C++ for CMEA II

Kjetil Olsen Lye

September 24, 2015

C++ in CMEA II

- Focus of the course is not C++.
- *However,* all programming exercises will be in C++.
- C++ is *very* relevant for later studies.
- Loads of use cases require C++ over higher level languages
- C++ is *fun to master*.
- We assume you know core C++.
- Main focus is *numerical* use of C++.

Necessary software

For this course, you will need

- A C++ compiler, one of:
  - **GCC**
  - **clang**
  - **Microsoft Visual Studio**
- Tools for plotting, one of:
  - **MATLAB**
  - **Python** with **numpy/matplotlib**
- *Recommended:* **cmake**

You probably have these programs from a previous course[1]

[1] Install instruction at: `https://www2.math.ethz.ch/education/bachelor/lectures/hs2015/other/cmea2/installation.txt`

Choosing between MATLAB and Python

You can choose yourself which platform you want to use.

We recommend you use the one you are most familiar with.

If the numbers differ a lot, we only will supply templates for the most used one.

Using C++ for numerical simulations

We will concentrate on a small feature set of C++

- We will only use double precision (`double`)
- We will use `std::vector` as our main container
- We will (probably) not need `unsigned` types
- Towards the end, we may use some external libraries:
  - Probably `eigen` for linear equations
  - More information will come later

C++ Concepts you should be familiar with

If you did the assignments in CMEA I, you know these things.

- Fundamental programming skills (loops, tests, functions, objects)
- A basic understanding of the C++ type system
- The difference between a value, a reference and a pointer
- Use of templated functions and classes (eg. `std::vector`)
- Basic output using `std::cout`

C++: Difference between references and values

What is printed?

```
#include <iostream>
void f1(int x) {
      x = 10;
}
void f2(int& x) {
      x = 11;
}
void f3(int* x) {
      *x = 12;
}
int main() {
     int x = 8;
     f1(x);  std::cout << x << std::endl;
     f2(x);  std::cout << x << std::endl;
     f3(&x); std::cout << x << std::endl;
}
```

Memory management in C++

Memory management will not be a central theme in this course. This is just a short list of things to think about.

- Any allocated object in C++ must be deleted
- If you allocate on the stack, the object is automatically deleted:

```
// must be deleted
double* a = new double[10];
delete[] a;
// will be deleted automatically
std::vector<double> b(10);
```

- **Avoid** `new/delete` if you can.
- It's a good idea to use a memory checker (valgrind, drmemory)
- For the advanced users: Check out `std::shared_ptr` and `std::unique_ptr`

Building C++ programs

- Building C++ programs is *complicated*.
- In this course, all template code will be built with **CMake**.
- On Linux, Cygwin and Mac, using **CMake** is straightforward.

First time use (per project):

```
# Create build directory
mkdir build
cd build
cmake ..
make
```

To build again, only `make` is necessary:

```
cd build
make
```

Running the files from the command line

Always read the README.txt on how to run the templates!
This is relevant for Linux, OS X and Cygwin (not MSVS).
We can run the generated executables directly in the command line:

