Bern University
of Applied Sciences

# 414078-HS2020-0 - C++ Programming II

# EXERCISE-03

## 1 Introduction

In this exercise we will finally start using the Qt-library to realize an user interface and integrate QCustomPlot in order to plot some data on it. In addition, you're given a ECG signal on which you'll calculate the average heartbeat frequency, of course using as much STL containers and algorithms as possible. As this is a more extensive exercises you'll get **2 points** for the submission.

You will learn the following topics when completing this exercise:

- Integrating Qt5 to your `CMake` project
- Add QCustomPlot to your project and some basic usage
- Plot a ECG signal and calculate the average heartbeat rate

## 2 Qt5 Library & CMake (0.5 P)

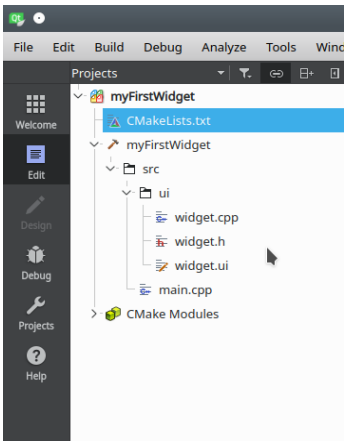
In order to create an application with user interface we'll include the Qt library in our CMake project. Unfortunately, the CMakeLists.txt file has to be written manually and there's no support of Qt Creator for automation so far. Therefore, we go through the necessary steps in this exercise and you'll learn the basic syntax of CMake.

The following is a step-by-step instruction to point you in the right direction:

1. As usual, create an Non-Qt Project with CMake as build tool.
2. The example CMakeLists.txt from exercise 1 on Ilias is already a good starting point - copy-/paste it into your new project. Rename the project and adapt the source files. Run CMake, Compile and run.
3. Now add a Qt Designer Form Class for a Widget to you project (File → New File ... → Qt → Qt Designer Form Class → Widget) and name it e.g. "Widget". In order to have a clean structure from the beginning, realize the following file hierarchy:

```
|- CMakeLists.txt
|-- src
    |-- main.cpp
    |-- ui
        |-- widget.cpp
        |-- widget.h
        |-- widget.ui
```

4. To add Qt5 support to our project copy the relevant part of the CMakeLists.txt example from http://doc.qt.io/qt-5/cmake-manual.html to your project. Keep the part with the CMAKE_CXX_FLAGS. Go through the file and learn to understand how CMake works.
5. In larger projects, it is convenient to assign (SET) all source files to a variable (e.g. helloworld_SRCS), which are then added later to the executable. Rename this variable to APP_SOURCES and add the sources, similar as in the example, but with the full path in respect to the CMakeLists.txt file, e.g src/ui/widget.cpp.
6. In order to tell CMake where to find the source files when placed in sub-directories, we have to specify a second variable which we name APP_INCLUDE_DIRS containing the paths to the files, e.g. src & src/ui. These include directories are finally specified with INCLUDE_DIRECTORIES( ${APP_INCLUDE_DIRS})
7. When running CMake now and everything is set up correctly, you should see the file structure in the project explorer of Qt-Creator




8. Finally to obtain our widget we have to run the application by returning execution control to its event loop as shown in the adapted main.cpp:

```
#include <QApplication>
#include "widget.h"

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);

    widget w;
    w.show();

    return a.exec();
}
```

9. You should see an empty window popping up when running the app!
10. Make sure to save the state of this empty widget project for later use, as it is a perfect starting point for other projects.

# 3 QCustomPlot Library (0.5 P)

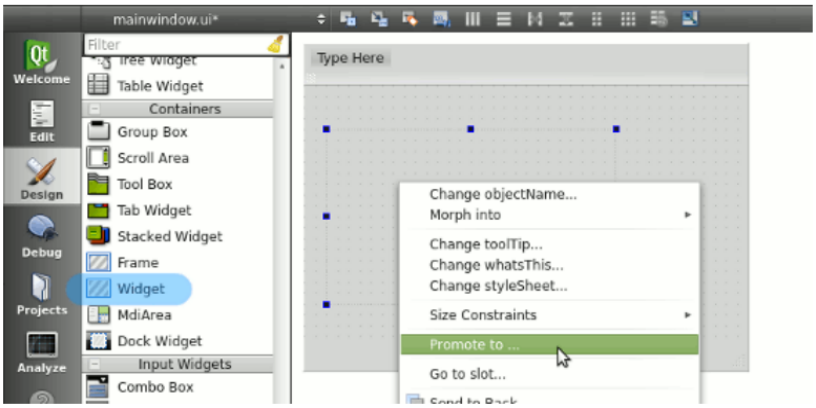
Now we integrate `QCustomPlot` in our project:

1. Copy the source files to your project, realize the following file hierarchy:

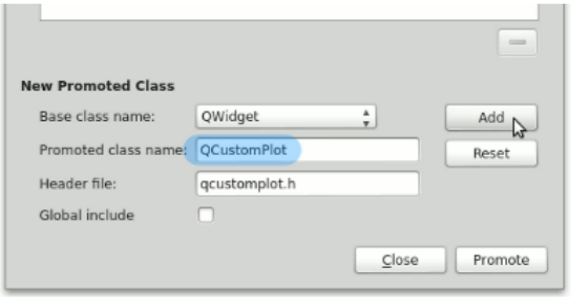
```
|- CMakeLists.txt
|-- src
    |-- main.cpp
    |-- ui
        |-- qcustomplot-source
        |   |-- qcustomplot.cpp
        |   |-- qcustomplot.h
        |-- widget.cpp
        |-- widget.h
        |-- widget.ui
```

