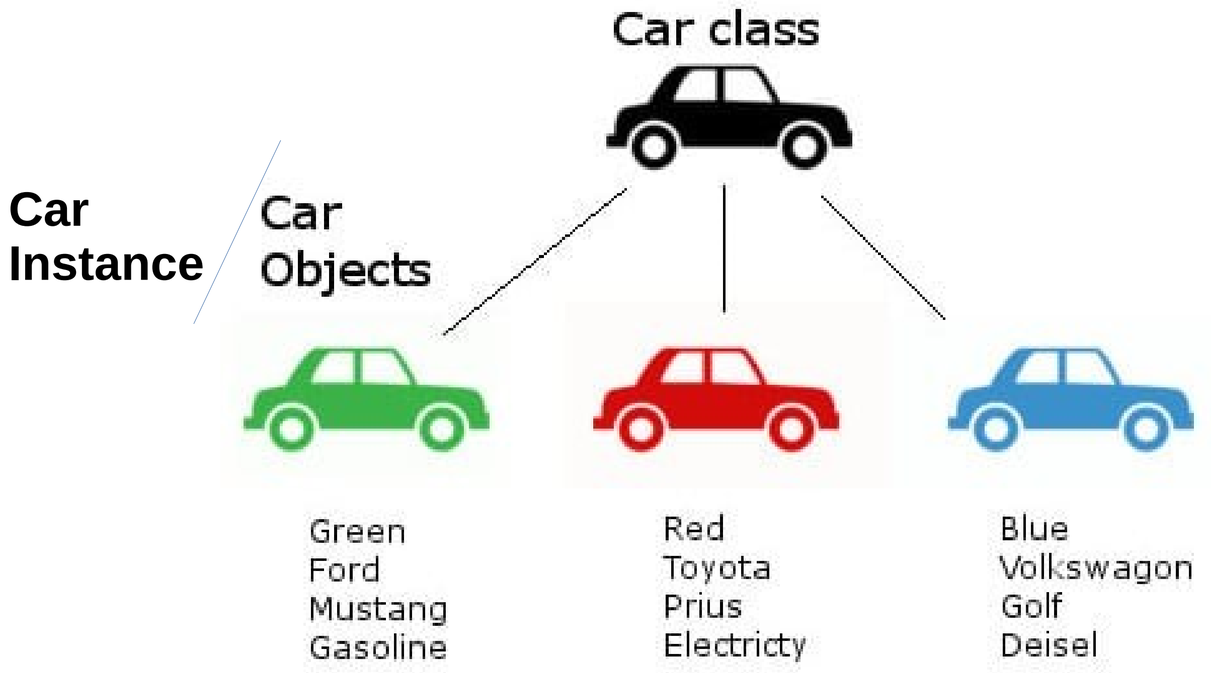
External libraries

Objectoriented Programing in C++

Thorbjørn Mosekjær Iversen
thmi@mmmi.sdu.dk

Sidste uge

- Klasser
- IDE’er



Denne uge

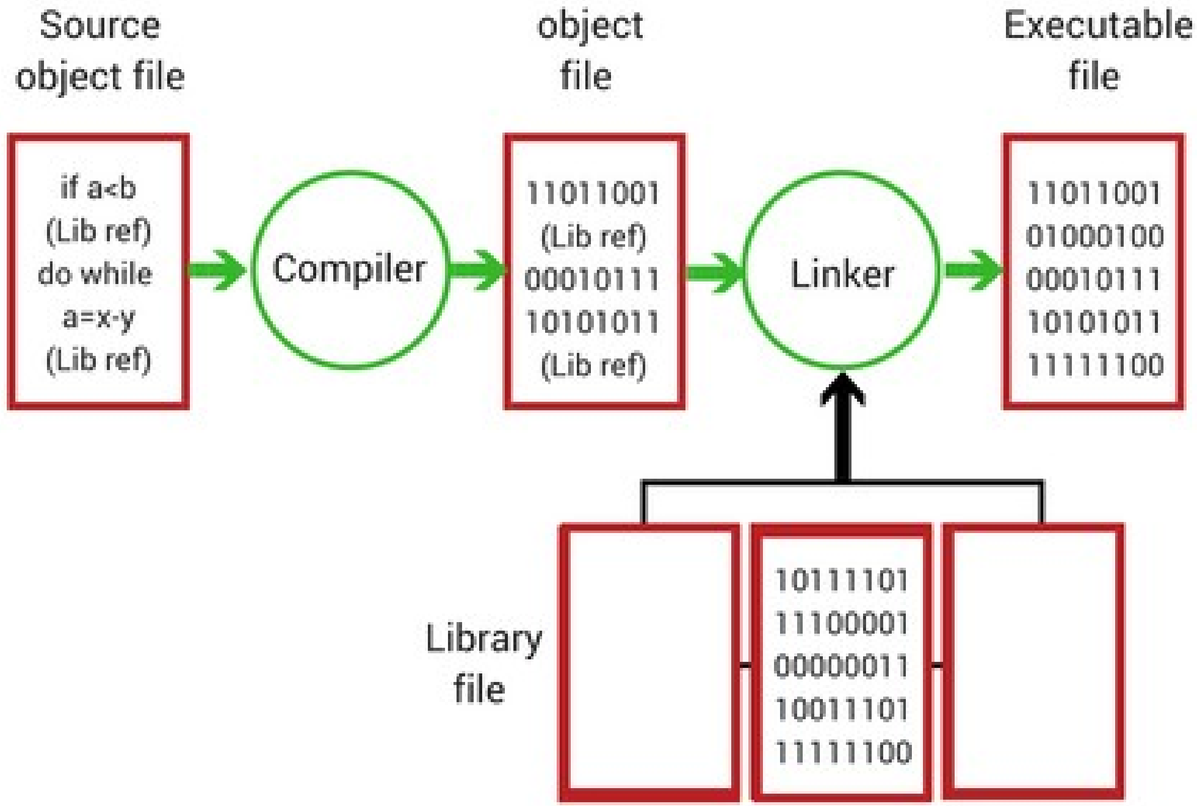
- Opdeling af kode i header og source filer
  - Preprocessor directives
- Including og linking til eksterne biblioteker