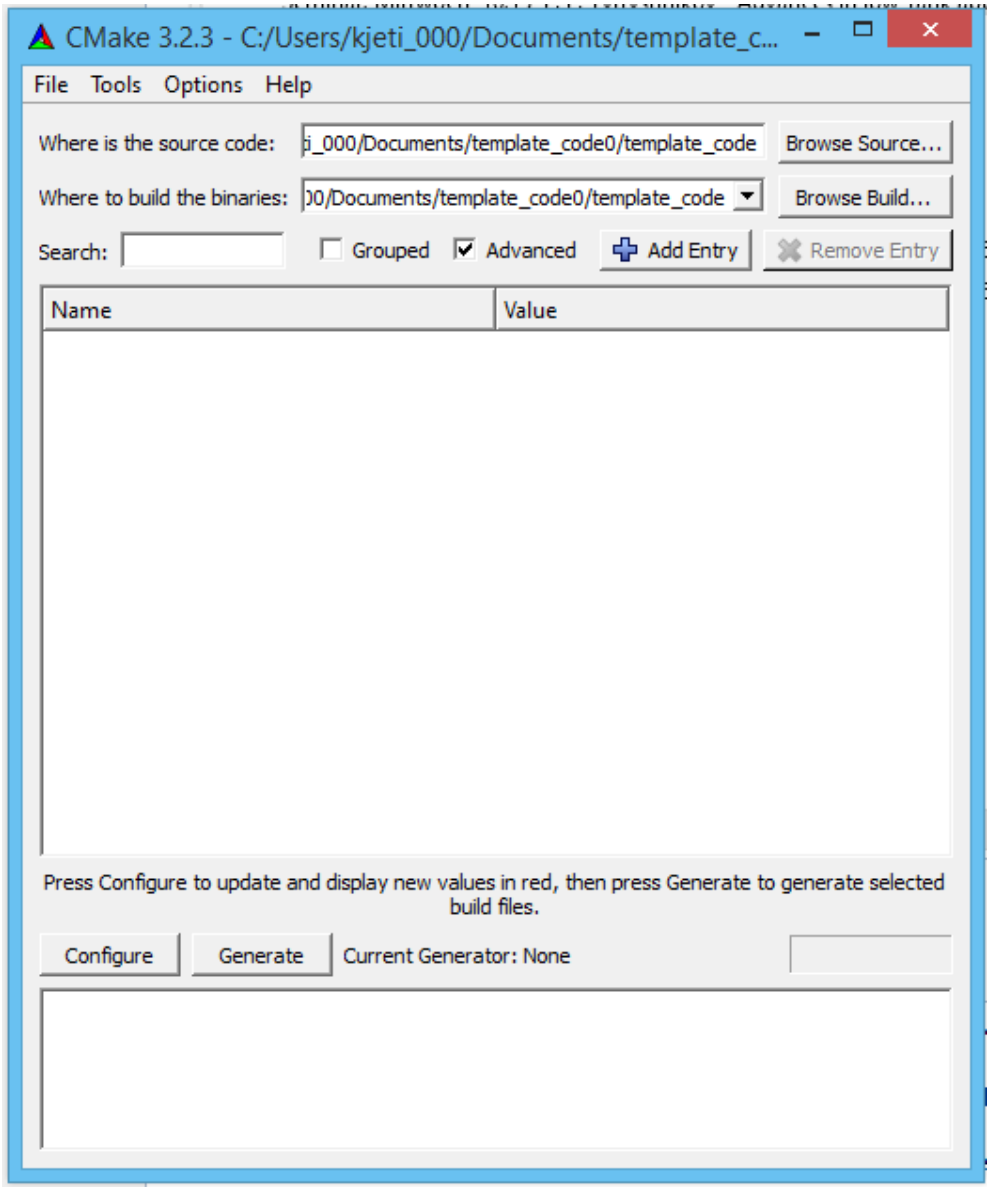
```
# NOTE: series01a is not executable
./series01c
./series01d
```

Windows building

It is possible to build from the command line, but we recommend to use the graphical approach.

