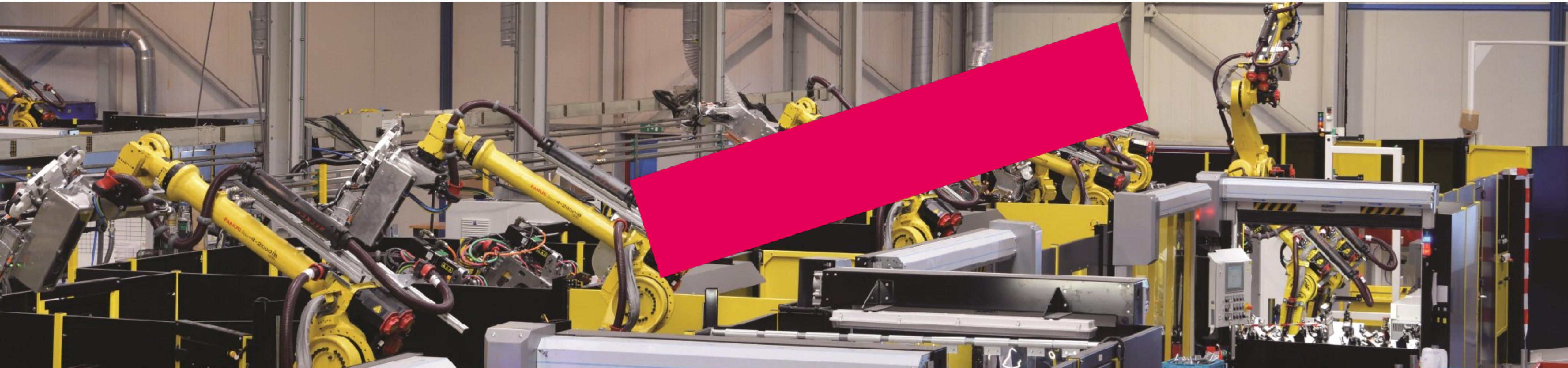


Programming 6



Workshop on CMake

johan.korten@han.nl

V1.0 Nov 2025

Workshop Topics overview

Clean code (beyond SOLID)

CMake

Unit testing

Commentaar

Patterns (Clean architecture)

Workshop Topics for Today

Build Nightmare

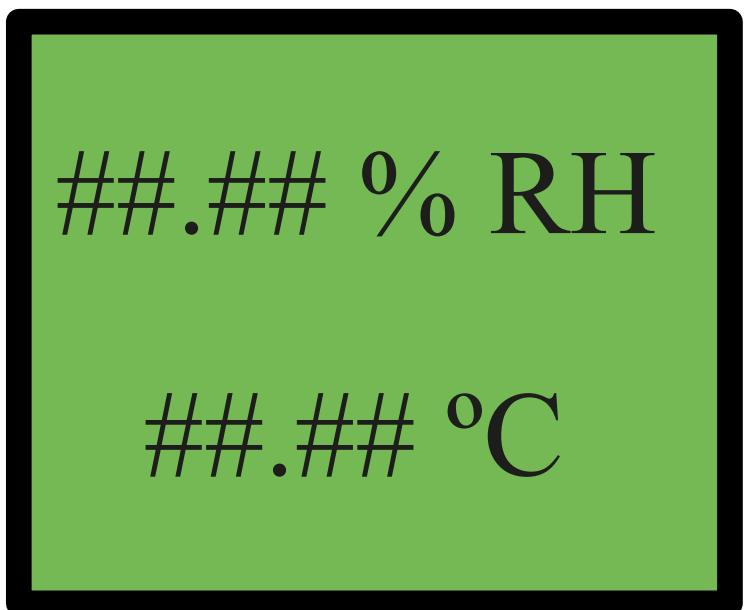
- Overcoming the Build Nightmare
- Cmake ins- and outs
- Cmake for Embedded targets
- Alternative to Make

