HIGHER SCHOOL OF ECONOMICS
NATIONAL RESEARCH UNIVERSITY

Introduction to Programming in C++
Setting Working Environment

Sergey Shershakov

List of software

- Web-browser
- Toolchain
  - for compiling C++ programs
- IDE
  - for comfortable working with code, debugging and building applications
- CMake building system
  - automation of app building
- Advanced text editor
  - working with individual text files
- Telegram
  - Communication
- Git client
  - dealing with git repositories

Programming tools

- You may use any setting of OS/Compiler/IDE/Editor/Debugger
- We refer to:
  - C++14 Standard implemented by gcc (ver. 7.3.0+)
    - MinGW ver. 7.3 port for gcc for Window is OK;
  - CMake as a (meta-)building system;
  - Qt Creator as a powerful minimalistic free IDE
    - CLion, CodeBlocks, MS VS are also OK, but **on your own risk!**
    - There will be a set of topics related to building GUI applications base on Qt library. So, using of Qt Creator from the very beginning is highly recommended.

- The following tutorial refer to Qt Creator + MinGW + CMake configuration launched on Windows.
  - Adjust the settings according to your individual case.

gcc / MinGW port for Windows

- gcc, The GNU Compiler Collection, is the set of tools for preprocessing, compiling and linking applications in C, C++, Fortran and Objective-C.
  - We need only C and C++ parts of gcc.

- In *Linux*, use your preferable package manager to install gcc and related tools.

- *Windows* port of gcc is known as MinGW (Minimalist GNU for Windows) project, which includes (besides of the above-mentioned soft) some additional tools used in Linux while building applications (e.g. `make` utility).

- *MacOS* by default is supplied with alternative toolchain called CLang (C language family frontend for LLVM) and, generally, there is no need to install it separately;
  - CLang compatible (partially?) by the *frontend interface* with gcc;
  - try `g++ -v` command in terminal to check the version (and presence) of CLang system.

General preparations

- **Never** use any spaces, Cyrillic or other national alphabet- or special (math and so on) symbols in paths to your tools and projects!
  - `c:\Program Files\MinGW` ← horror
  - `c:\users\mike\Application 01\...` ← terror
  - `c:\...\Рабочий стол\Приложулечка\...` ← nightmare
  - `c:\users\Вася\...` ← even the Russian user name for Windows account will make you trouble even if you'll put your files in a different path. Unfortunately.

- Create a dedicated directory with shorter possible path for you programming stuff:
  - `c:\se\` ← good
  - `c:\se\tools\MinGW` ← perfect
  - `c:\se\prj\workshops\w01` ← brilliant

se is for *software engineering*

Before installing tools separately…

- First, consider a Qt Library package with embedded MinGW and CMake tools harmonized with the library itesf (pre-compiled binaries and so on).

- Go this slide for details.

