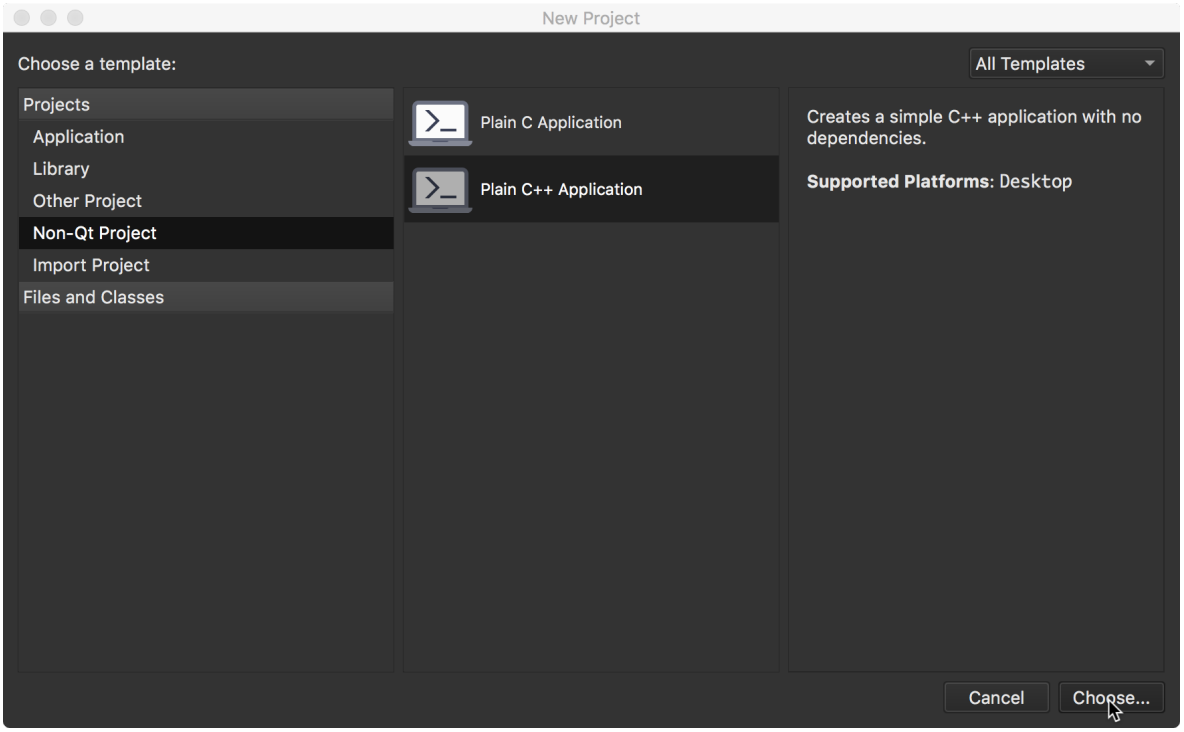
Go to `https://www.qt.io/download`, and follow the instructions. Make sure to select a Qt version under the tab *Qt*, typically you select the newest version Qt.11.1(as of October 10, 2018). For me that is *macOS*, but that may differ on Windows. Qt Creator should be automatically selected under *Tools*.

2 Creating your first project

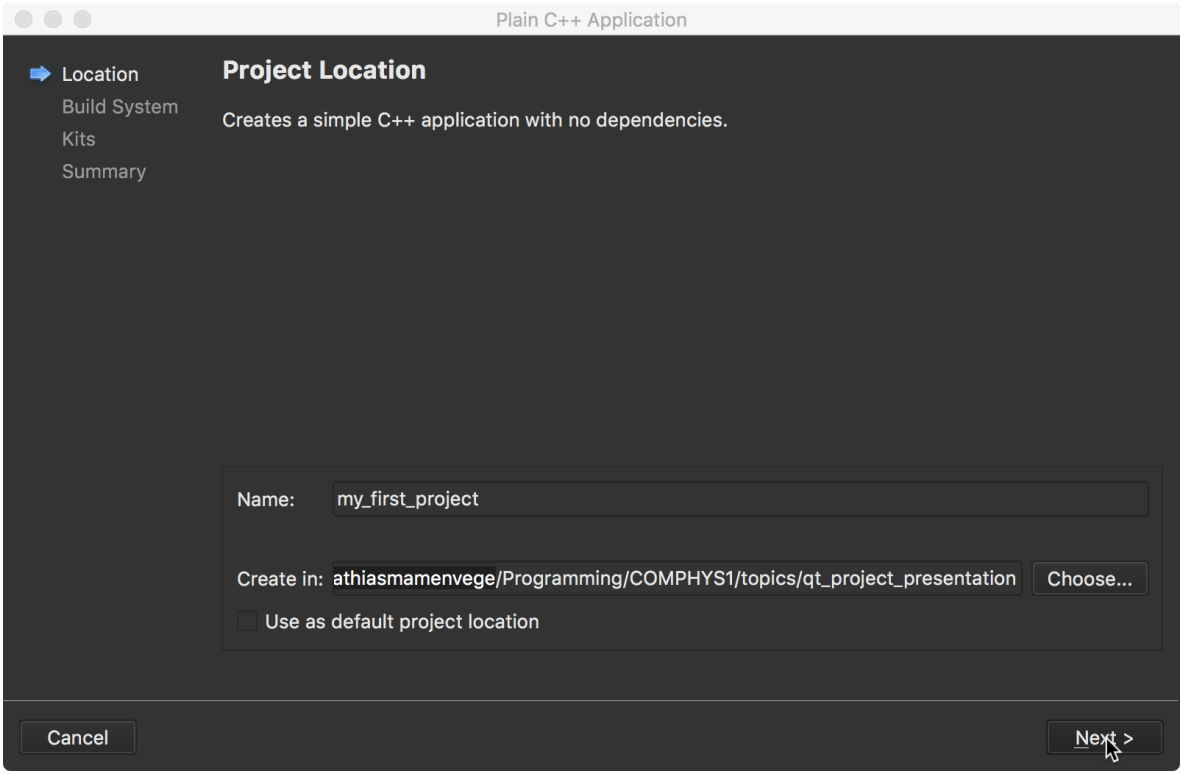
When starting up Qt Creator for the first time, this should be your starting screen. Start by clicking *New Project*.