Example project for today

Build Nightmare

SHT45

- Temperature
- Humidity Sensor

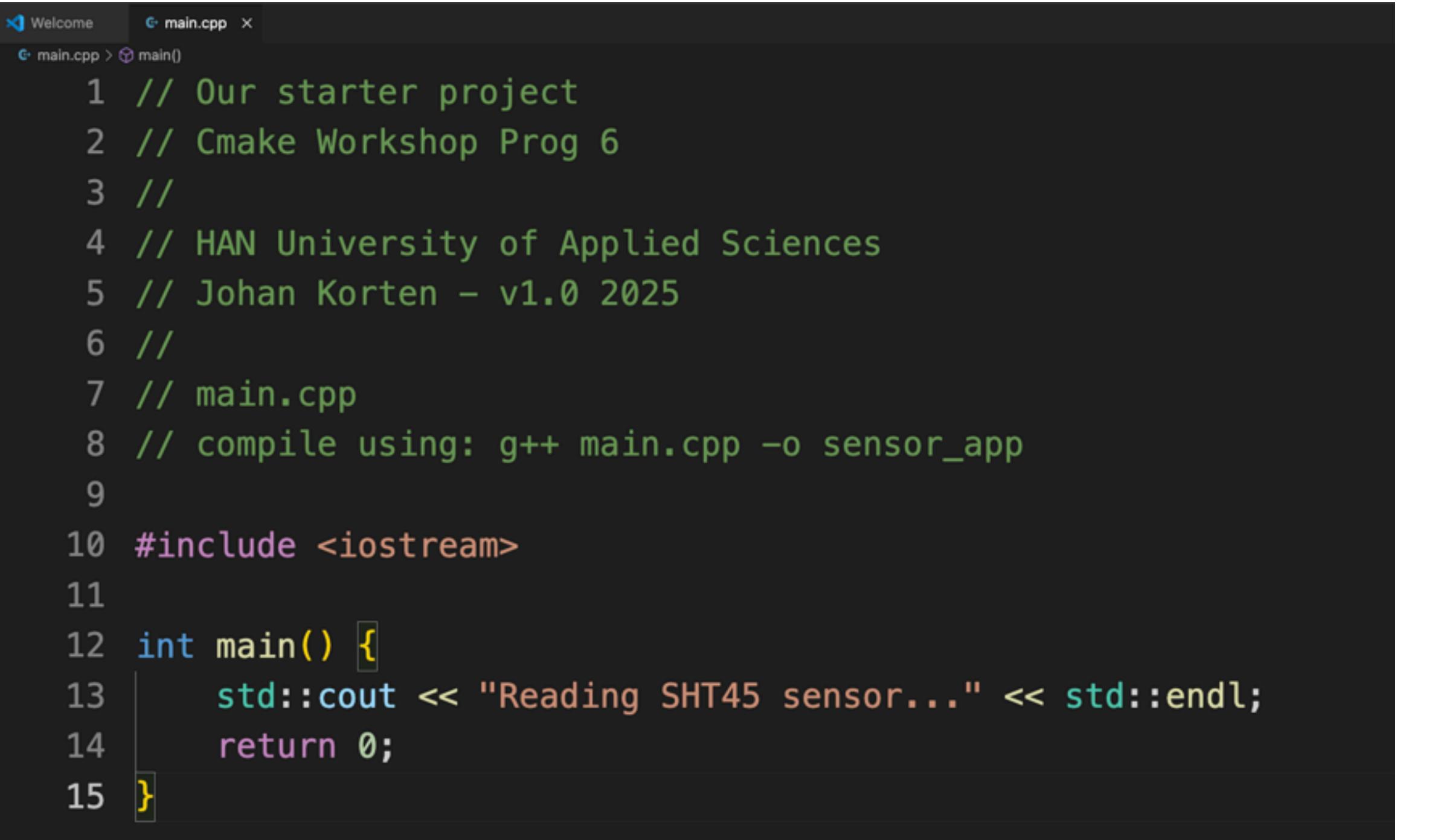


Lab
Environment
Monitor



Pragmatic start - Stepping stone 1

Build Nightmare

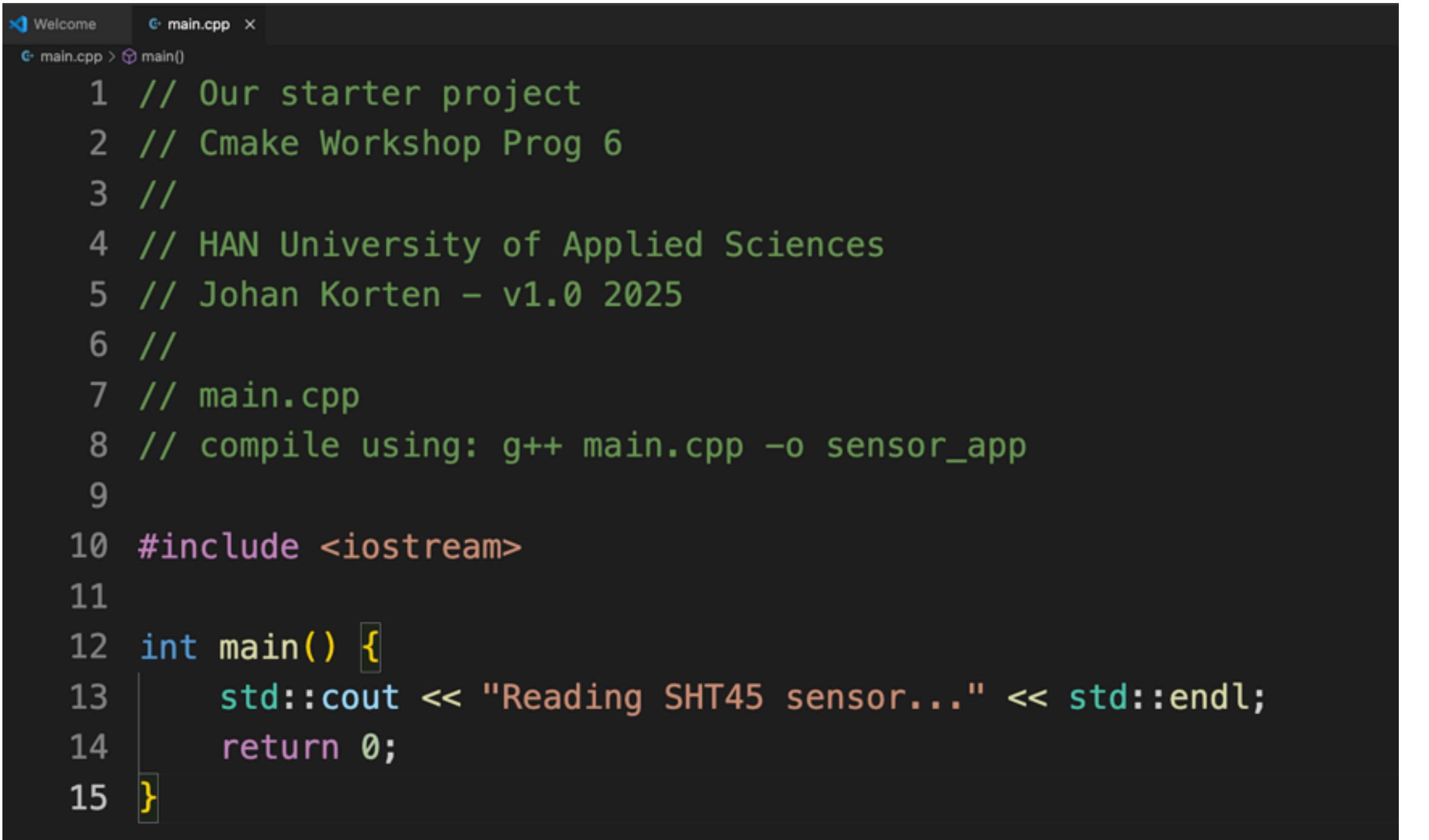


A screenshot of a code editor window titled "main.cpp". The code editor shows the content of the main.cpp file. The code is a simple C++ program with comments and a main function.

```
1 // Our starter project
2 // Cmake Workshop Prog 6
3 //
4 // HAN University of Applied Sciences
5 // Johan Korten - v1.0 2025
6 //
7 // main.cpp
8 // compile using: g++ main.cpp -o sensor_app
9
10 #include <iostream>
11
12 int main() {
13     std::cout << "Reading SHT45 sensor..." << std::endl;
14     return 0;
15 }
```