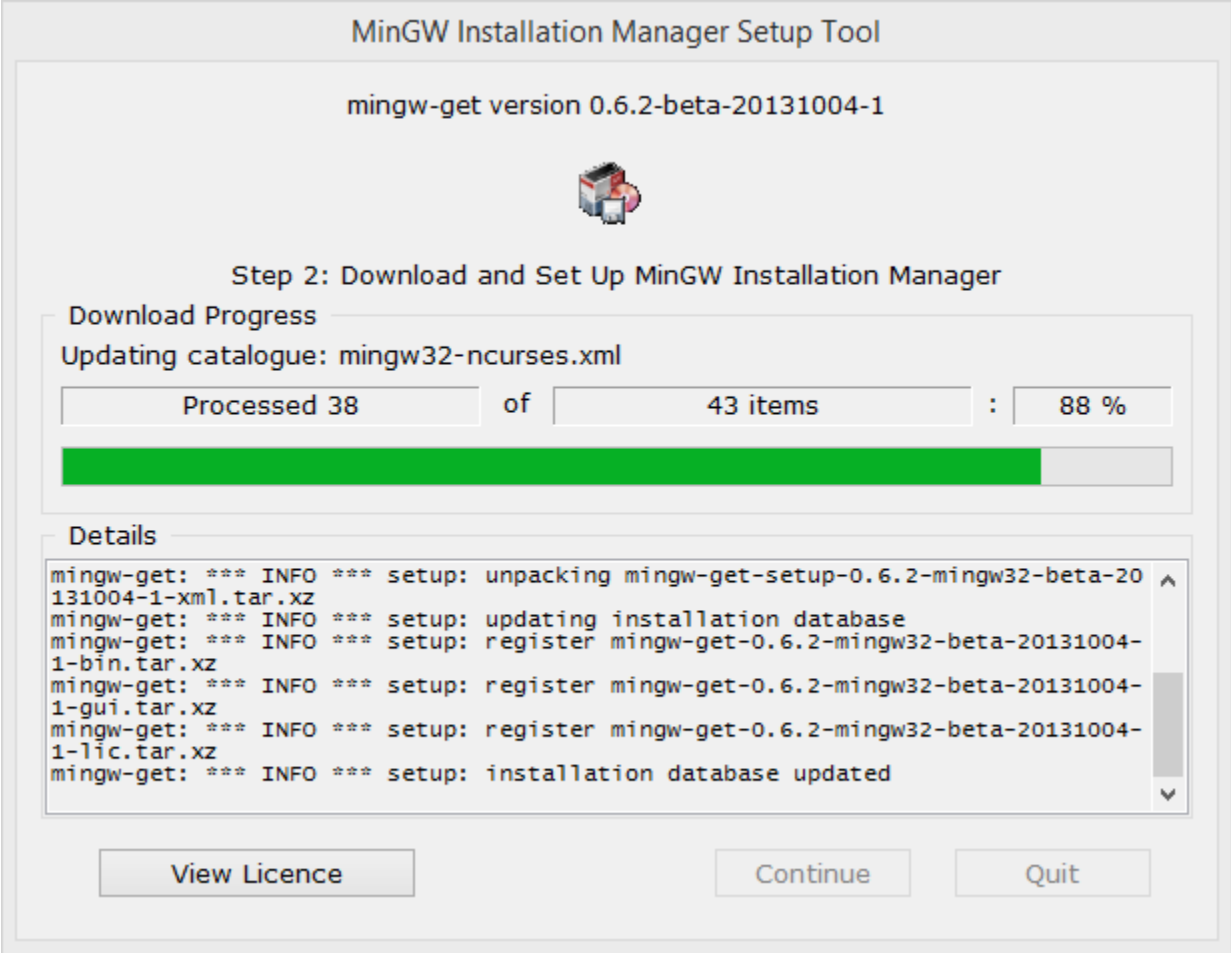
Installing MinGW for Windows

- You may install MinGW separately or as a part of Qt library. Now we consider a stand-alone installation.
- You will need both 32- and 64-bit version. Install them independently of each other.

1. Open https://sourceforge.net/projects/mingw/
2. Download web-installer (small executable, `mingw-get-setup.exe`). Consider correct platform (32 or 64, there are individual installers for them).
3. Click Install button.
4. In the next page provide a pretty path, e.g. `C:\se\tools\MinGW\64`.
5. Click Continue button and allow the installer to download meta-information about packages.

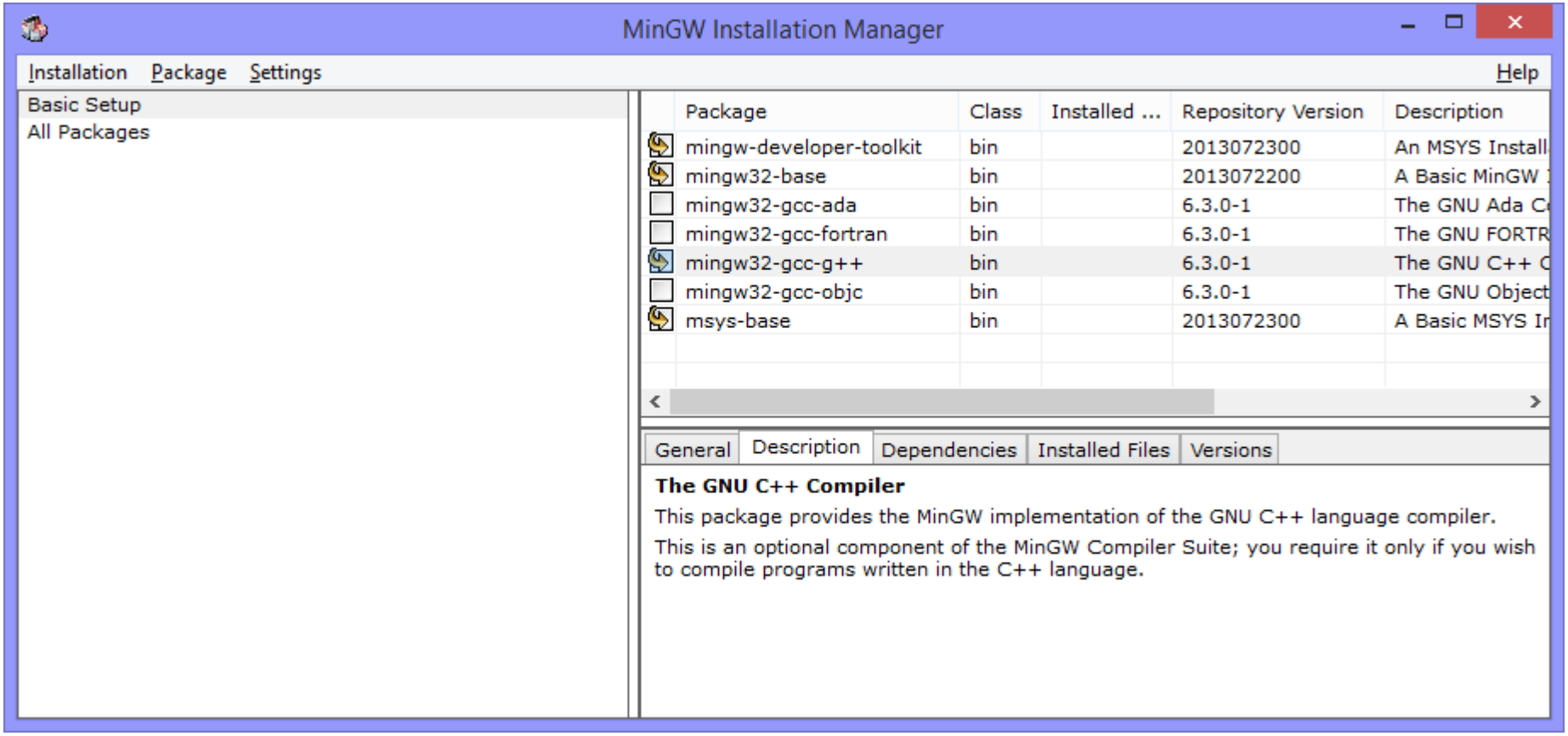
64 is for 64-bit version.
32 will be put next to 64

Installing MinGW for Windows (2)



Installing MinGW for Windows (3)

- In the list of available packages one need to select (*Mark for Installation*) all those ones related to C and C++ languages (g++) an base packages.
- Use menu Installation → Apply Changes.



Installing CMake for Windows

- Similar to MinGW CMake can be installed separately or it goes as a part of Qt library (while installing Qt Creator).