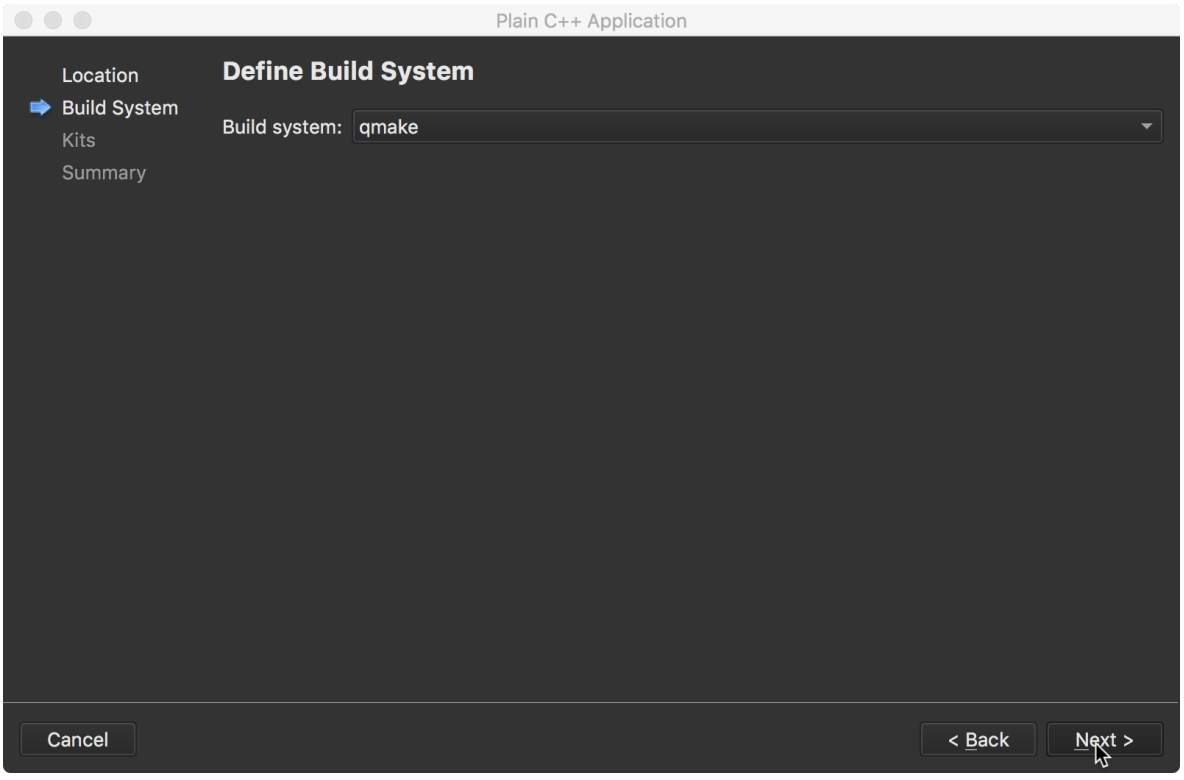
You now choose a template. Select *Non-Qt Project* in the sidebar to the left, then select *Plain C++ Application* and click next.



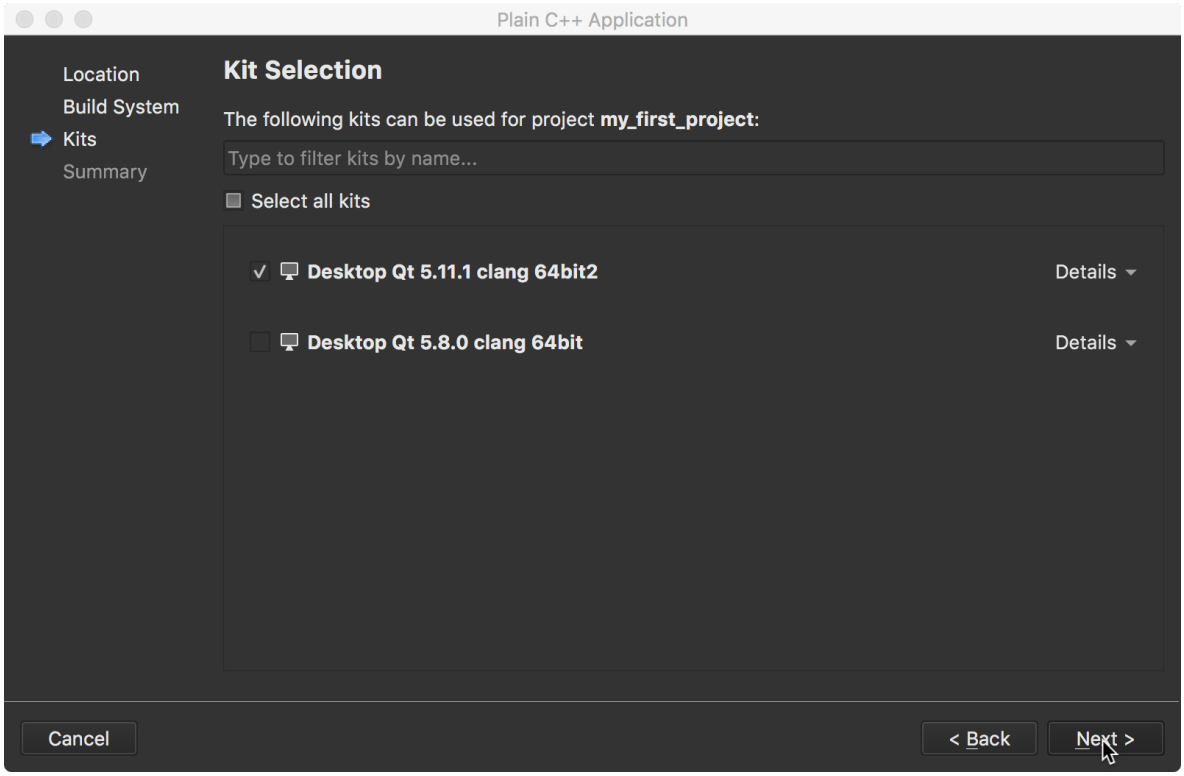
Location. Then you give a name to your project and select its location.