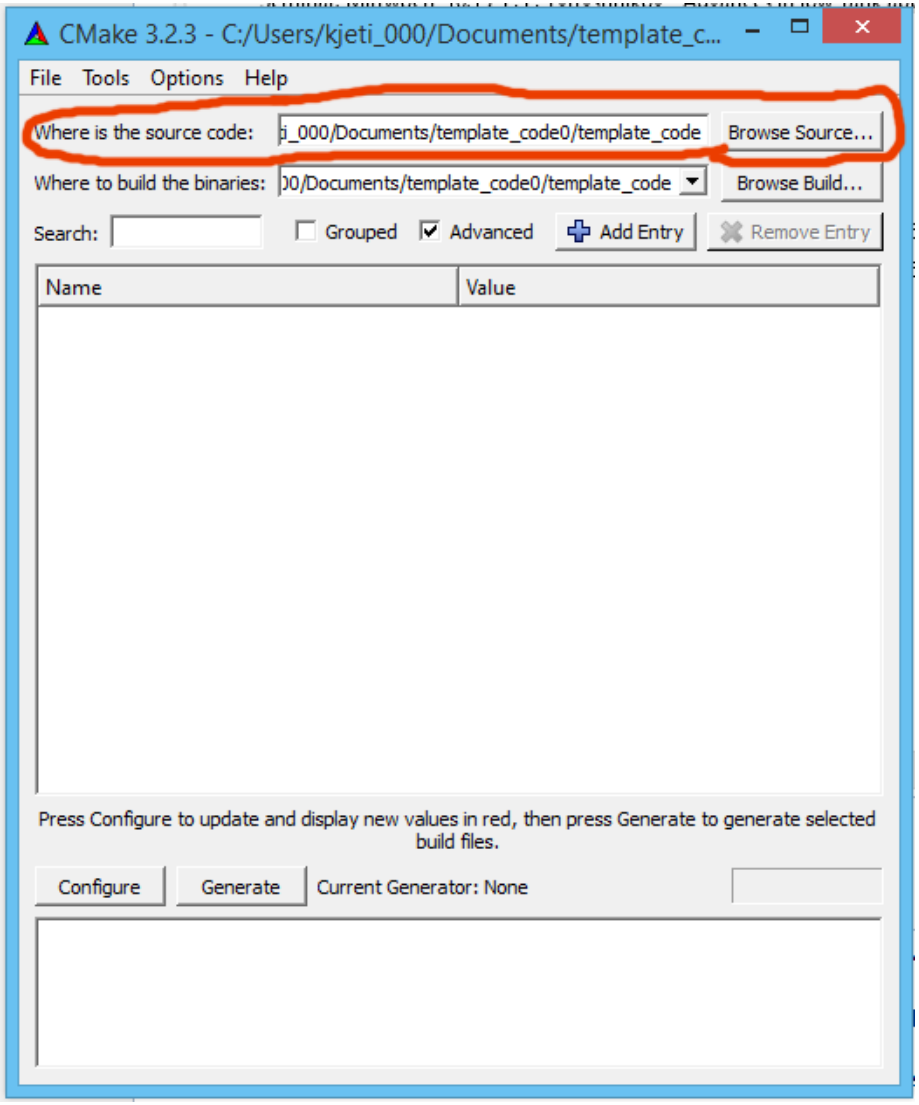
Windows building

1) Start CMake



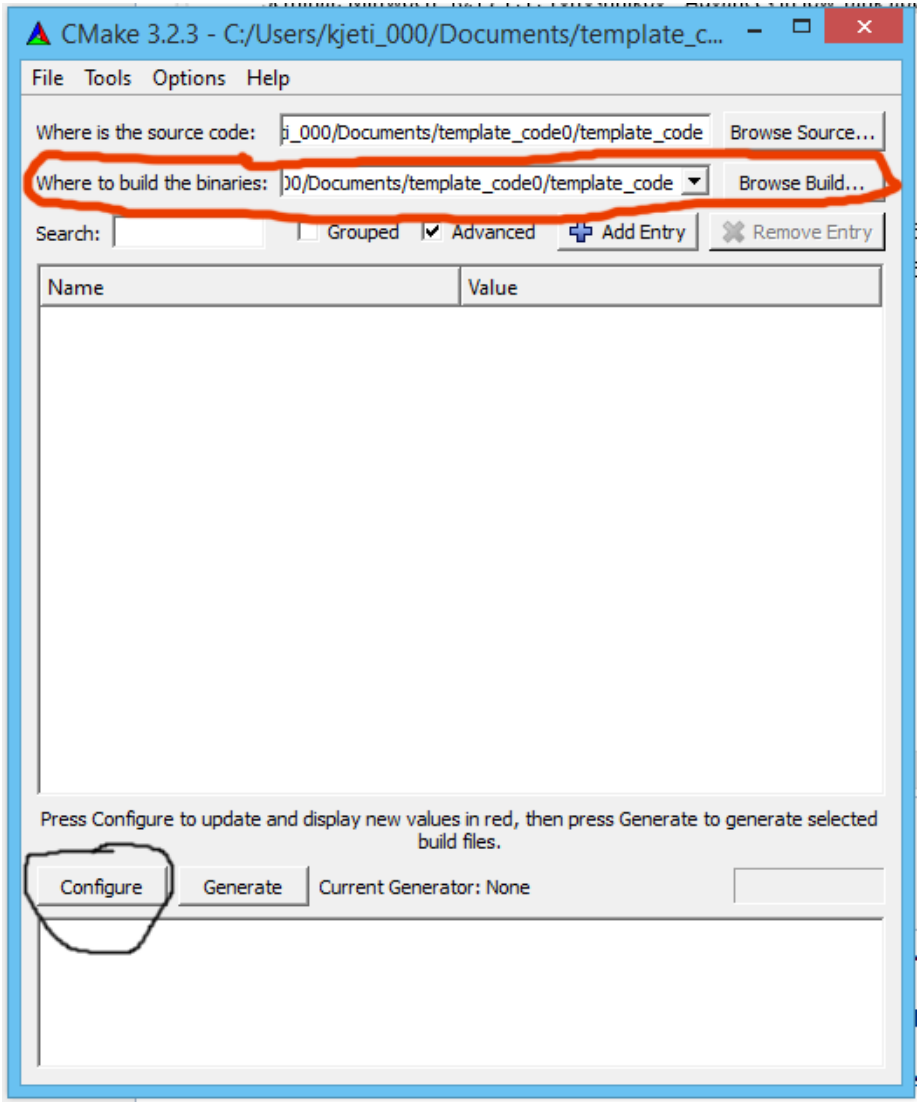
Windows building

In the field "Where is the source code", insert the path to the top folder contaning the `CMakeLists.txt` file