1. Navigate to https://cmake.org/
2. Download any installer (you may choose either 32- or 64-bir version, both are ok; you need only one).
3. Choose a pretty path, e.g. `c:\se\tools\cmake`.

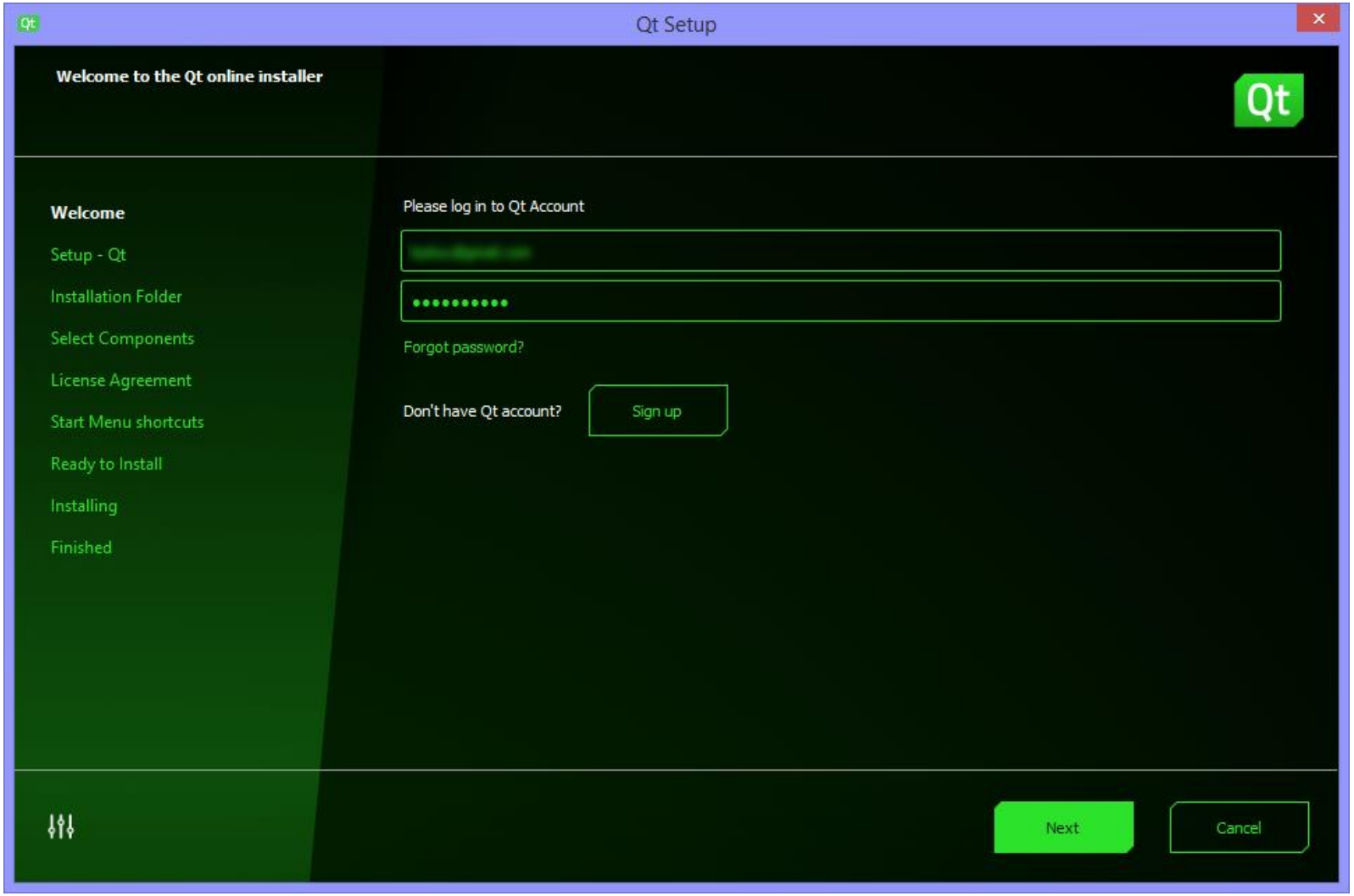
There is no need for any other special preparations with cmake.

Installing Qt Creator

- *Qt Creator* is a part of *Qt library* suit. However, it can be installed separately from the library itself.