SDU
UNIVERSITY OF
SOUTHERN DENMARK

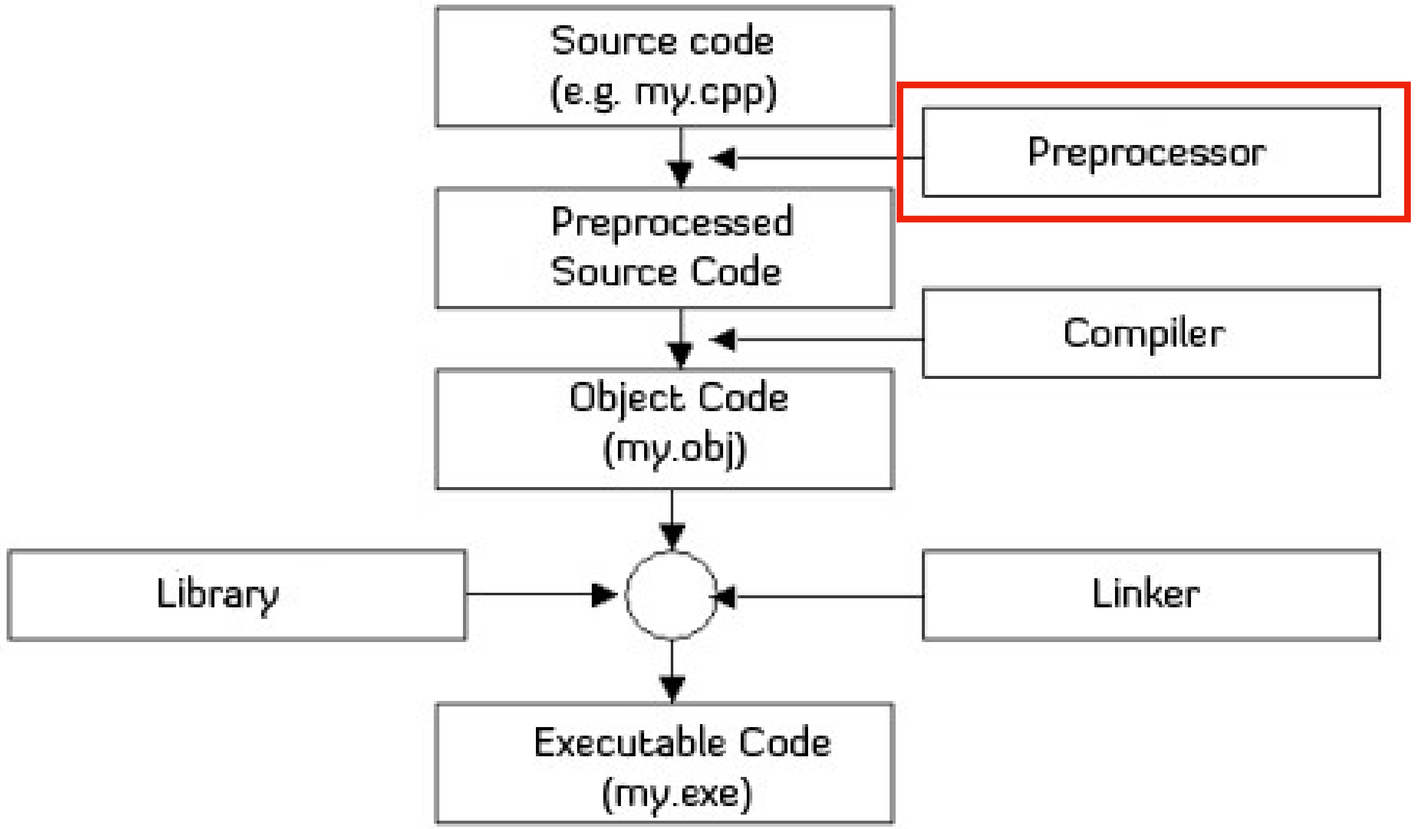
Preprocessor directives

(and splitting a class into multiple files)

Compiling (and linking)



The Pre-processor (per-compiler)



Splitting a Class into different files

main.cpp Compile: g++ main.cpp

```cpp
#include <iostream>

/* Class declaration and definition */
class MyClass{
private:
    int _myInt;
public:
    MyClass(int myInt):_myInt(myInt)
    {
    }
    int getValue(){
        return _myInt;
    }
};

/* Main method */
int main()
{
    MyClass myObject(7);
    std::cout << myObject.getValue() << std::endl;
    return 0;
}
```

main.cpp Compile: g++ main.cpp

```cpp
#include <iostream>

/* Class declaration */
class MyClass{
private:
    int _myInt;
public:
    MyClass(int _myInt);
    int getValue();
};

/* Class definitions */