2. To add the sources to your `CMake` Project you have to adapt the `APP_SOURCES` and `APP_INCLUDE_DIRS`, respectively. Run `CMake`, Compile and run.
3. `QCustomPlot` requires the Qt-module `QtPrintSupport`. Add it similarly as it is done for the the widget module, i.e. use `find_package` and link the libraries. After a successful build, the project is now ready to use `QCustomPlot`.
4. To add a plot to your widget change to the designer mode, i.e. open the `widget.ui` file and follow the guide below:

The project is now ready to use QCustomPlot. Place a regular QWidget on your form in the desired location. Right click on it and hit Promote to...

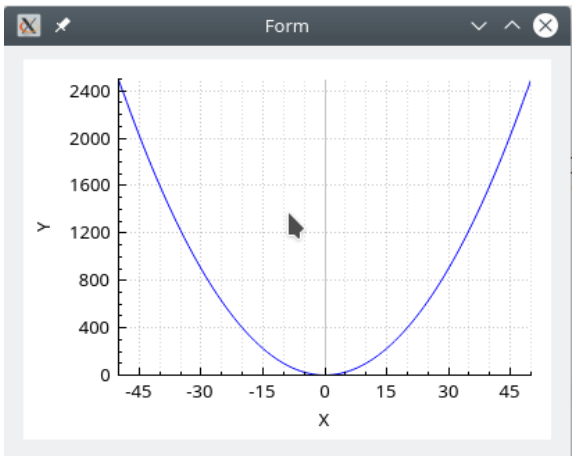



In the appearing dialog, enter `QCustomPlot` in the input field next to *Promoted class name*. The input next to *Header file* should automatically fill with the correct `qcustomplot.h` value. Hit *Add* to add QCustomPlot to the promoted classes list and finally hit *Promote* to turn the QWidget on your form into a QCustomPlot.




5. Name the `QCustomPlot` widget for example `qcpwidget`. You have now access to the plot and can use the code below to plot some data. Write a function `void plotData()` in `widget.cpp`, which is e.g. called in the constructor and copy/paste the snippet:

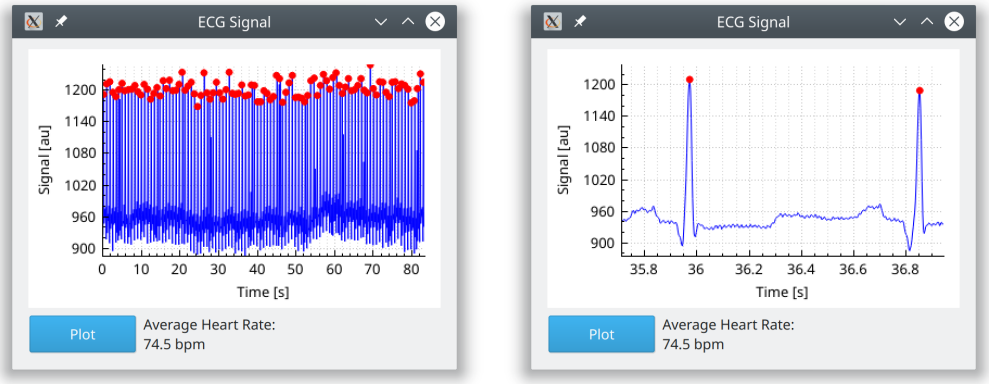
```
// Set userinteraction: zoom and drag
ui->qcpwidget->setInteraction(QCP::iRangeDrag,true);
ui->qcpwidget->setInteraction(QCP::iRangeZoom,true);
connect(ui->qcpwidget,SIGNAL(mouseDoubleClick(QMouseEvent*)),
        ui->qcpwidget, SLOT(rescaleAxes()) );

// Add Graph and set some properties
ui->qcpwidget->addGraph();
ui->qcpwidget->xAxis->setLabel("X");
ui->qcpwidget->yAxis->setLabel("Y");
ui->qcpwidget->xAxis->grid()->setSubGridVisible(true);

// Create data: f(x) = x^2
std::vector<double> X(101);
std::iota(X.begin(),X.end(),-50);
std::vector<double> Y(101);
std::transform(X.begin(),X.end(),
               Y.begin(),
               [](double x){return x*x;});

// Plot data
ui->qcpwidget->graph(0)->setData(QVector<double>::fromStdVector(X),
                                 QVector<double>::fromStdVector(Y));

//
ui->qcpwidget->rescaleAxes();
ui->qcpwidget->replot();
```




6. Get a little familiar with `QCustomPlot` (`https://www.qcustomplot.com/`) and add a second graph with the function `f(x) = 1/x` to the same plot using red circle markers without a connecting line.

## 4 EKG Signal - Heart Rate (1 P)

Finally we can start with the actual task! Write a program which reads and plots the ecg signal read from *ekg.txt*, finds the heartbeats and visualizes them. In addition, calculates the average heart rate and display the result in a `QLabel` on the GUI. Assume a sampling rate of 360 Hz. Your result should look similar to this:




## 5 Submission

Submit your source code (as a zip-file) to Ilias **before the deadline** specified in Ilias.