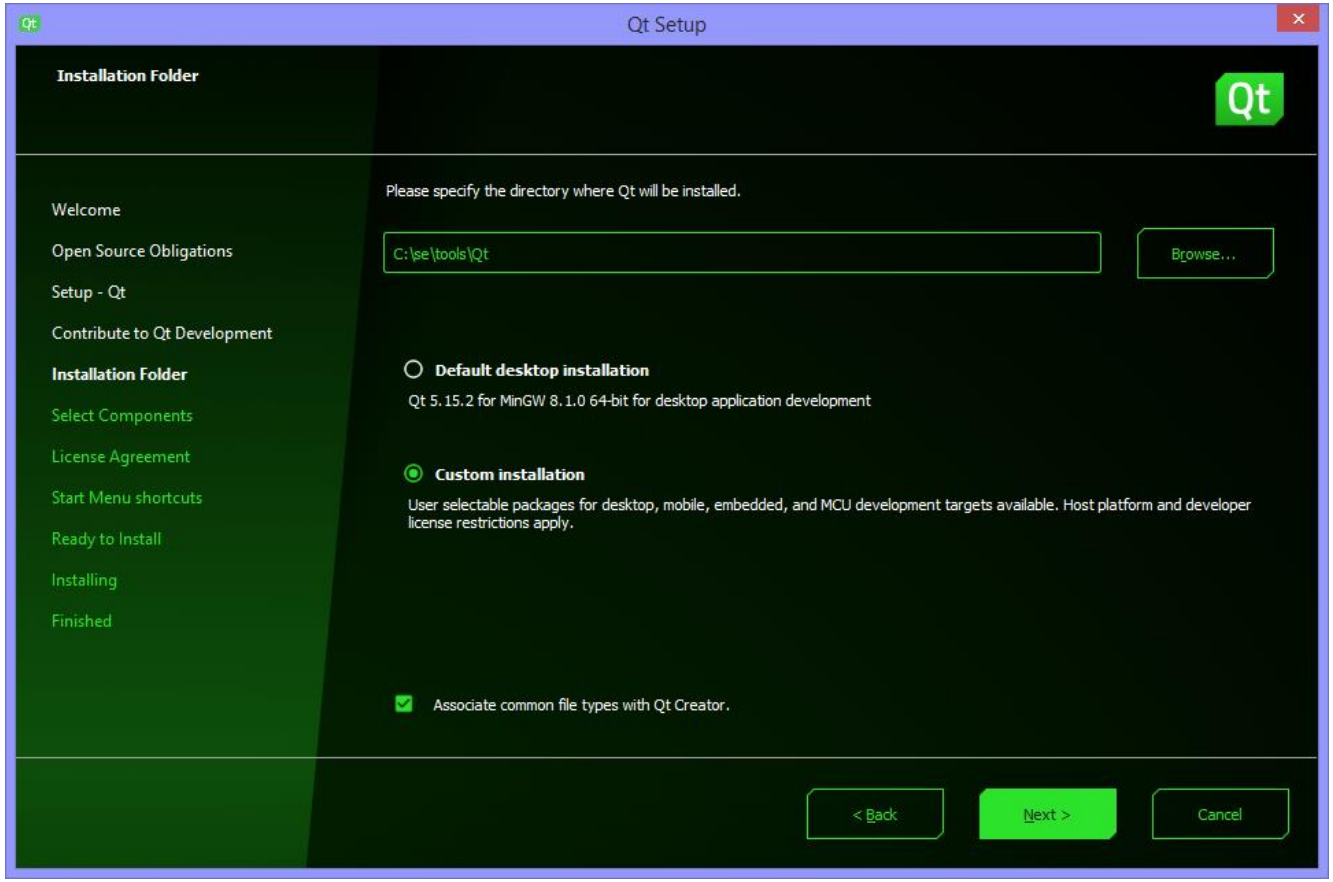
1. Navigate to https://www.qt.io/.
2. Go to Download section and choose “Downloads for open source users” option (box).
3. Scroll page down until find [Download the Qt Online Installer] button. Click it.
4. Download the online installer (small executable file). Run it and grant to it all necessary permission allowing to reach internet access.
5. In order to continue, you will need a Qt account. Please, make it through the official web-site, it doesn't take too much time (and, yes, it's free).

Installing Qt Creator (2)



Installing Qt Creator (3)

6. Apply all license stuff.
7. Choose a pretty path to the Qt base directory, e.g. `c:\se\tools\Qt`.
8. Choose “Custom installation” option.



Installing Qt Creator (4)

- From the list of components, choose the following:
  - Under the Qt/5.13.2/ node:
    - `Qt/5.13.2/MinGW 7.3.0 32-bit`
    - `Qt/5.13.2/MinGW 7.3.0 64-bit`
    - `Qt/5.13.2/Sources`

This allow to install precompiled libraries for MinGW 7.3 toolchain (not the toolchain itself!)

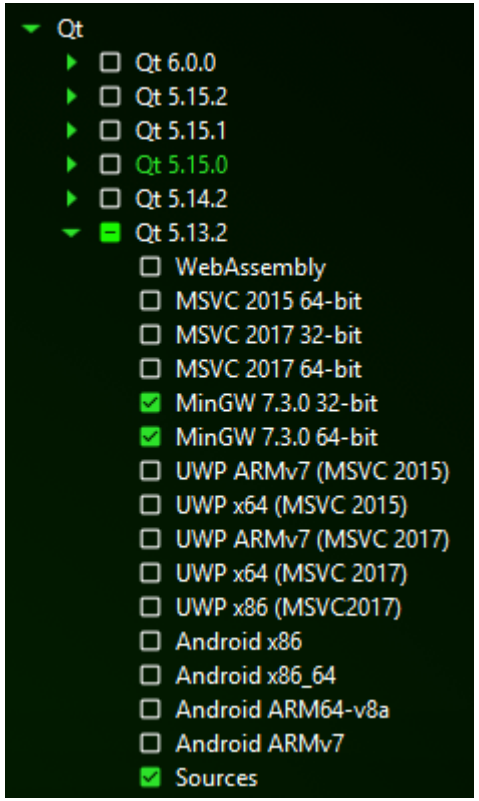
  - Under the Qt/Developer and Designer Tools node:
    - `Qt/Developer…/Qr Creator 4.14`
    - `Qt/Developer…/Qr Creator 4.14 CDB Debugger Support`
    - `Qt/Developer…/MinGW 7.3.0 32-bit`
    - `Qt/Developer…/MinGW 7.3.0 64-bit`
    - `Qt/Developer…/CMake 3.19.2 64-bit`