MyClass::MyClass(int myInt):_myInt(myInt)
{
}

int MyClass::getValue()
{
    return _myInt;
}

/* Main method */
int main()
{
    MyClass myObject(7);
    std::cout << myObject.getValue() << std::endl;
    return 0;
}
```

main.cpp Compile: g++ main.cpp myClass.h myClass.cpp

```cpp
#include <iostream>
#include "myClass.h"

/* Main method */
int main()
{
    MyClass myObject(7);
    std::cout << myObject.getValue() << std::endl;
    return 0;
}
```

myClass.h

```cpp
#ifndef MY_CLASS_H
#define MY_CLASS_H

/* Class declaration */
class MyClass{
private:
    int _myInt;
public:
    MyClass(int _myInt);
    int getValue();
};

#endif
```

myClass.cpp

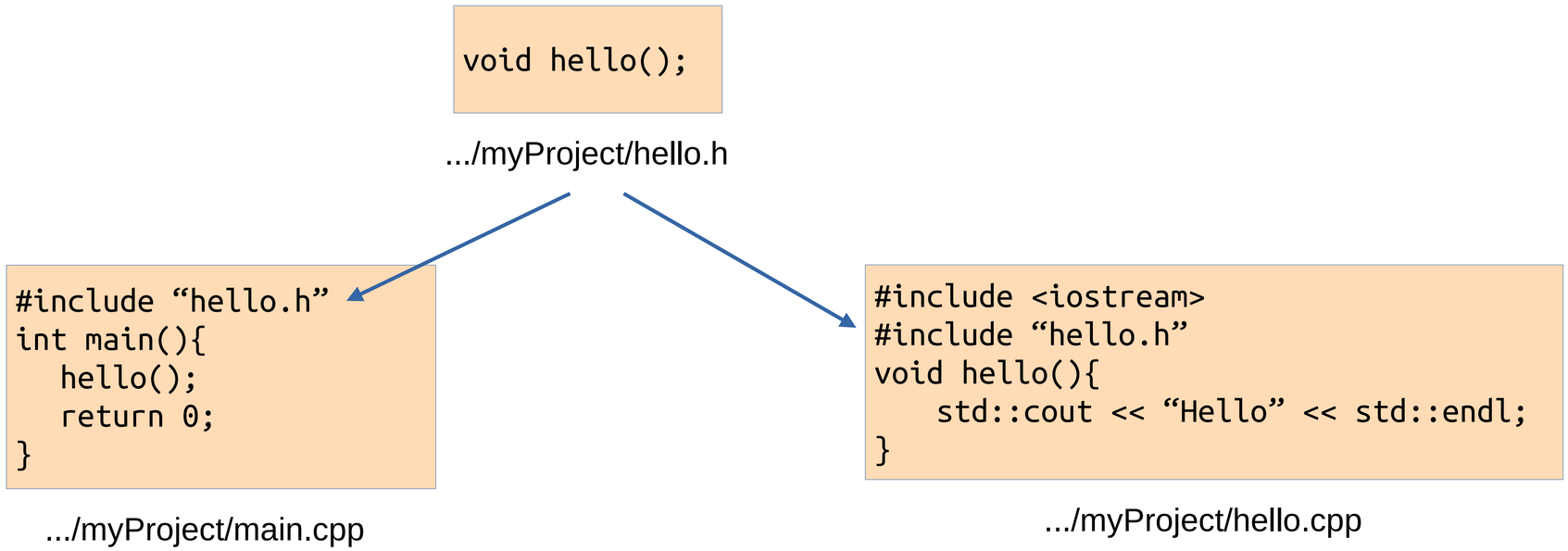
```cpp
#include "myClass.h"

/* Class definitions */
MyClass::MyClass(int myInt):_myInt(myInt)
{
}

int MyClass::getValue()
{
    return _myInt;
}
```