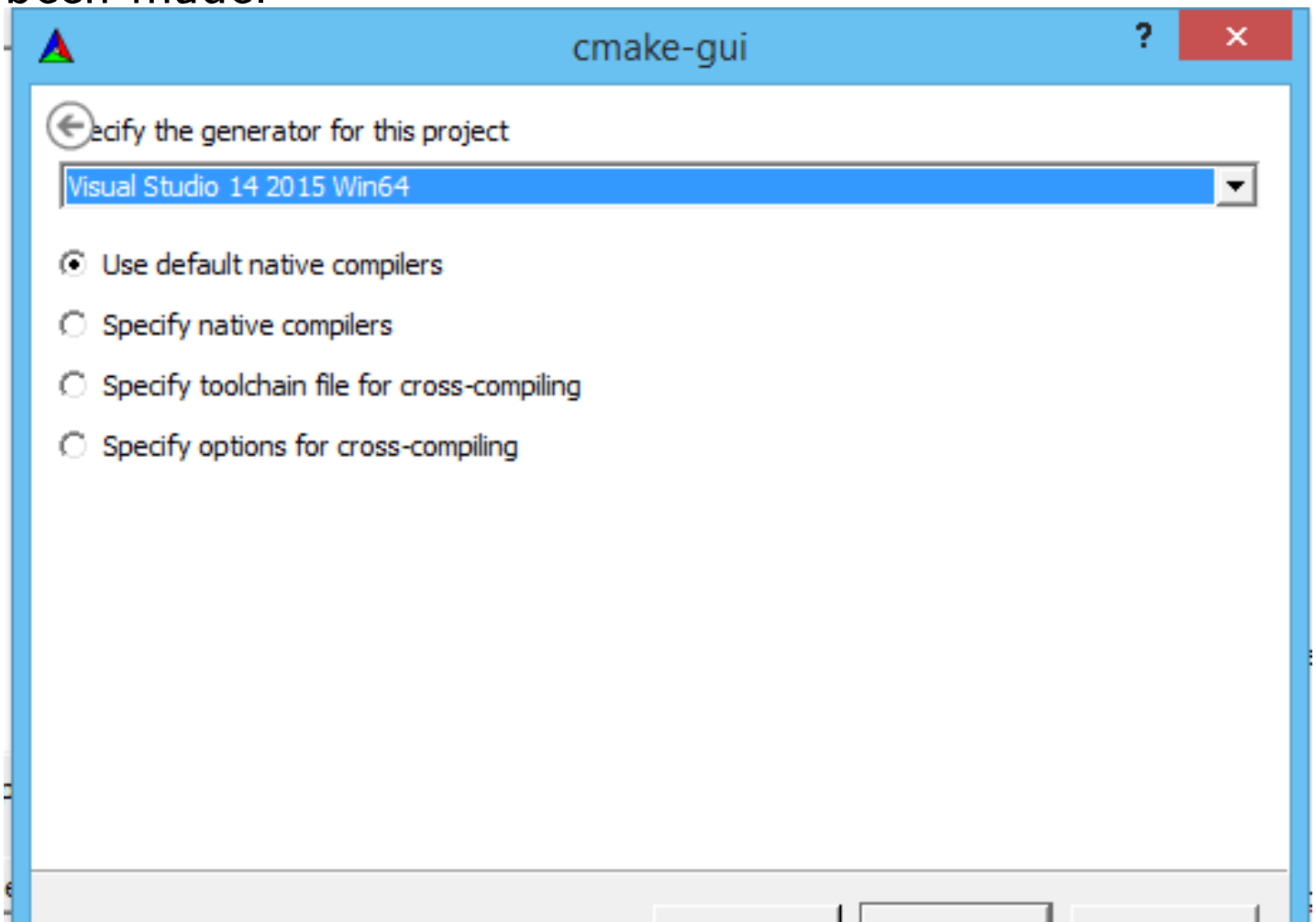
Windows building

In "Where to build the binaries", you can put the same folder, but we recommend you create a new folder "build" to hold the binaries. Once this is done, press "Configure"



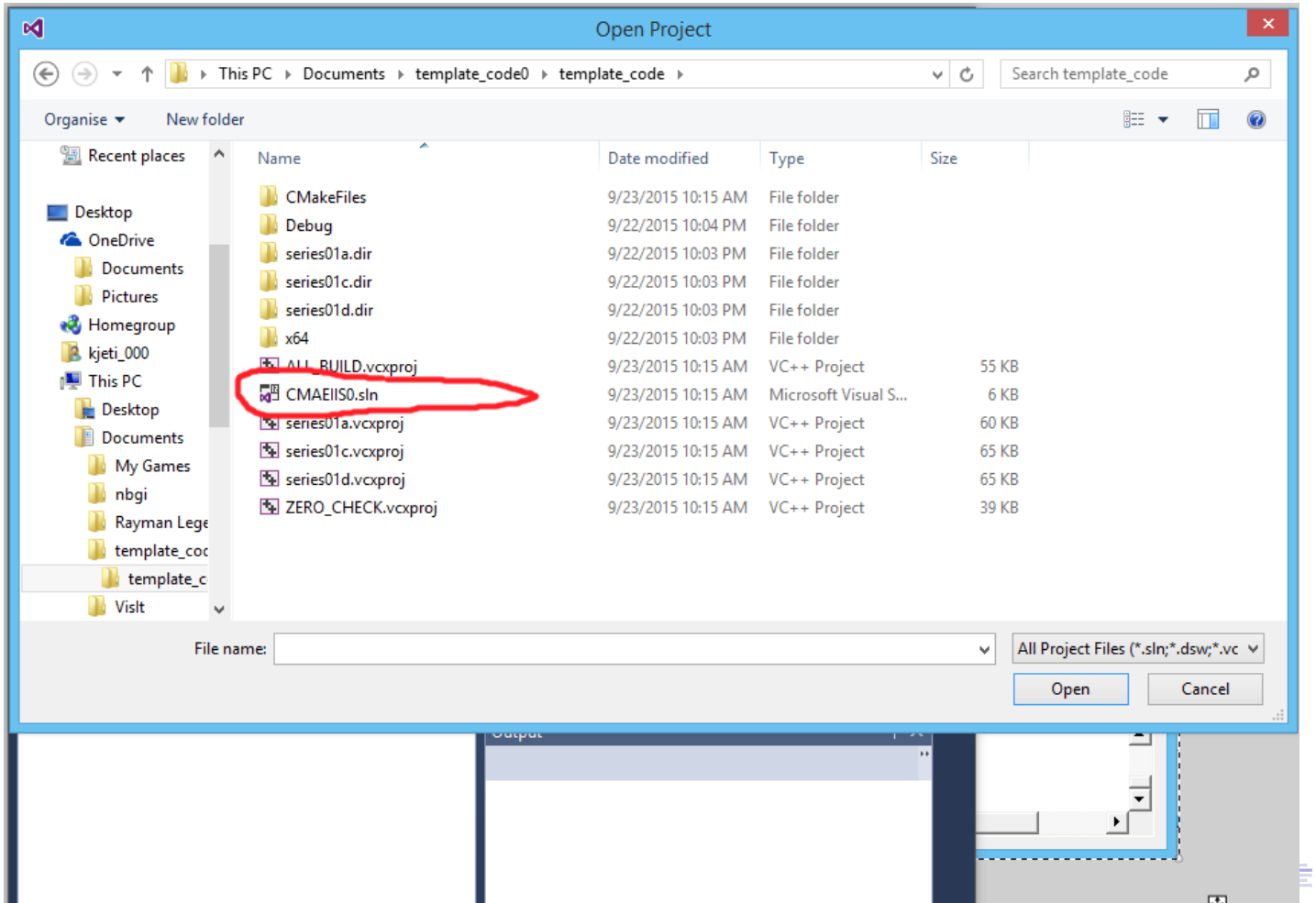
Windows building

You will now get up the following image. Choose the option "Visual Studio 14 2015 Win64" (Adjust accordingly if you use another version). At the end, select "Finish". If there were no errors, press "Generate". A new solution file (.sln) should have been made.