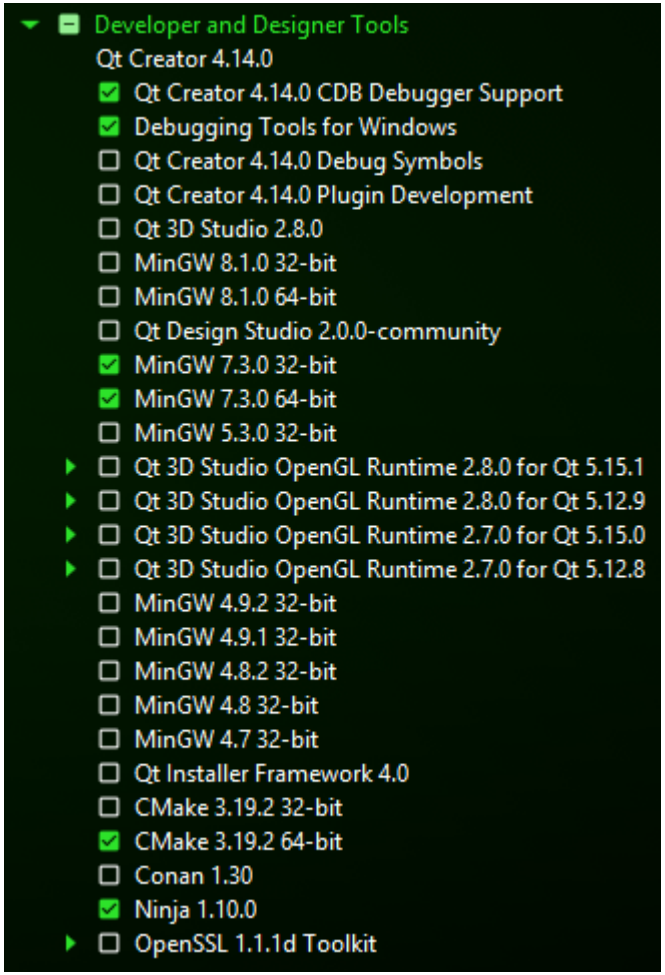
These are MinGW 7.3 toolchain itself — optionally

This is CMake tool

Installing Qt Creator (5)

- Do not choose other packages unless you really need them. They weigh tons!

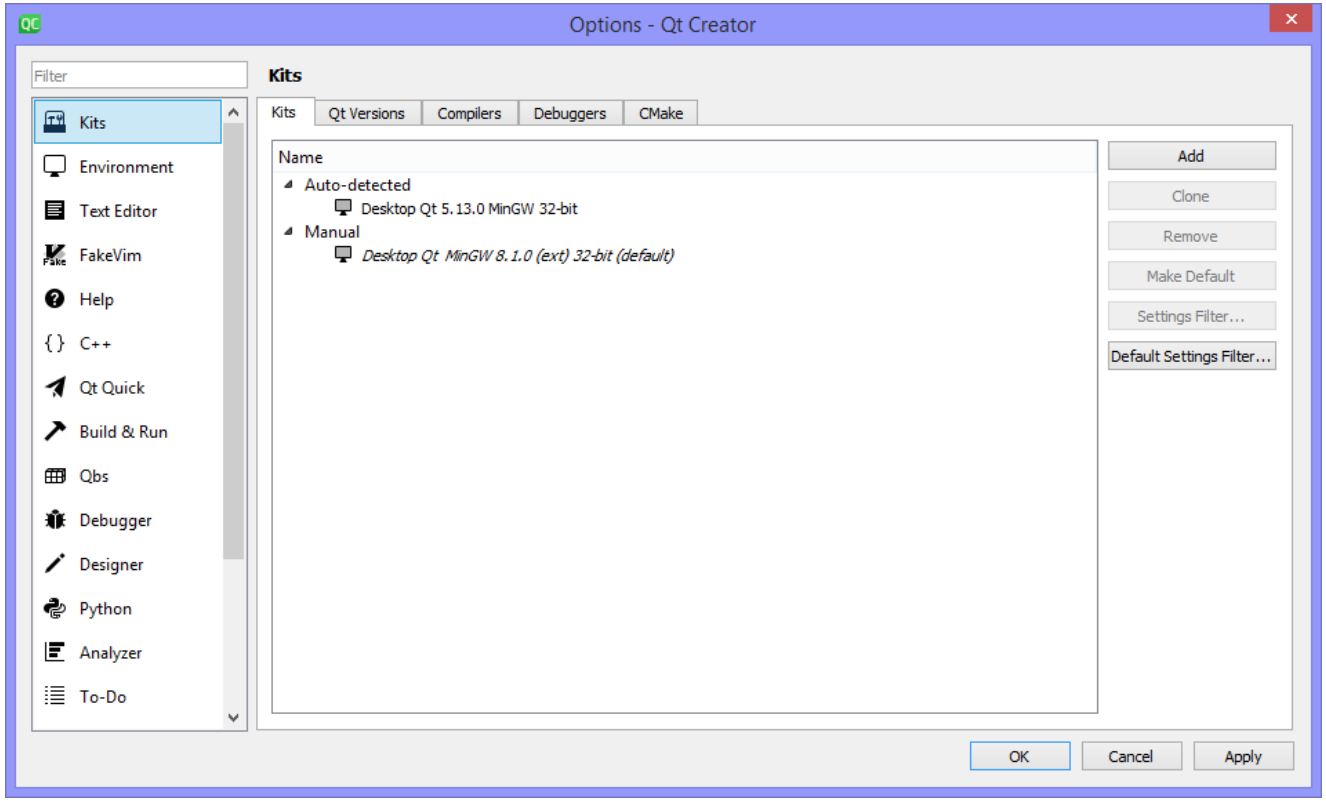




Configuring Qt Creator with custom MinGW and CMake

- Qt Creator can find your toolchain and CMake installations on its own. However, sometimes it could be useful to do this manually.

1. Launch Qt Creator.
2. Navigate to menu Tools → Options and make sure that Kits pane is open.



Configuring Qt Creator with custom MinGW and CMake (2)

- Click Add button and configure a newly created kit accordingly (see pictures below).