Pragmatic start - Stepping stone 1

Build Nightmare



```
>Welcome   main.cpp x
G· main.cpp > ⚙ main()

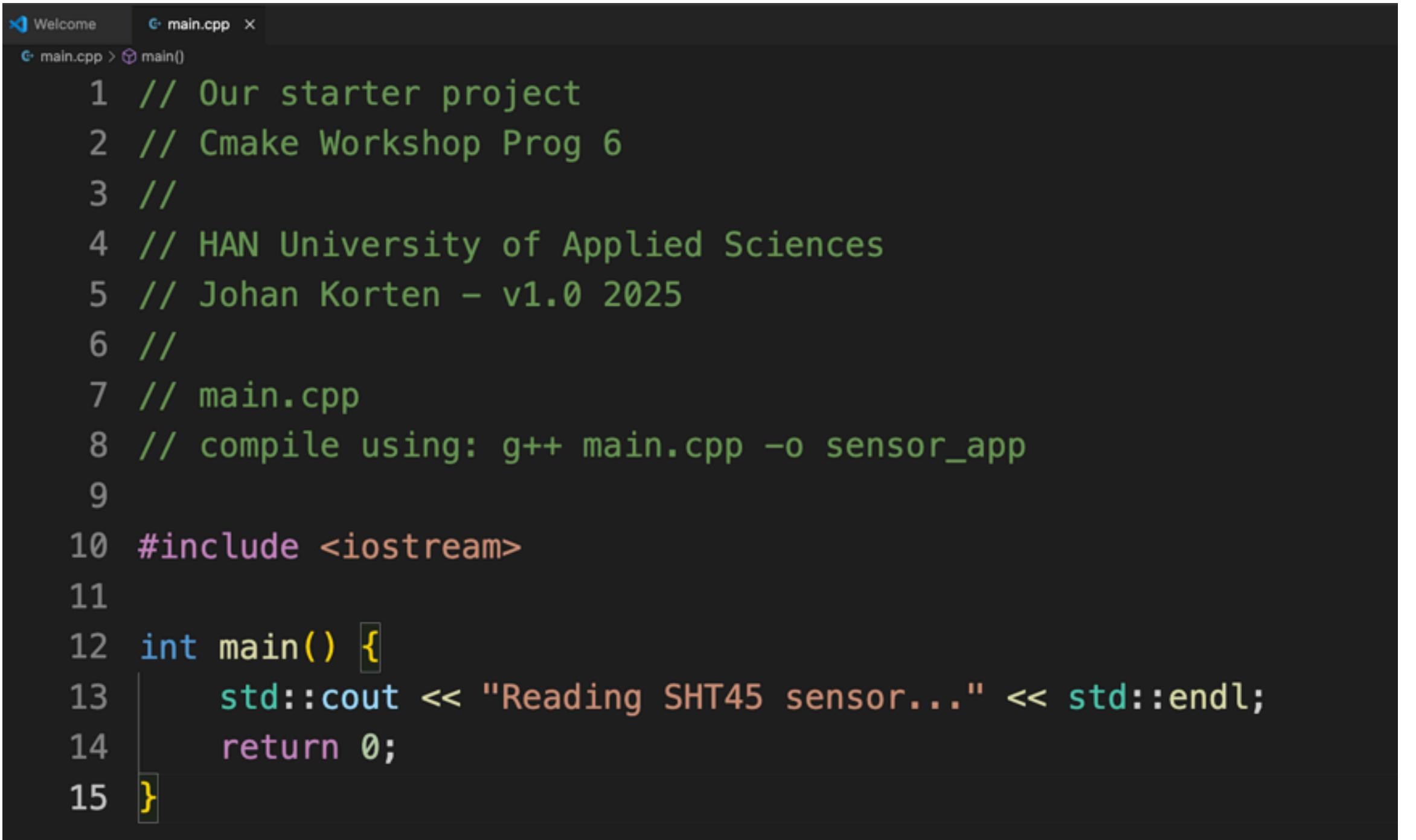
1 // Our starter project
2 // Cmake Workshop Prog 6
3 //
4 // HAN University of Applied Sciences
5 // Johan Korten - v1.0 2025
6 //
7 // main.cpp
8 // compile using: g++ main.cpp -o sensor_app
9
10 #include <iostream>
11
12 int main() {
13     std::cout << "Reading SHT45 sensor..." << std::endl;
14     return 0;
15 }
```

Compile and test:

```
jakorten@device-35 SimpleProject % g++ main.cpp -o sensor_app
jakorten@device-35 SimpleProject % ./sensor_app
Reading SHT45 sensor...
jakorten@device-35 SimpleProject %
```