The Pre-processor

- Preprocessor directives are parsed before actually compiling source code.
- Example: #include “functions.h” will copy the contents of the referenced header file.

Header and source files (.h/.hpp and .cpp/.cc)

- Header files are used mainly to declare symbols.
  - Remember: all functions must be declared before they can be used
- Source files are used to define symbols.



Include guard: ifndef - define - endif

- A header file should contain an “include guard” to avoid multiple declarations of the same symbols in a cyclic inclusion relationship.

```
// Author: me
// License: Pay me $100 every time you read this

#ifndef HERBERTS_MATH_FUNCTIONS_INCLUDE
#define HERBERTS_MATH_FUNCTIONS_INCLUDE

#include <cmath>

double sine(double x);
…

# endif // HERBERTS_MATH_FUNCTIONS_INCLUDE
```

Include guard: pragma once

- Not part of standard, but widely supported

```
// Author: me
// License: Pay me $100 every time you read this

#pragma once

#include <cmath>

double sine(double x);
…
```

Predefined preprocessor definitions

```
#ifdef _MSC_VER
    … Windows code
#else
    … Linux/Unix code
#endif
```

Splitting a Class into different files

main.cpp Compile: g++ main.cpp

```cpp
#include <iostream>

/* Class declaration and definition */
class MyClass{
private:
    int _myInt;
public:
    MyClass(int myInt):_myInt(myInt)
    {
    }
    int getValue(){
        return _myInt;
    }
};

/* Main method */
int main()
{
    MyClass myObject(7);
    std::cout << myObject.getValue() << std::endl;
    return 0;
}
```

main.cpp Compile: g++ main.cpp

```cpp
#include <iostream>

/* Class declaration */
class MyClass{
private:
    int _myInt;
public:
    MyClass(int _myInt);
    int getValue();
};

/* Class definitions */
MyClass::MyClass(int myInt):_myInt(myInt)
{
}

int MyClass::getValue()
{
    return _myInt;
}

/* Main method */
int main()
{
    MyClass myObject(7);
    std::cout << myObject.getValue() << std::endl;
    return 0;
}
```

main.cpp Compile: g++ main.cpp myClass.h myClass.cpp

```cpp
#include <iostream>
#include "myClass.h"

/* Main method */
int main()
{
    MyClass myObject(7);
    std::cout << myObject.getValue() << std::endl;
    return 0;
}
```

myClass.h

```cpp
#ifndef MY_CLASS_H
#define MY_CLASS_H

/* Class declaration */
class MyClass{
private:
    int _myInt;
public:
    MyClass(int _myInt);
    int getValue();
};

#endif
```

myClass.cpp

```cpp
#include "myClass.h"

/* Class definitions */
MyClass::MyClass(int myInt):_myInt(myInt)
{
}

int MyClass::getValue()
{
    return _myInt;
}
```

SDU
UNIVERSITY OF
SOUTHERN DENMARK

External libraries
(including and linking)

Allerførst: Hvordan downloader og compiler jeg libraries

- Mange opensource biblioteker kan hentes gennem Ubuntu apt
  - e.g. `sudo apt install libmodbus-dev`
- Mange andre opensource biblioteker ligger på github
  - e.g. `git clone https://github.com/stephane/libmodbus`
  - Læs dokumentation for det givne bibliotek for at se hvordan man compiler det

Hvordan linker man?