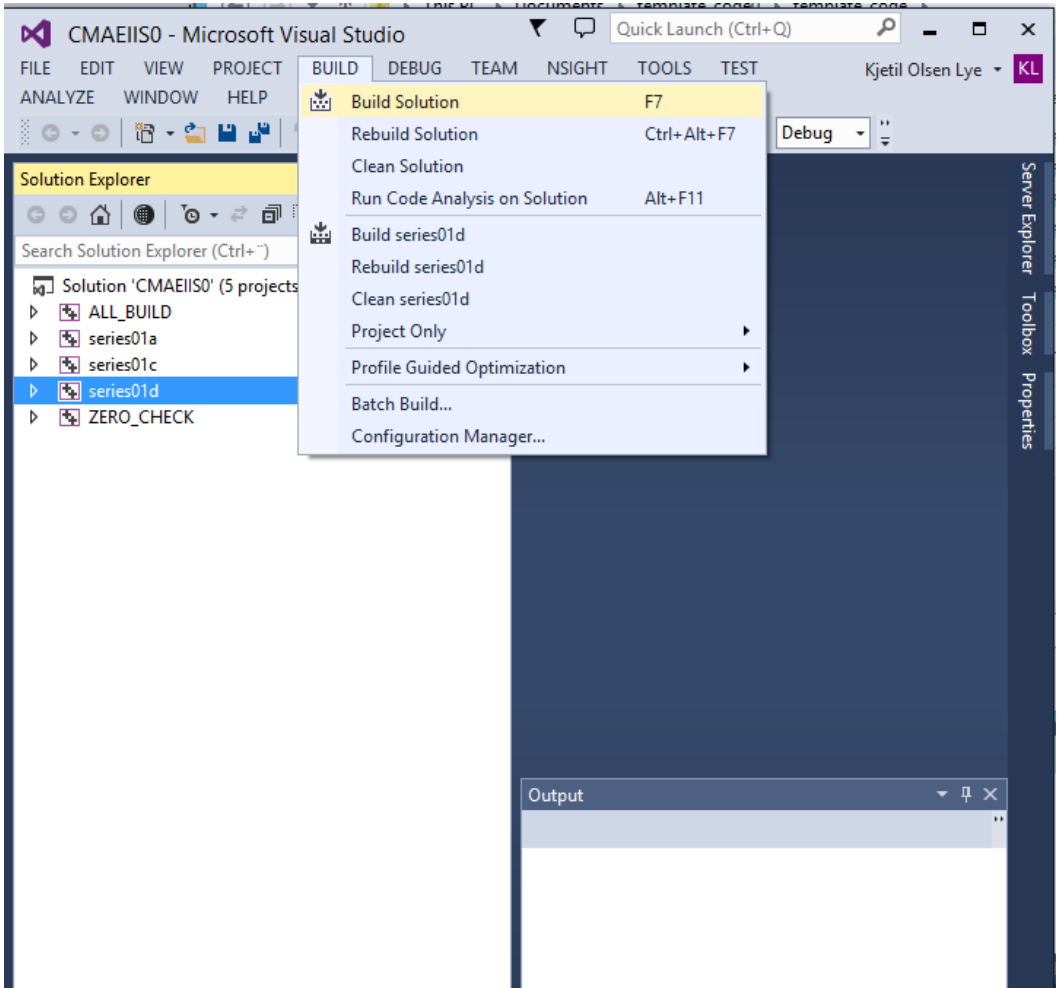
Windows building

Now open Microsoft Visual Studio and select "File" > "Open" > "Project/Solution". Go to the build folder you specified to CMake, and there should be a Solution(.sln) file there, choose this file. (In this course, the solution file will always be CMEAIISx, where x is the series number).