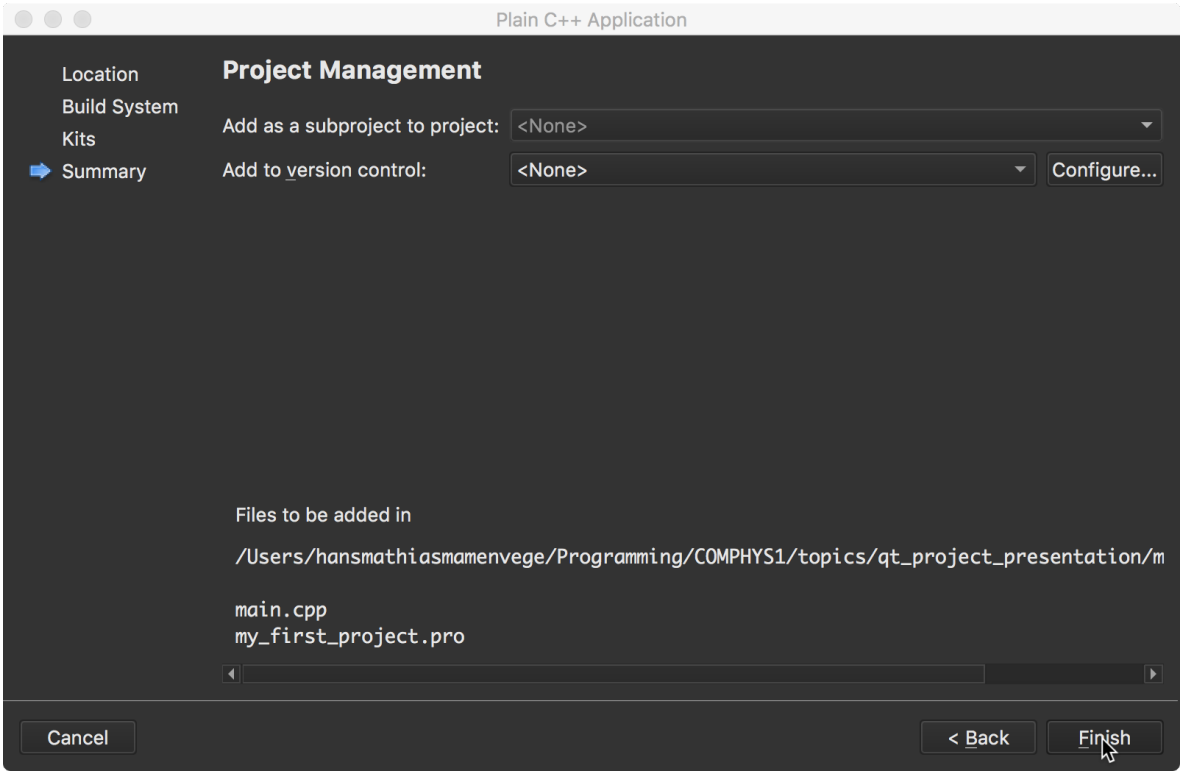
Build System. Default is *qmake*, and is what we recommend.



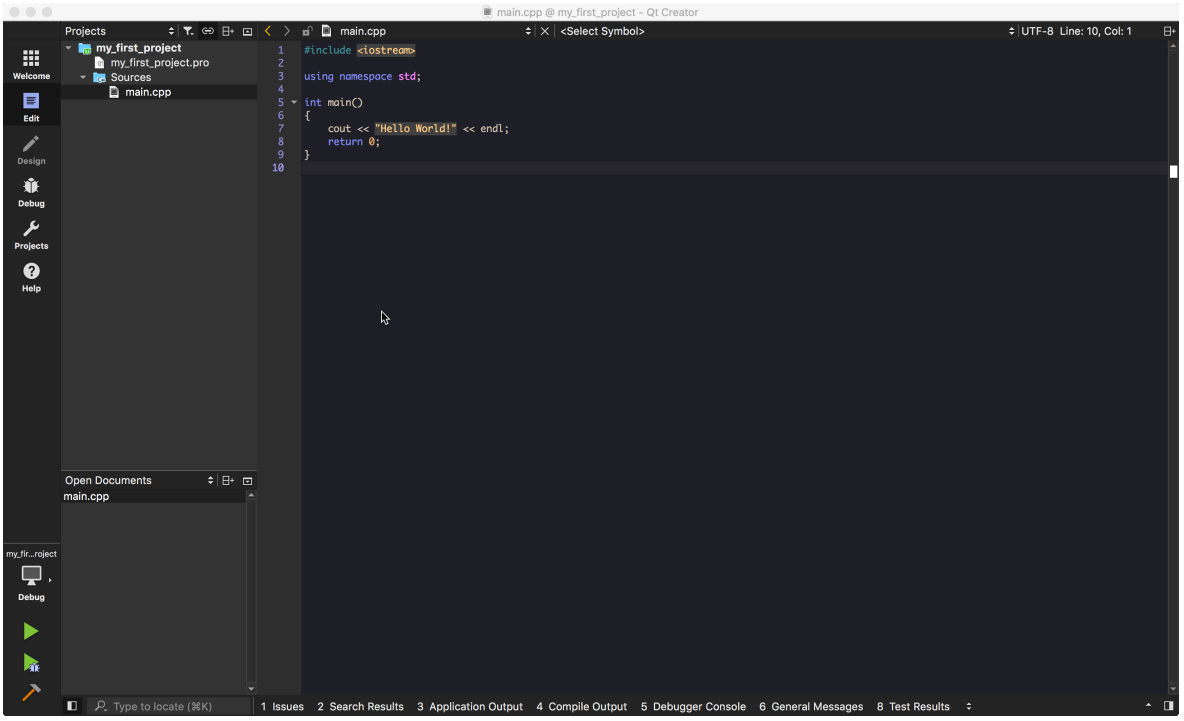
Kit Selection. Select the newest kit available. May appear as *Desktop*(on Ubuntu), *Desktop Qt 5.11.1 clang 64bit*(on Mac) or similar. **Note:** if no kits are available, look back into the *Installing Qt* section1.