- Læs dokumentationen for det library man linker til!
  - e.g. libmodbus

https://libmodbus.org/getting_started/

Code Sample

hello.c

```c
#include <stdio.h>
#include <modbus.h>

int main(void) {
  modbus_t *mb;
  uint16_t tab_reg[32];

  mb = modbus_new_tcp("127.0.0.1", 1502);
  modbus_connect(mb);

  /* Read 5 registers from the address 0 */
  modbus_read_registers(mb, 0, 5, tab_reg);

  modbus_close(mb);
  modbus_free(mb);
}
```

To compile this snippet, you can pass the header and library parameters to your C compiler (`gcc` , `llvm` , ...) with `pkg-config --cflags --libs libmodbus` :

```
cc hello.c `pkg-config --cflags --libs libmodbus`
```

Hvordan linker man?

- Det er her magien finder sted:

https://libmodbus.org/getting_started/

Code Sample

hello.c

```c
#include <stdio.h>
#include <modbus.h>

int main(void) {
  modbus_t *mb;
  uint16_t tab_reg[32];

  mb = modbus_new_tcp("127.0.0.1", 1502);
  modbus_connect(mb);

  /* Read 5 registers from the address 0 */
  modbus_read_registers(mb, 0, 5, tab_reg);

  modbus_close(mb);
  modbus_free(mb);
}
```

To compile this snippet, you can pass the header and library parameters to your C compiler (`gcc` , `llvm` , ...) with `pkg-config --cflags --libs libmodbus` :

```
cc hello.c `pkg-config --cflags --libs libmodbus`
```

```c
#include <modbus.h>
```

```
cc hello.c `pkg-config --cflags --libs libmodbus`
```

Eksterne libraries

Hvad skal inkluderes for at ens kode kan arbejde sammen med andre biblioteker?

- Declarations

  Information om hvilke funktioner, klasser, etc. der er tilgængelige og hvordan de benyttes
- Definition

  Implementationen af ovenstående funktioner, klasser, etc. ()

Eksterne libraries

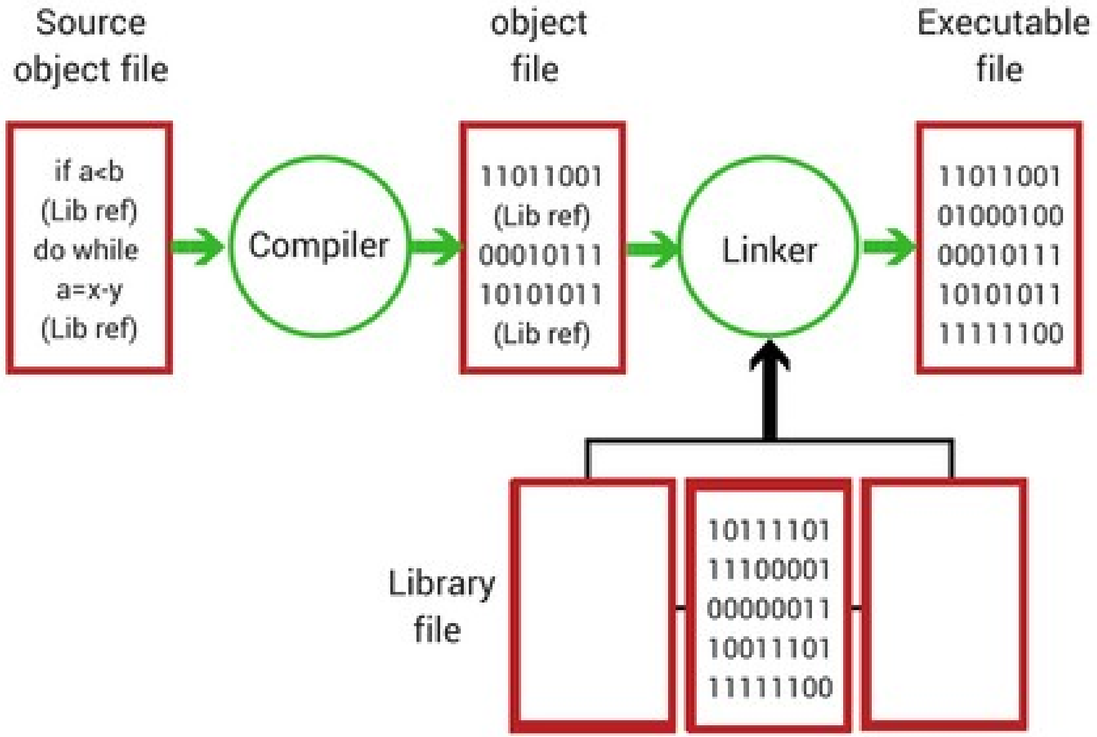
Hvad skal inkluderes for at ens kode kan arbejde sammen med andre biblioteker?