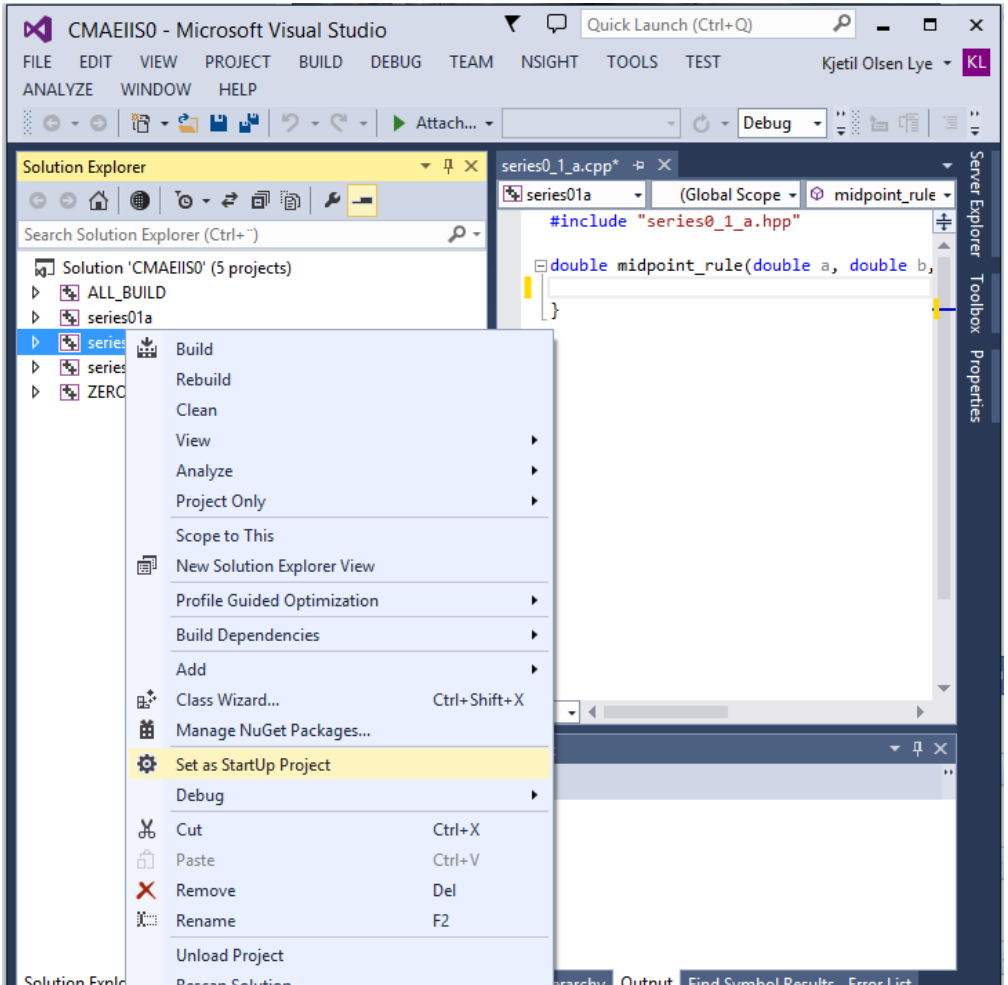
Windows building

You can build the whole solution by going to "Build" > "Build Solution" (You can also press F7 on most systems). IMPORTANT: The template code will not build out of the box since it is missing code. If the compiler complains about missing return statements, you can temporarily add `return 0;` and fix it later.



Windows building

Once the solution is build (should be less than half a minute), you can start the projects. If the exercise involves a `main`-file, you can run the project by Right clicking on the relevant project, and select "Set as startup project". Once this is done, you press Ctrl + F5 to run (Or go to "Debug" > "Start without Debugging")



Plotting the results

- It is possible to plot directly from C++
- However: This is not recommended.
- Good: Separate simulation code and visualization code
- We will use external tools for plotting (either Python or MATLAB).
- Basic plotting procedure:
  - Run C++ program
  - In C++ program, write data to file
  - Load data from file in visualization program
  - Plot the data
- In the templates, we supply the function `writeToFile` (in `writer.hpp`), which will output a text file readable by both MATLAB and Python.