Summary. Everything should now be set up and ready to go, and you can begin programming.



If you are missing a compiler, you might get a lot of errors under the *Issues* panel. Go back to *Installing Qt* section1 for how to install a compiler for your operating system.



3 Project settings

3.1 Build settings

A plethora of important settings can be accessed from the *Build Settings* panel found in the side bar of Qt Creator.

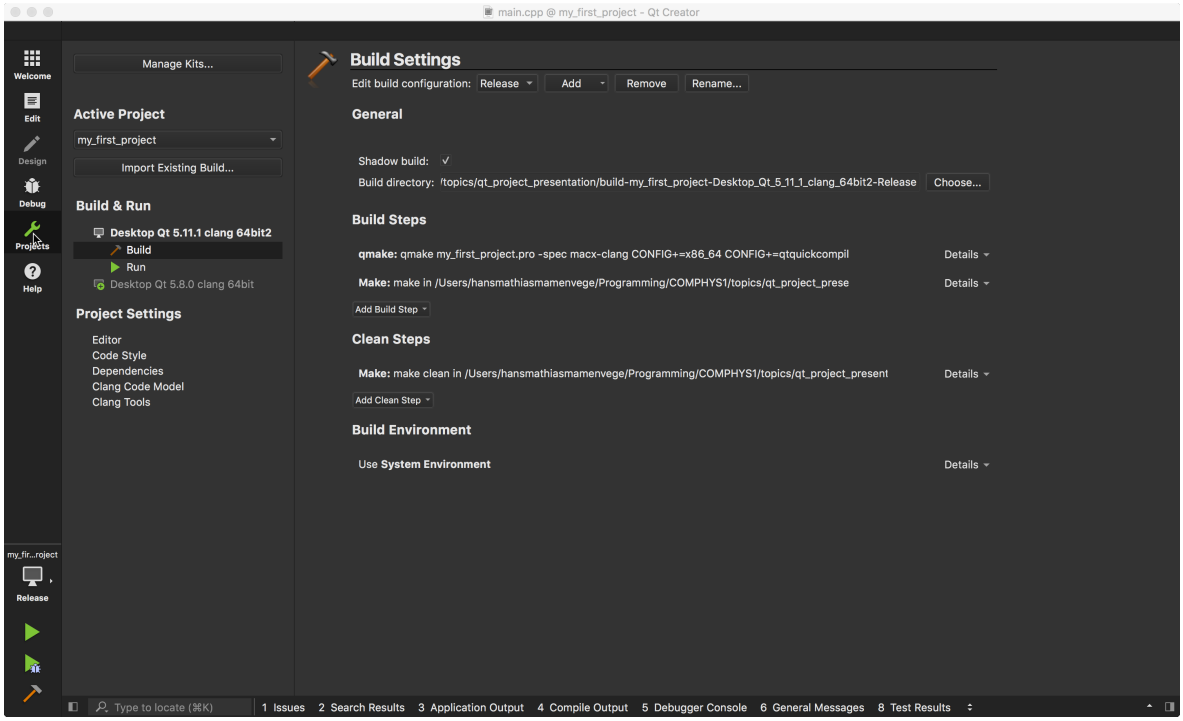


Figure 1: Build settings panel.

3.2 Changing build folder

By default, Qt Creator will create a build folder located one folder above the project folder.

```
\home\<user>\programming\my_project
\home\<user>\programming\build-my_first_project-
    Desktop_Qt_5_11_1_clang_64bit2-Release
\home\<user>\programming\build-my_first_project-
    Desktop_Qt_5_11_1_clang_64bit2-Profiling
\home\<user>\programming\build-my_first_project-
    Desktop_Qt_5_11_1_clang_64bit2-Debug
```

It will typically make on folder for each of the `Release`, `Debug` and `Profiling` builds. If you want to change this go to the settings folder as listed in the figure above 1. This can be changed under the *General* section, by selecting a *build directory* depending on what build you are currently set to,

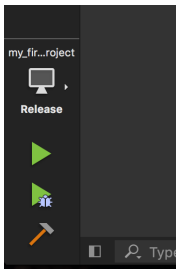


Figure 2: Build selecting found in the lower left corner.

3.3 Run settings

If we now click the *Run Settings* as seen on the *Build Settings* panel, we get the following screen up,