Name:	Desktop Qt %{Qt:Version} MinGW 8.1.0 (ext) 32-bit
File system name:	
Device type:	Desktop
Device:	Локальный ПК (default for Desktop)
Sysroot:	
Compiler:	C:
	C++: MinGW 8.1.0 32-bin for C++
Environment:	No changes to apply.
	Force UTF-8 MSVC compiler output
Debugger:	GNU gdb 8.1 for MinGW 8.1.0 32-bit
Qt version:	None
Qt mkspec:	
Additional Qbs Profile Settings:	
CMake Tool:	CMake64
CMake generator:	<none> - MinGW Makefiles, Platform: <none>, Toolset: <none>
CMake Configuration:	CMAKE_CXX_COMPILER:STRING=%{Compiler:Executable:Cxx}; CMAKE_C_COMPILER:STRIN...

Provide a name for you custom kit.

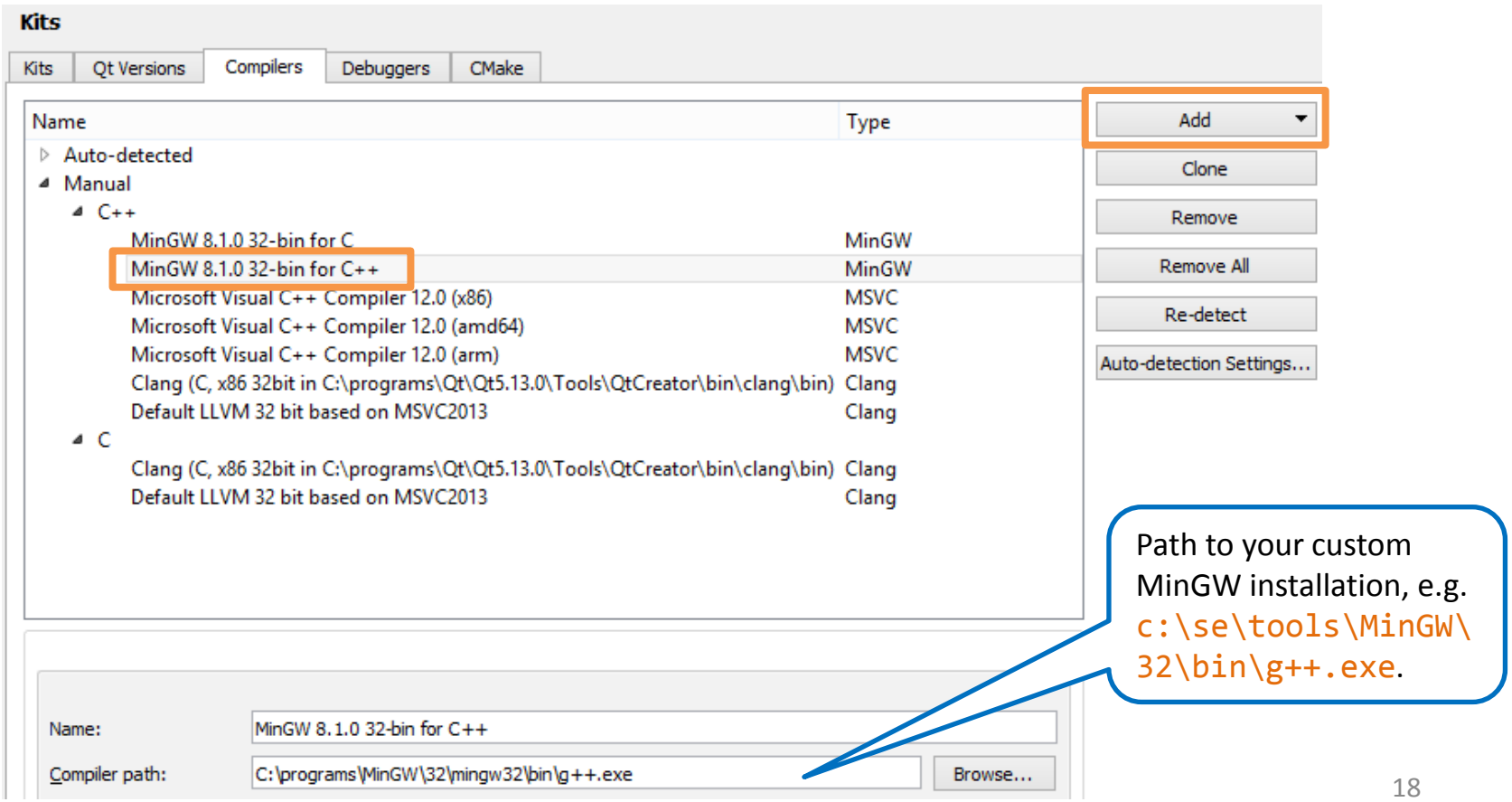
Check you C++ toolchain option. See the slide.

Make sure the gdb debugger is selected.

Check you CMake option. See the slide.