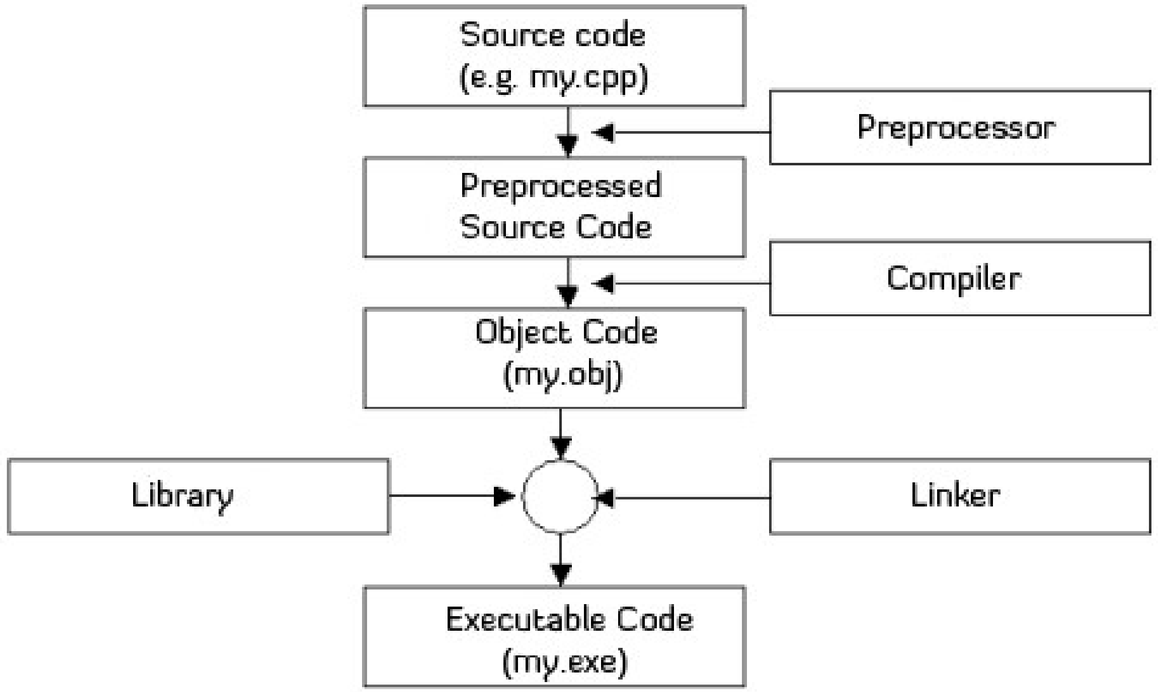
- Declarations (**header filerne (*.h)**)

  Information om hvilke funktioner, klasser, etc. der er tilgængelige og hvordan de benyttes
- Definition (**library filerne (lib*.so)**)

  Implementationen af ovenstående funktioner, klasser, etc. ()

Eksterne libraries





Standard lokation for header og library filer

Som standard ligger header filerne I

- Headerfilerne: /usr/include/

  e.g.
  ```
  /usr/include/modbus/modbus-rtu.h
  /usr/include/modbus/modbus-tcp.h
  /usr/include/modbus/modbus-version.h
  /usr/include/modbus/modbus.h
  ```

- Libraryfilerne: /usr/lib/x86_64-linux-gnu/

  e.g.
  ```
  /usr/lib/x86_64-linux-gnu/libmodbus.so
  ```

Standard lokation for header og library filer

Som standard ligger header filerne I

- Headerfilerne: /usr/include/

  e.g.
  ```
  /usr/include/modbus/modbus-rtu.h
  /usr/include/modbus/modbus-tcp.h
  /usr/include/modbus/modbus-version.h
  /usr/include/modbus/modbus.h
  ```

