C++ Programming I

Getting Started

C++ Programming
FS 2020

Dr. P. Arnold &
Dr. A. Schneider
Bern University of Applied Sciences

Agenda

- **Getting Started**
  - Linux
  - Windows
  - Mac

Agenda

- **Getting Started**
  - Linux
  - Windows
  - Mac
- **First Program**
  - CMake

Getting Started

Platform

Which platform to use?

C++ is platform independent, various IDE exists

- Windows Microsoft - Visual *C/C*++ , commercial
- MacOS X - XCode, free
- Unix - KDevelop, Eclipse, **QtCreator** etc., Open-Source, i.e. source code available
- Unix - GCC = Gnu Compiler Collection, free compiler

For newcomers, Linux (e.g Ubuntu) is the recommended development platform due to the free and well-engineered C++ 11 compiler.

Alternatively install Virtual Box, although not really convenient for software development!

In this course

(K)Ubuntu and **QT-Creator** are default.

Linux

Debian based Distributions

The GCC (Gnu Compiler Collection including the gcc and g++ compilers) is usually already installed with Ubuntu (and Mac). To test, open the unix terminal and type “gcc –version”. On my Kubuntu machine this gives the following output:

gcc-version

```
$ gcc --version

gcc (Ubuntu 5.4.0-6ubuntu1~16.04.5) 5.4.0 20160609 Copyright (C)
     2015 Free Software Foundation, Inc. This is free software;
     see the source for copying conditions.  There is NO
     warranty; not even for MERCHANTABILITY or FITNESS FOR A
     PARTICULAR PURPOSE.
```

Make sure your compiler version is at least **gcc 4.8** to enable c++11 features.

Linux

Debian based Distributions

To install the build tools and the complete Qt-Creator/qt5 toolchain with examples and documentation simply run:

Install Qt Creator IDE and tools

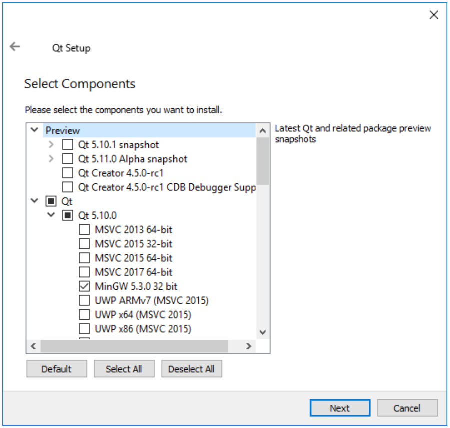
```
% Build tools
sudo apt-get install build-essential gdb cmake cmake-curses-gui

% Install Qt with examples and openGL support for widgets
sudo apt-get install qtcreator qt5-default qttools5-dev-tools
     qt5-doc qtbase5-examples qtbase5-doc-html libgl1-mesa-dev
```

Windows 10

Install Qt with MinGW (recommended)

The installation on Windows with MinGW-Compiler is straight-forward following these instructions:

1. Get the open source version of Qt from: `https://www.qt.io/download`
2. Follow the instructions of the installer. Skip the account creation
3. Select Qt 5.12.1 (or newer) sub-folder for installation with corresponding MinGW-Compiler



Qt Setup

Select Components

Please select the components you want to install.

- Preview
  - [ ] Qt 5.10.1 snapshot
  - [ ] Qt 5.11.0 Alpha snapshot
  - [ ] Qt Creator 4.5.0-rc1
  - [ ] Qt Creator 4.5.0-rc1 CDB Debugger Supp
- ■ Qt
  - ■ Qt 5.10.0
    - [ ] MSVC 2013 64-bit
    - [ ] MSVC 2015 32-bit
    - [ ] MSVC 2015 64-bit
    - [ ] MSVC 2017 64-bit
    - [✓] MinGW 5.3.0 32 bit
    - [ ] UWP ARMv7 (MSVC 2015)
    - [ ] UWP x64 (MSVC 2015)
    - [ ] UWP x86 (MSVC 2015)

Latest Qt and related package preview snapshots

Windows 10

Install CMake

- CMake is an open-source, cross-platform family of tools designed to build, test and package software. CMake is used to control the software compilation process using simple platform and compiler independent configuration files, and generate native makefiles and workspaces that can be used in the compiler environment of your choice

1. Get CMake from: `https://cmake.org/`
2. For best experience with Qt-Creator get version 3.7.2 or later: `https://cmake.org/files/v3.13/cmake-3.13.4-win64-x64.msi`
3. During install, check “add CMake to system tools” for convenience, so that CMake gets detected automatically by Qt Creator
4. Verify you have a working tool-chain, i.e. you can build and run simple programs. Therefore, launch Qt Creator and create a CMake based C++ project.

Mac

Install Qt XCode

For MacOS X the C++ -Compiler is part of XCode.