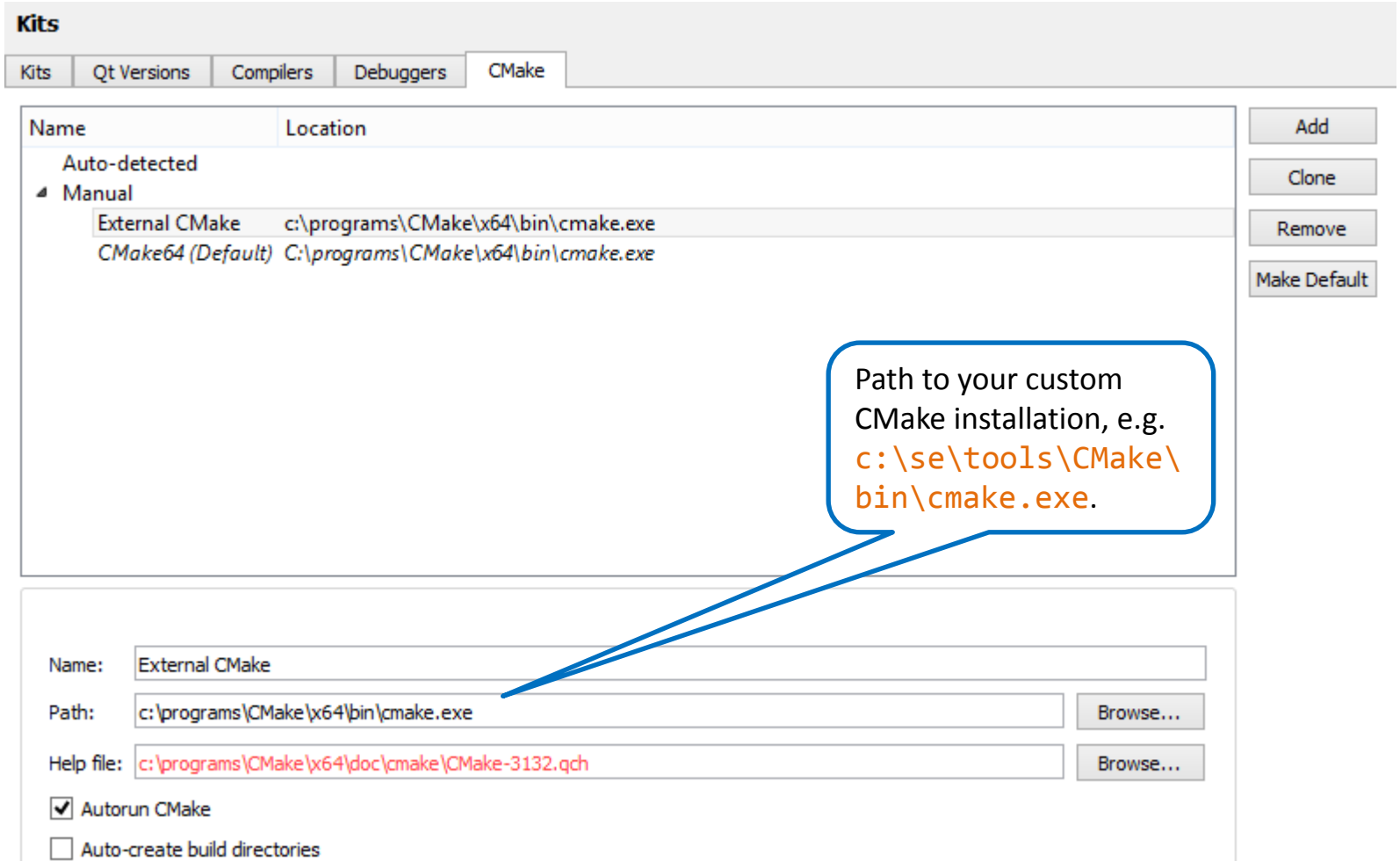
Qt Creator: setting custom compiler

- Open Kits → Compilers tab.
- Choose a custom compiler (add one using Add button if needed). Configure it accordingly (see pictures below).



Qt Creator: setting custom CMake

- Open Kits → CMake tab.
- Choose a custom Cmake (add one using Add button if needed). Configure it accordingly (see pictures below).



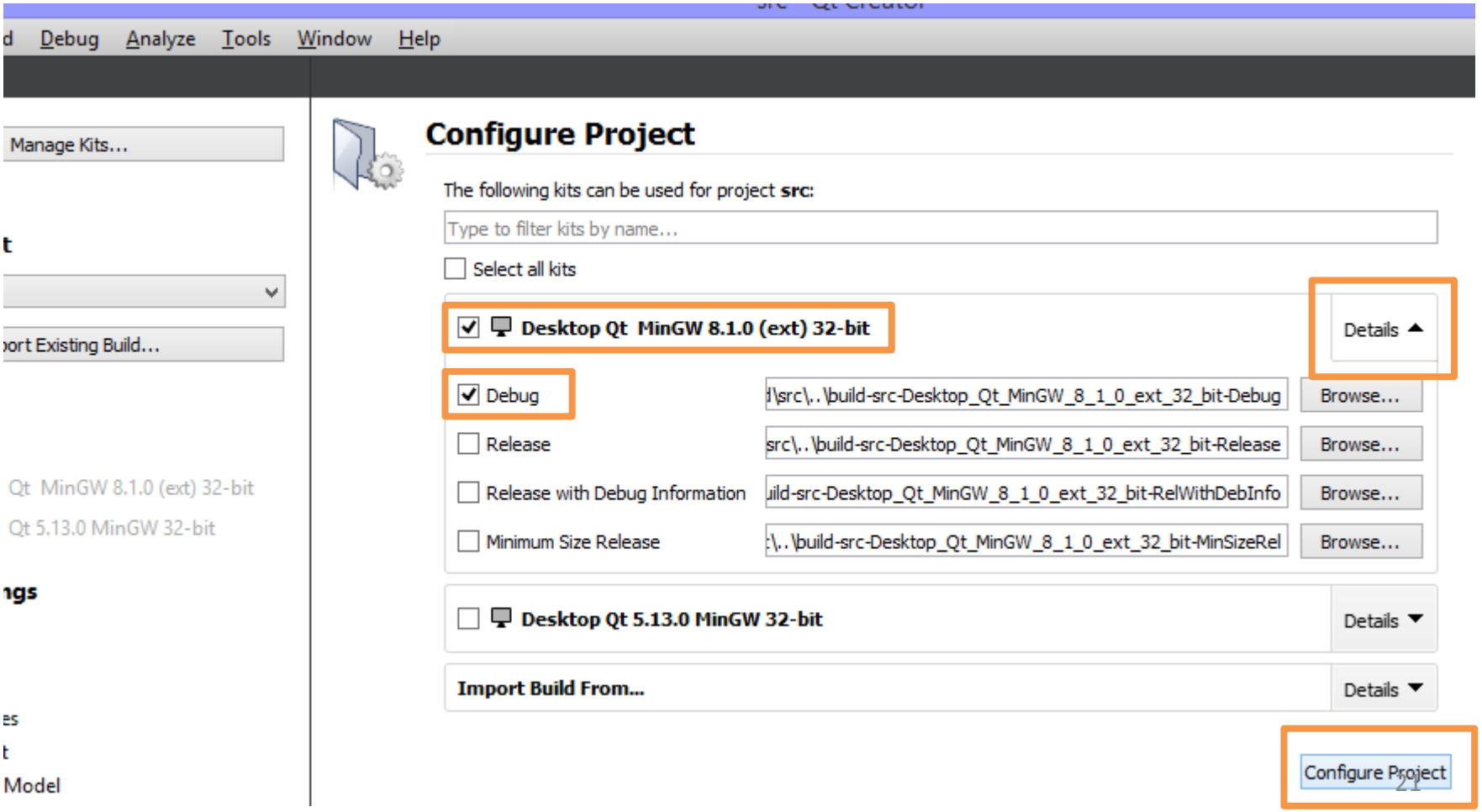
Building a minimum working example (MWE)