- Libraryfilerne: /usr/lib/x86_64-linux-gnu/

  e.g.
  ```
  /usr/lib/x86_64-linux-gnu/libmodbus.so
  ```

Vi skal give compileren stien til den mappe som h-filerne ligger i, dvs.
/usr/include/modbus

Vi kan herefter include h-filerne vha. angle-brackets, e.g.

```
#include <modbus.h>
```

Vi skal fortælle compileren hvilke library filer den skal linke til dvs.

libmodbus.so

(Hvis library filerne ligger i en anden mappe den den der er standard skal vi også fortælle compileren dette.)

Compile med g++

- `g++ main.cpp -I/usr/include/modbus -lmodbus`
- Se manualen for en ubuntu commando: man g++
  - `-Idir`

    Add the directory dir to the list of directories to be searched for header files during preprocessing.

    e.g. **modbus.h** ligger i **/usr/include/modbus** så det er denne mappe der er include-directory. Vi kan herefter benytte **#include <modbus.h>** i vores c++ programmer.
  - `-llibrary`

    Search the library named library when linking. (The linker searches a standard list of directories for the library. The directories searched include several standard system directories plus any that you specify with -L.)

    e.g. modbus librariet hedder **modbus**. Dvs. library-filen der ligger på computeren hedder **lib**`modbus`**.so** og linkes til med `-lmodbus`
  - `-Ldir`

    Add directory dir to the list of directories to be searched for -l.

    e.g. **libmodbus.so** ligger i **/usr/lib/x86_64-linux-gnu/** hvilket er standard path for library files. Derfor behøver vi ikke give path-navnet med. Hvis vi vil gøre det eksplicit kan man tilføje **`-L/usr/lib/x86_64-linux-gnu/`** til compiler kaldet, dvs. `g++ main.cpp -I/usr/include/modbus -L/usr/lib/x86_64-linux-gnu/ -lmodbus`

Tilbage til modbus’ dokumentation

- Fra forrige slide:

```
g++ main.cpp -I/usr/include/modbus -lmodbus
```

- Fra modbus’ dokumentation:

```
cc hello.c `pkg-config --cflags --libs libmodbus`
```

pkg-config

- Lad os gå tilbage til modbus compile commandoen

```
cc hello.c `pkg-config --cflags --libs libmodbus`
```

compiler, e.g.

`g++`

En eller flere source filer, e.g.

`main.cpp matrix.cpp matrix.h`

Command substitution. Kan også laves vha. $(), e.g.

`$(pkg-config --cflags --libs libmodbus)`

pkg-config

Et commandline værktøj til at generere de nødvendige compiler flag for at compile og linke til installerede libraries

- e.g.

```
pkg-config --cflags libmodbus
```

- output: -I/usr/include/modbus

```
pkg-config --libs libmodbus
```

- output: -lmodbus

Compile vha. pkg-config

- For at opsummere

`compiler source-files compiler flags`

e.g.

`g++ main.cpp matrix.h matrix.cpp $(pkg-config --cflags --libs libmodbus)`

som i terminalen bliver lavet om til kaldet:

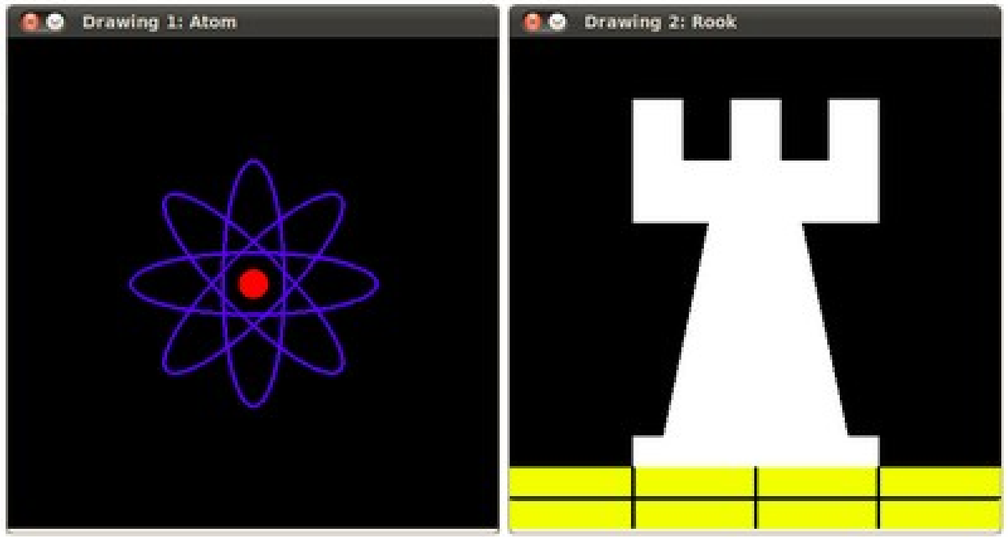
- `g++ main.cpp matrix.h matrix.cpp -I/usr/include/modbus -lmodbus`