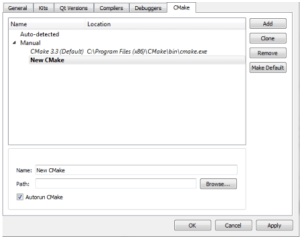
1. Install XCode from Apples App Store
2. Get the open source version of Qt from:
   `https://www.qt.io/download`
3. Follow the instructions of the installer. Skip the account creation

Mac

Install CMake

- CMake is an open-source, cross-platform family of tools designed to build, test and package software. CMake is used to control the software compilation process using simple platform and compiler independent configuration files, and generate native makefiles and workspaces that can be used in the compiler environment of your choice

1. Get CMake from: `https://cmake.org/`
2. For best experience with Qt-Creator get version 3.7.2 or later: `https://cmake.org/files/v3.13/cmake-3.13.4-Darwin-x86_64.dmg`
3. Set up cmake according to the official Qt documentation: `http://doc.qt.io/qtcreator/creator-project-cmake.html`



Name	Location
Auto-detected	
Manual	
CMake 3.3 (Default)	*C:\Program Files (x86)\CMake\bin\cmake.exe*
New CMake	

Name: New CMake

Path:

[✓] Autorun CMake

First Program

First Program

Hello World

- Get QT-Creator (Homework01.pdf)
- Compile and run the helloworld example in a console
- Compile with: g++ helloworld.cpp -o helloworld
- In a console run with: ./helloworld

First Program

Hello World

- Get QT-Creator (Homework01.pdf)
- Compile and run the helloworld example in a console
- Compile with: g++ helloworld.cpp -o helloworld
- In a console run with: ./helloworld

```
#include <iostream>

int main()
{
        std::cout << "Hello World" << std::endl;
        return 0;
}
```

```
g++ helloworld.cpp -o helloworld
./helloworld

Hello World!
```

First Program
Hello World - Analysis

```cpp
// Pre−processor directive
#include <iostream>

// Start of your program
int main()
{
   /* Write to the screen using std::cout */
   std::cout << "Hello World" << std::endl;

   // Return a value to the OS
   return 0;
}
```

First Program

Hello World - Analysis

```cpp
// Pre–processor directive
#include <iostream>

// Start of your program
int main()
{
   /* Write to the screen using std::cout */
   std::cout << "Hello World" << std::endl;

   // Return a value to the OS
   return 0;
}
```

- The preprocessor directive `#include` command occurs before the actual compilation starts. It tells the preprocessor to include the content of the specified file at the current line. In this example, `iostream` lets us use the `std::cout` and `std::endl` functions to write on the screen.
- The `int main()` is the body of your Program. The execution of a C++ program always starts here.
- The `{}` indicate that everything inside them is part of the function. In this case, they denote that everything inside is a part of the "main" function.

First Program
Hello World - Analysis

```cpp
// Pre—processor directive
#include <iostream>

// Start of your program
int main()
{
   /* Write to the screen using std::cout */
   std::cout << "Hello World" << std::endl;

   // Return a value to the OS
   return 0;
}
```

- The “;” denotes the end of a line. Most lines of C++ code need to end with a semicolon.
- `cout` (console-out) writes the “Hello World” to the screen. `cout` is a *stream* defined in the standard `std` and therefore `std::cout`. The stream insertion parameter « puts the text in the stream and `std::endl` ends a line.
- `main()` is a function and always returns an integer: 0 for success and -1 in the event of an error. Other error codes using the available range of integers can be used.

First Program

Hello World - Analysis

```cpp
// Pre-processor directive
#include <iostream>

// Start of your program
int main()
{
   /* Write to the screen using std::cout */
   std::cout << "Hello World" << std::endl;

   // Return a value to the OS
   return 0;
}
```

- C++ supports two styles of comments
  - `//` indicates the start of a comment until the end of the line
  - `/* */` indicates that the contained text is a comment
- Use `using namespace std` in order to use `cout` instead of `std::cout`

CMake

CMakeLists Example

```cmake
# Set Name of project and language
project(HelloWorld LANGUAGES CXX)

# Set cmake version
cmake_minimum_required(VERSION 3.0)

# set build type to Debug/Release
set(CMAKE_BUILD_TYPE "Debug")

# Create executable using the specified src
add_executable(${PROJECT_NAME} helloworld.cpp)

# Define required c++ standard to C++11
target_compile_features(${PROJECT_NAME} PUBLIC cxx_std_11)

# Set compile options, enable warnings
target_compile_options(${PROJECT_NAME} PRIVATE
    $<$<OR:$<CXX_COMPILER_ID:Clang>,$<CXX_COMPILER_ID:GNU>>: -Wall>
    $<$<CXX_COMPILER_ID:MSVC>: /W3>
)
```

- Comments are set with #
- Demo - Getting Started

Thank You
Questions

???