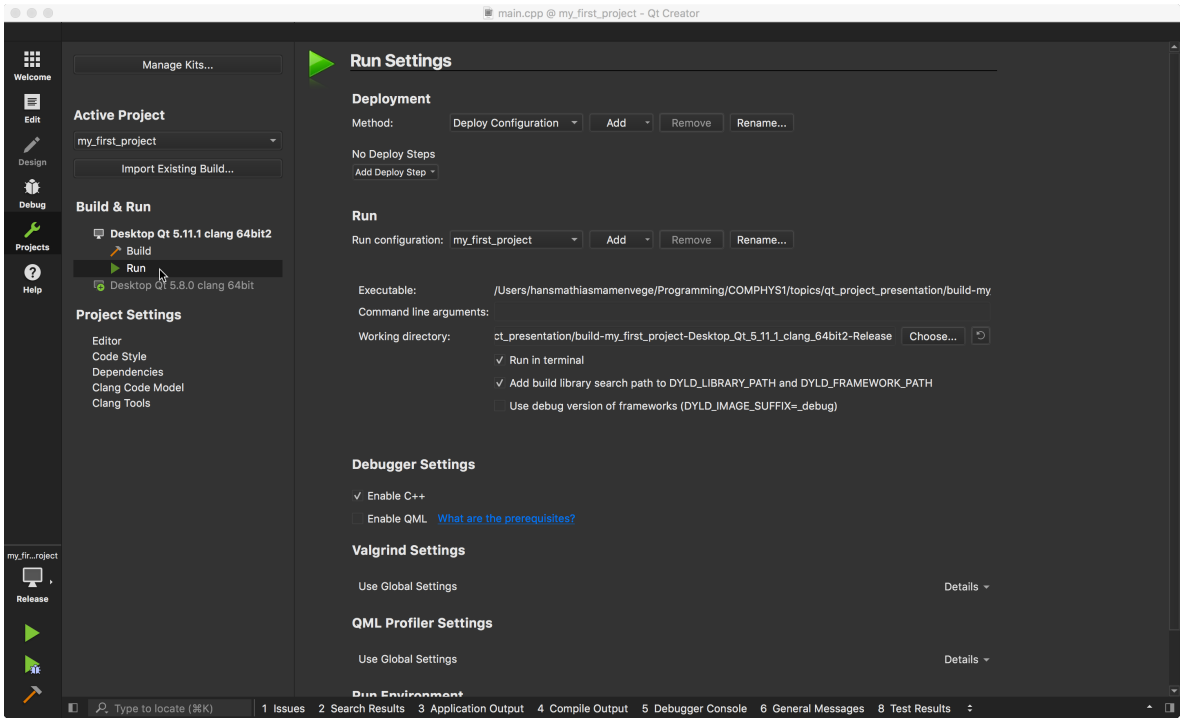


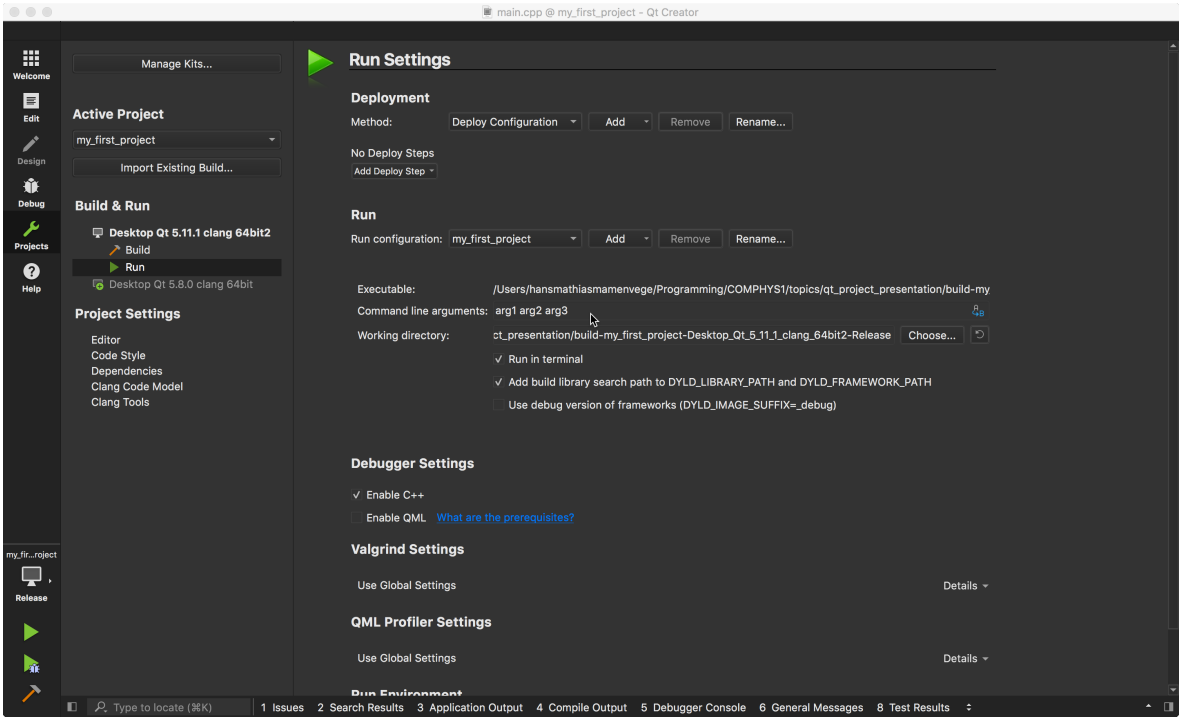
Figure 3: Run settings panel found at under the build settings panel1.

3.4 Command line arguments

If you want to add command line arguments to your program, there are two ways of doing it. One is to run the program from the terminal in a regular fashion,

```
$ ./my_first_project arg1 arg2 arg3
```

Or, you can add the arguments under *Run* section in the *Command line arguments*,



You can also turn of the *Run in terminal* mode by unchecking the box slightly below the *Command line arguments* input.

```
$ ./my_first_project arg1 arg2 arg3
```

4 Expanding your project

4.1 The `.pro`-file

The `.pro` contains the settings for QMake, which is the compiling tool used by Qt(instead something like cmake). Your `.pro` file will look like this at the beginning of your project,

```
TEMPLATE = app
CONFIG += console c++11
CONFIG -= app_bundle
CONFIG -= qt

SOURCES += \
        main.cpp
```

From the `.pro` file you can specify what C++version you are going to use(C++11 should be the default), what optimizations to utilize and what external libraries we are to use.