Eksempel: OpenCV

OpenCV er et computer vision library der har mange funktioner til image processing. Det giver blandet andet muligheden for at tegne simpel grafik.

Undgå at compile OpenCV selv. Brug i stedet: **sudo apt install libopencv-dev** og følg tutorialen:

https://docs.opencv.org/4.x/d3/d96/tutorial_basic_geometric_drawing.html



OpenCV’s hello world

```
#include <opencv2/core.hpp>
#include <opencv2/highgui.hpp>

int main() {
    cv::Mat src = cv::imread("/home/myUsername/Documents/testImg.png");
    cv::imshow("MyImage",src);
    cv::waitKey(0);
}
```

Eksempel OpenCV

Til øvelserne i dag skal i linke til OpenCV:

Man kan gøre det manuelt:

```
g++ main.cpp -I/usr/include/opencv4/ -lopencv_highgui -lopencv_imgcodecs -lopencv_core
```

eller benytte pkg-config

```
g++ main.cpp $(pkg-config --cflags --libs opencv4)
```

som i terminalet bliver lavet om til kaldet:

```
g++ main.cpp -I/usr/include/opencv4/opencv -I/usr/include/opencv4 -lopencv_stitching -lopencv_aruco -lopencv_bgsegm -lopencv_bioinspired -lopencv_ccalib -lopencv_dnn_objdetect -lopencv_dnn_superres -lopencv_dpm -lopencv_highgui -lopencv_face -lopencv_freetype -lopencv_fuzzy -lopencv_hdf -lopencv_hfs -lopencv_img_hash -lopencv_line_descriptor -lopencv_quality -lopencv_reg -lopencv_rgbd -lopencv_saliency -lopencv_shape -lopencv_stereo -lopencv_structured_light -lopencv_phase_unwrapping -lopencv_superres -lopencv_optflow -lopencv_surface_matching -lopencv_tracking -lopencv_datasets -lopencv_text -lopencv_dnn -lopencv_plot -lopencv_ml -lopencv_videostab -lopencv_videoio -lopencv_viz -lopencv_ximgproc -lopencv_video -lopencv_xobjdetect -lopencv_objdetect -lopencv_calib3d -lopencv_imgcodecs -lopencv_features2d -lopencv_flann -lopencv_xphoto -lopencv_photo -lopencv_imgproc -lopencv_core
```

Eller benytte CMake ...

CMake

```
# This is a simple CMakeLists.txt file
cmake_minimum_required(VERSION 2.6 FATAL_ERROR)
Set(CMAKE_CXX_STANDARD 11)

project(myProject)

include_directories("/usr/include/opencv4/")
link_directories("/usr/lib/x86_64-linux-gnu/")
add_executable(myExe main.cpp)
target_link_libraries(myExe opencv_core opencv_highgui opencv_imgcodecs)
```


Obs: Automatically looks for lib`opencv_core`.so and calls the compiler with `-lopencv_core`

find_package

- OpenCV er configureret med CMake og kommer med en OpenCVConfig.cmake
  - Ligger I `/usr/lib/x86_64-linux-gnu/cmake/opencv4/`
  - Definerer automatisk Cmake variable der indeholder includes og libraries
    - `OpenCV_INCLUDE_DIRS`
    - `OpenCV_LIBS`
- Samme funktionalitet kan opnås ved at skrive sine egne `findXXX.cmake` filer som ligger I ens projekt

CMake

```
cmake_minimum_required(VERSION 2.8)
project( myProject )

find_package( OpenCV REQUIRED )

include_directories( ${OpenCV_INCLUDE_DIRS} )
add_executable( myExe DisplayImage.cpp )
target_link_libraries( myExe ${OpenCV_LIBS} )
```