- Consider a prepared simplest CMake-based project offering for Workshop1.

1. Open it in Qt Creator: menu *File* → *Open File or Project*, then navigate to the folder with unpacked project, e.g. `c:\se\prj\workshops\01\code\helloworld\src` and choose `CMakeLists.txt` file.

Building a minimum working example (MWE) (2)

2. In the **Configure Project** form choose the desirable version of MinGW. Click [Details] button on the right and uncheck all checkbox except “Debug”. Leave the default path and click [Configure Project] button.



main.cpp (ex1 @ Workshop1) - Qt Creator

File Edit View Build Debug Analyze Tools Window Help

Projects

- Workshop1
  - CMakeLists.txt
  - ex1
    - Source Files
      - main.cpp

main.cpp | <Select Symbol> | Windows (CRLF) | UTF-8 | Line: 24, Col: 1

```
1  // include some definitions needed for woring with standard input/output (streams)
2  #include <iostream>
3  #include <climits>
4
5  // the entrypoint of the application
6  int main()
7  {
8      // put a string literal "Hello world!" to the standard output stream in order
9      // to print it at the console, which is normally associated with this stream
10     std::cout << "Hello world!";
11
12
13     size_t s1 = sizeof(char);
14     size_t s2 = sizeof(short);
15     size_t s3 = sizeof(int);
16     size_t s4 = sizeof(long);
17     size_t s5 = sizeof(long long);
18
19
20
21     // must return a "errorlevel code": 0 is for OK
22     return 0;
23 }
24
```

Project Structure

Qt Creator looks like this in case of successful configuration.

T... Google Test (none)

General Messages | Filter

```
-- Detecting CXX compile features - done
-- Configuring done
-- Generating done
CMake Warning:
  Manually-specified variables were not used by the project:

    CMAKE_C_COMPILER
    QT_QMAKE_EXECUTABLE


-- Build files have been written to: C:/Users/sergey/AppData/Local/Temp/QtCreator-niwPQd/qtc-cmake-wVpTLPHy
Elapsed time: 00:04.
```

No errors at CMakeLists.txt parsing stage.

Type to locate (Ctrl+D) | 1 Iss... 5 | 2 Search... | 3 Applic... | 4 Compil... | 5 QML D... | 6 To-Do ... | 7 Gener... | 8 Versio... | 9 Test R...

main.cpp (ex1 @ Workshop1) - Qt Creator

File Edit View Build Debug Analyze Tools Window Help

Projects

- Workshop1
  - CMakeLists.txt
  - ex1
    - Source Files
      - main.cpp

main.cpp | main() -> int | Windows (CRLF) | UTF-8 | Line: 14, Col: 24

```
1  // include some definitions needed for woring with standard input/output (streams)
2  #include <iostream>
3  #include <climits>
4
5  // the entrypoint of the application
6  int main()
7  {
8      // put a string literal "Hello world!" to the standard output stream in order
9      // to print it at the console, which is normally associated with this stream
10     std::cout << "Hello world!";
11
12
13     size_t s1 = sizeof(char);
14     size_t s2 = sizeof(short);
15     size_t s3 = sizeof(int);
16     size_t s4 = sizeof(long);
17     size_t s5 = sizeof(long long);
18
19
20
21     // must return a "errorlevel code": 0 is for OK
```

Qt Creator looks like this while running and debugging an app.

Debugger | Debugger Preset | Start debugging of startup project | Views

Debugger Perspectives

Perspective	Debugged Application
Debugger Preset	-

Breakpoint Preset

Debuggee	Function
-	-
-	-
-	-

T... | Google Test (none)

Compile Output | Filter

```
21:39:48: Running steps for project Workshop1...
21:39:48: Persisting CMake state...
21:39:51: Starting: "C:\programs\CMake\cmake-3.18.2-win64-x64\bin\cmake.exe" --build . --target all
Scanning dependencies of target ex1
[ 50%] Building CXX object CMakeFiles/ex1.dir/ex1/main.cpp.obj
[100%] Linking CXX executable ex1.exe
[100%] Built target ex1
21:39:53: The process "C:\programs\CMake\cmake-3.18.2-win64-x64\bin\cmake.exe" exited normally.
21:39:53: Elapsed time: 00:04.
```

"Build OK" is the only message appearing in the case of proper programming!

Type to locate (Ctrl+D) | 1 Iss... 5 | 2 Search... | 3 Applic... | 4 Compil... | 5 QML D... | 6 To-Do ... | 7 Gener... | 8 Versio... | 9 Test R...

The end!