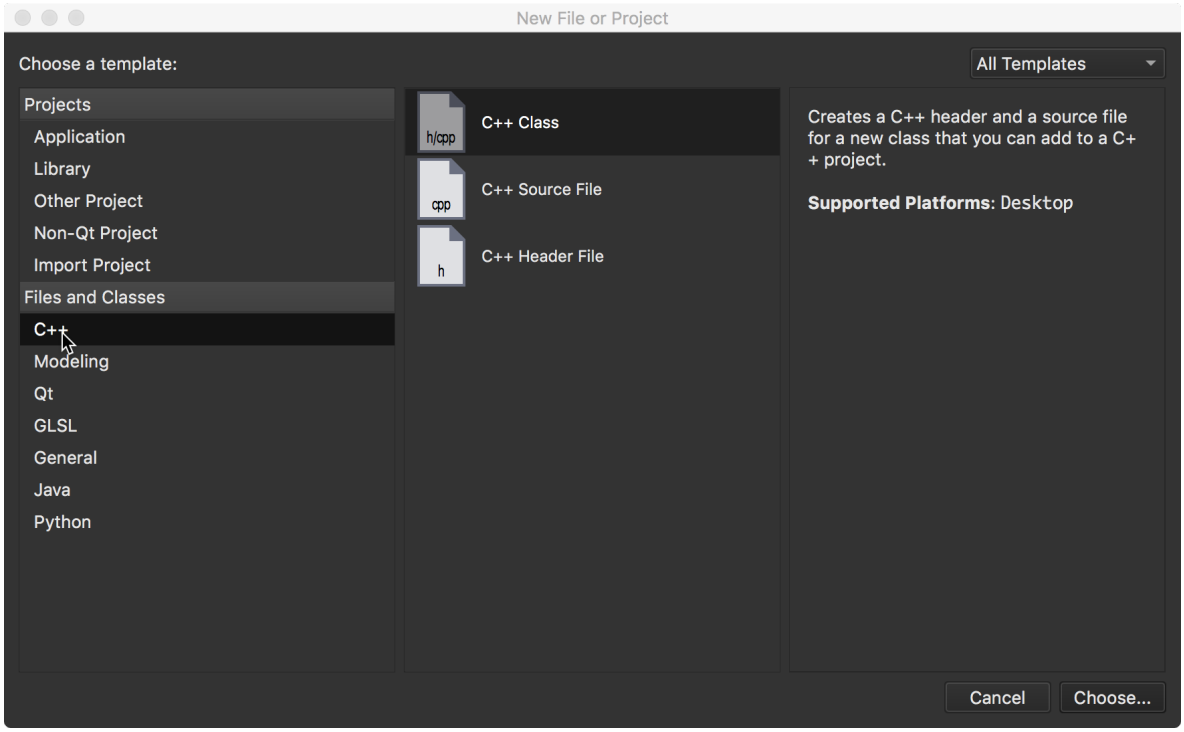
4.2 Adding files and classes

When adding new files to your project, either

- press `CMD+N`.
- click *New File* under *File*

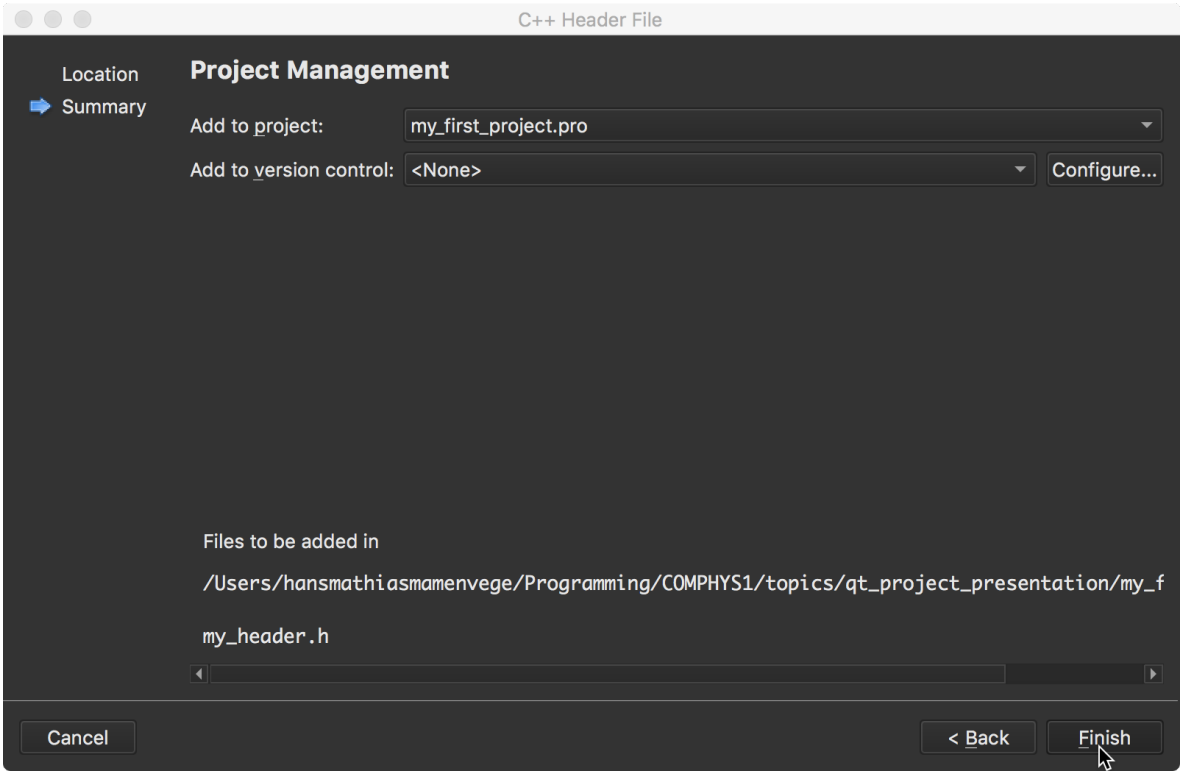
- right click at the folder you wish to create a new file in the *Projects* tab and select *Add New....*

If you do either of the of the first two you should get up a panel similar to the one you got when starting a new folder. Instead of looking under the *Projects* templates, go beneath that to *Files and Classes* and select *C++*. Then select the type of file you are going to choose,