Simple example: C++ program to compute sine

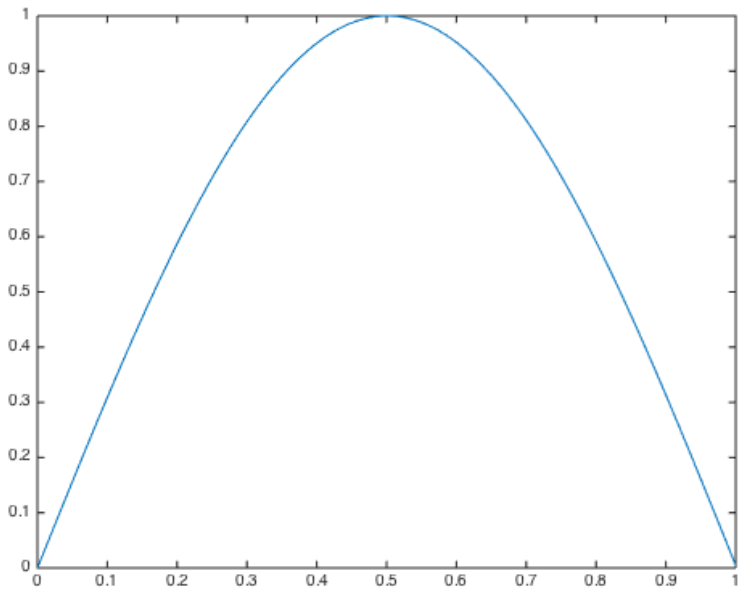
Suppose we compile and run the following program in C++

```
#include "writer.hpp" // made for CMEA II
#include <math.h>

int main() {
    std::vector<double> sineValues(1000);

    for(int i = 0; i < sineValues.size(); ++i) {
        sineValues[i] = sin(3.14 * i / 1000.0);
    }

    writeToFile("sine_values.txt", sineValues);
}
```

Plotting from MATLAB

We can plot the results in MATLAB with the following code (remember to have MATLAB in the same folder as the data!)

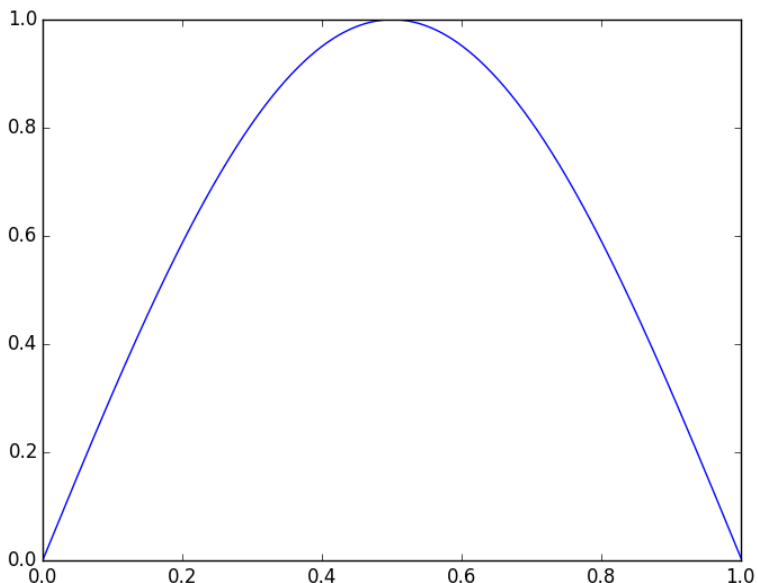
```
% Load data into a vector
data = load('sine_values.txt');
% Now we can do anything to the data
plot(linspace(0, 1, length(data)), data)
```



Plotting from Python

We can plot the results in Python with almost the same code (remember to have Python in the same folder as the data!)

```
import numpy, pylab

sine = numpy.loadtxt('sine_values.txt')
x = numpy.linspace(0, 1, len(sine))

pylab.plot(x, sine)
pylab.show()
```



Some final words

- C++ is great, but complex
- Do not start the assignment the day before!
- You will run into problems with compiling and running
- Use the teaching assistants for help with this!

Last hour will be in the ordinary tutorial rooms

Teaching assistant	**Room**
Michel Breyer	HG D 7.2
Carlo Del Don	HG E 22
Kjetil Lye	HG F 26.5
Laura Scarabosio	HG G 26.5
Alessio Zanchettin	HG E 33.1