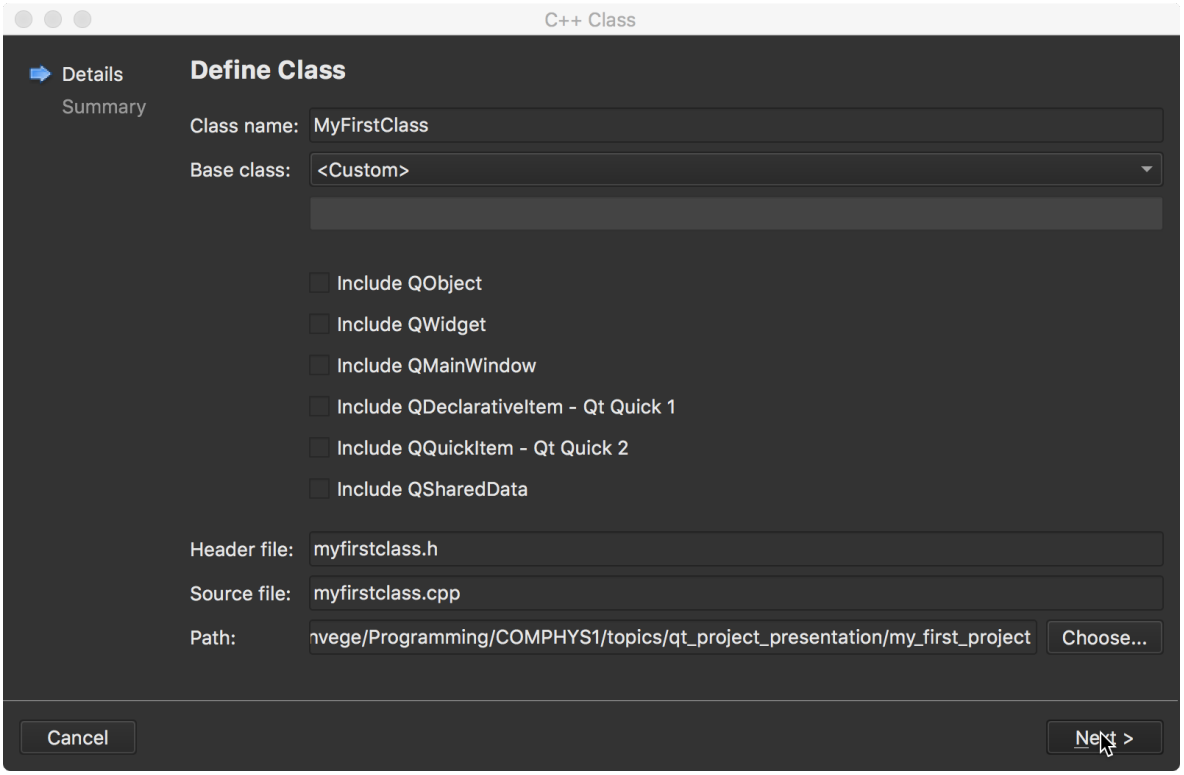
4.2.1 New header files

If you are going to create a header, select that and give it a name and location in the next panel. The summary screen will then look something like this,



4.2.2 New Classes

If you are setting up a class, select that and give it a name.



The files you are including will then be added to the `.pro` file automatically, and will now look something like,

```
TEMPLATE = app
CONFIG += console c++11
CONFIG -= app_bundle
CONFIG -= qt

SOURCES += \
        main.cpp \
    firstclass.cpp

HEADERS += \
    firstclass.h \
    firstheader.h
```

4.3 Adding existing files

Say we now want to add a files `"lib.h"` and its functions `"lib.cpp"` to our project. After taking note of their location, you simply add

```
<path_to_lib.h>/lib.h
```

under `HEADERS`, and

```
<path_to_lib.cpp>/lib.cpp
```

under `SOURCES` in the `.pro` file. Then you include the file by writing `#include "lib.h"` in the file you need it in. [1]

[1] Go here for an explanation for why we use `"lib.h"` and not `<lib.h>`.

5 Guides to external libraries for Qt

After introducing the `.pro` file, we can look at how libraries is including. In the `.pro` this is done with primarily two commands:

```
LIBS += -L/<path_to_library> -l<library_name>
INLCUDEPATH += <path_to_search_for_headers_in>
```

5.1 Armadillo

5.1.1 Installation

- **Windows**. Look at this guide for how to install Armadillo on Windows.
- **Mac**. Install Armadillo by simply typing,

  ```
  $ brew install armadillo
  ```

  in the terminal.
- **Ubuntu**. To install Armadillo on Ubuntu, use

  ```
  $ sudo apt-get install libarmadillo-dev
  ```

5.1.2 Including Armadillo in Qt Creator

- **Mac**. To include Armadillo on Mac in Qt Creator, you have to add following to your `.pro` file. This is assuming Brew linked Armadillo correctly and installed all dependencies.

  ```
  INCLUDEPATH += /usr/local/include
  LIBS += -L/usr/local/lib -llapack -lblas -larmadillo
  ```

  The path was used here is the path provided by Brew when typing

  ```
  brew info armadillo
  ```

  After adding the path go to *Build* at the top, and select *Run qmake*.
- **Ubuntu**. After installing the required libraries, simply include

  ```
  LIBS += -llapack -lblas -larmadillo
  ```

  in your `.pro` file.
- **Windows**. See post on Piazza or go to the bottom of the same document listed in under the installation of Armadillo.

To include Armadillo in your C++program, go to your `.pro` file and add the following,
An example of a full `.pro` file on Mac would be

```
TEMPLATE = app
CONFIG += console c++11
CONFIG -= app_bundle
CONFIG -= qt

SOURCES += \
```

```
    main.cpp

INCLUDEPATH += /usr/local/include
LIBS += -L/usr/local/lib
LIBS += -llapack -lblas -larmadillo
```

5.1.3 Troubleshooting

- **Mac**. If you run into an issue where you get `Image not found` at run time, try to reinstall `arpack` and Armadillo with brew.

5.2 OpenMP

OpenMP is a method of parallelizing code.

5.3 MPI

6 Profiling your code

Two profiling methods will be presented in this section,

- *Memory profiling*. Will check if any memory is lost. I.e. you forget to de-allocate any memory.
- *Function profiling*. Will check the time spent in each function.

Methods for profiling code can be found at Qt Wiki for Profiling and Memory Checking Tools. Here, we will present Valgrind as the choice of profiler.

To view output of the Valgrind profiler(in case you do not use Qt Creator), use KCacheGrind(or QCacheGrind for Mac users) which can be found here.

6.1 Installing Valgrind

6.1.1 Mac

Install Valgrind with brew,

```
$ brew install valgrind
```

If you use MacOS High Sierra(¿ 10.13), you might get an error on the form of,

```
valgrind: mmap-FIXED(0x7fff5f400000, 8388608) failed in UME (load_unixthread1)
     with error 22 (Invalid argument).
```

The fix is to uninstall Valgrind and reinstall with the `--HEAD`.

```
$ brew uninstall valgrind
$ brew install valgrind --HEAD
```

6.1.2 Ubuntu

Simple use

```
$ sudo apt-get install valgrind
```

and you should have it.

6.2 Installing K/QCachegrind

6.2.1 Mac

```
$ brew install kcachegrind
```

6.2.2 Ubuntu

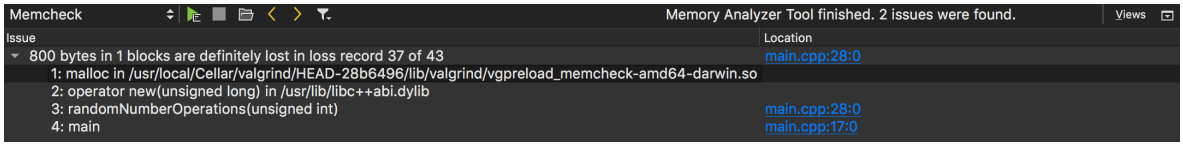
```
$ sudo apt-get install kcachegrind
```

6.3 Running Valgrind memory check(memcheck)

6.3.1 Qt Creator

In Qt Creator, first compile your program in debug mode. Then select Analyze -¿ Valgrind Memory Analyzer with GDB. If the drop-down panel is something else than Debugger click it and select memcheck. If it does not start, click Run.

The output will then on the shape of



6.3.2 Terminal

If you want to run in terminal, you might want to have your program compiled in debug mode(not required though), and type following

```
$ valgrind --leak_check=full ./my_executable
```

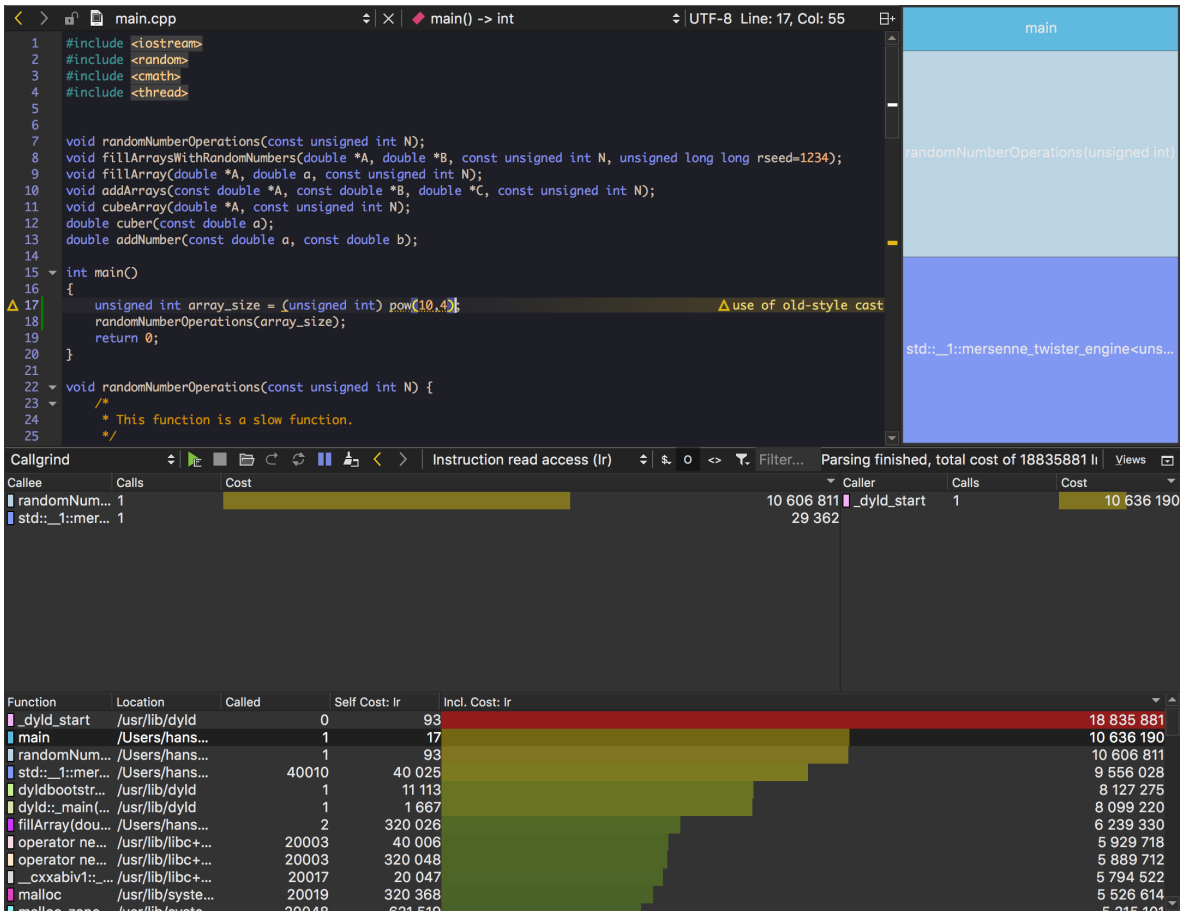
The argument `--leak_check=full` prompts Valgrind to output a detailed overview of the locations of where it detected memory leaks.

Mac users: Valgrind may display many leaks which are related to the OS, and thus can be discarded.

6.4 Profiling with Valgrind(Callgrind)

6.4.1 Qt Creator

If Qt Creator, you select Analyze → Valgrind Function Profiler. Qt Creator should now be profiling your code for you. You result should look something like this:



6.4.2 Terminal

To run a Callgrind in terminal, use

```
$ valgrind --tool=callgrind ./my_executable
```

This produces an output on the format of `callgrind.out.<pid>`. The last numbers is just the Process identifier. To view this file (depending on OS), use, *Mac*:

```
$ qcachegrind callgrind.out.<pid>
```

Ubuntu:

```
$ kcachegrind callgrind.out.<pid>
```

7 Optimizing your code

By including `-march-native` you allow for processor specific compilation.

`-O3` you will allow get vectorization enabled, together with the optimizations from `-O2`, `-O1`, `-OFast`.

With `-fopt-info` you will see what compiler optimizations is performed.