Towards a library - Stepping stone 2

Build Nightmare



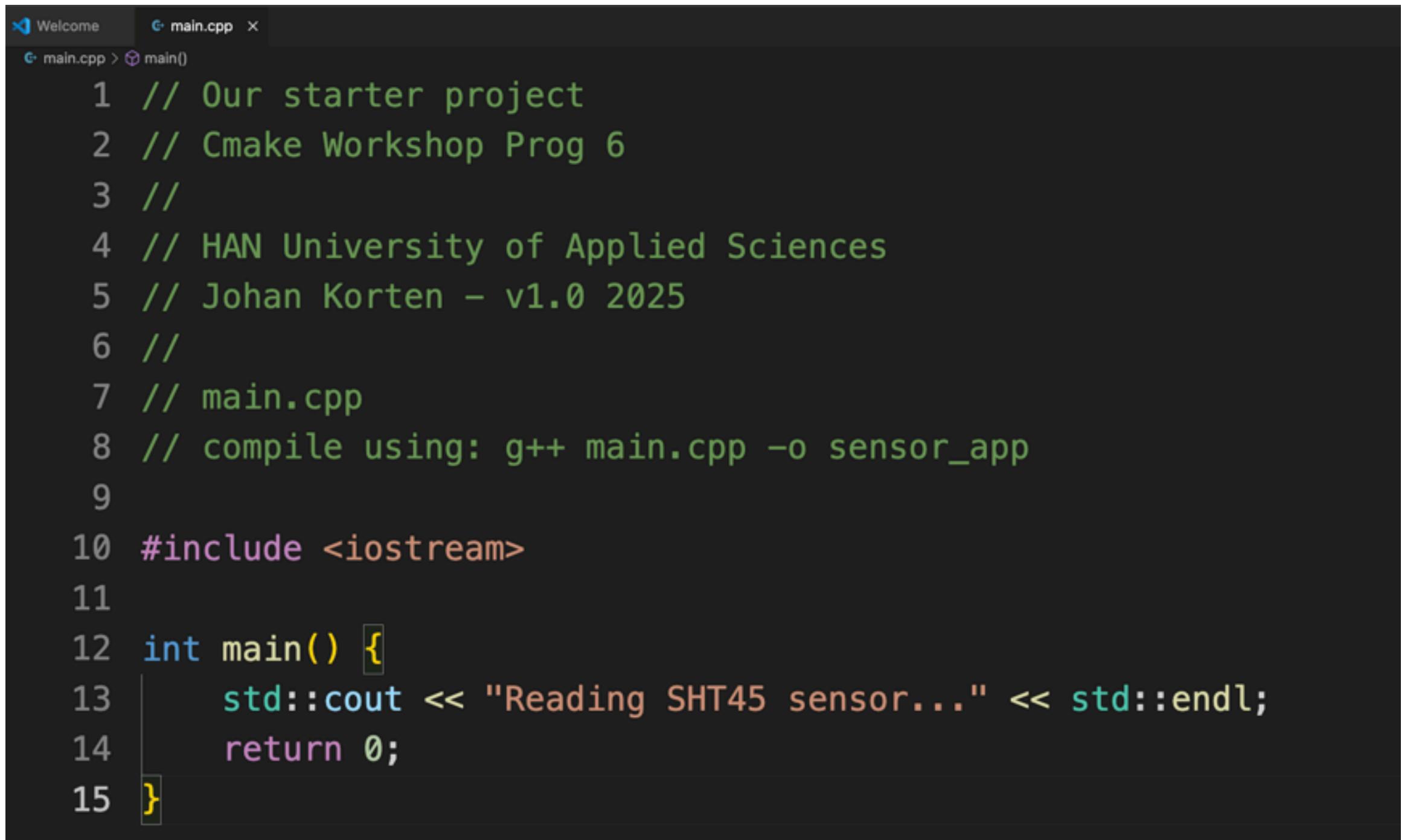
A screenshot of a code editor window titled "main.cpp". The code editor shows the following C++ code:

```
1 // Our starter project
2 // Cmake Workshop Prog 6
3 //
4 // HAN University of Applied Sciences
5 // Johan Korten - v1.0 2025
6 //
7 // main.cpp
8 // compile using: g++ main.cpp -o sensor_app
9
10 #include <iostream>
11
12 int main() {
13     std::cout << "Reading SHT45 sensor..." << std::endl;
14     return 0;
15 }
```

Compile and test:

Towards a library - Stepping stone 2

Build Nightmare



```
>Welcome  main.cpp x
G· main.cpp > ⚙ main()

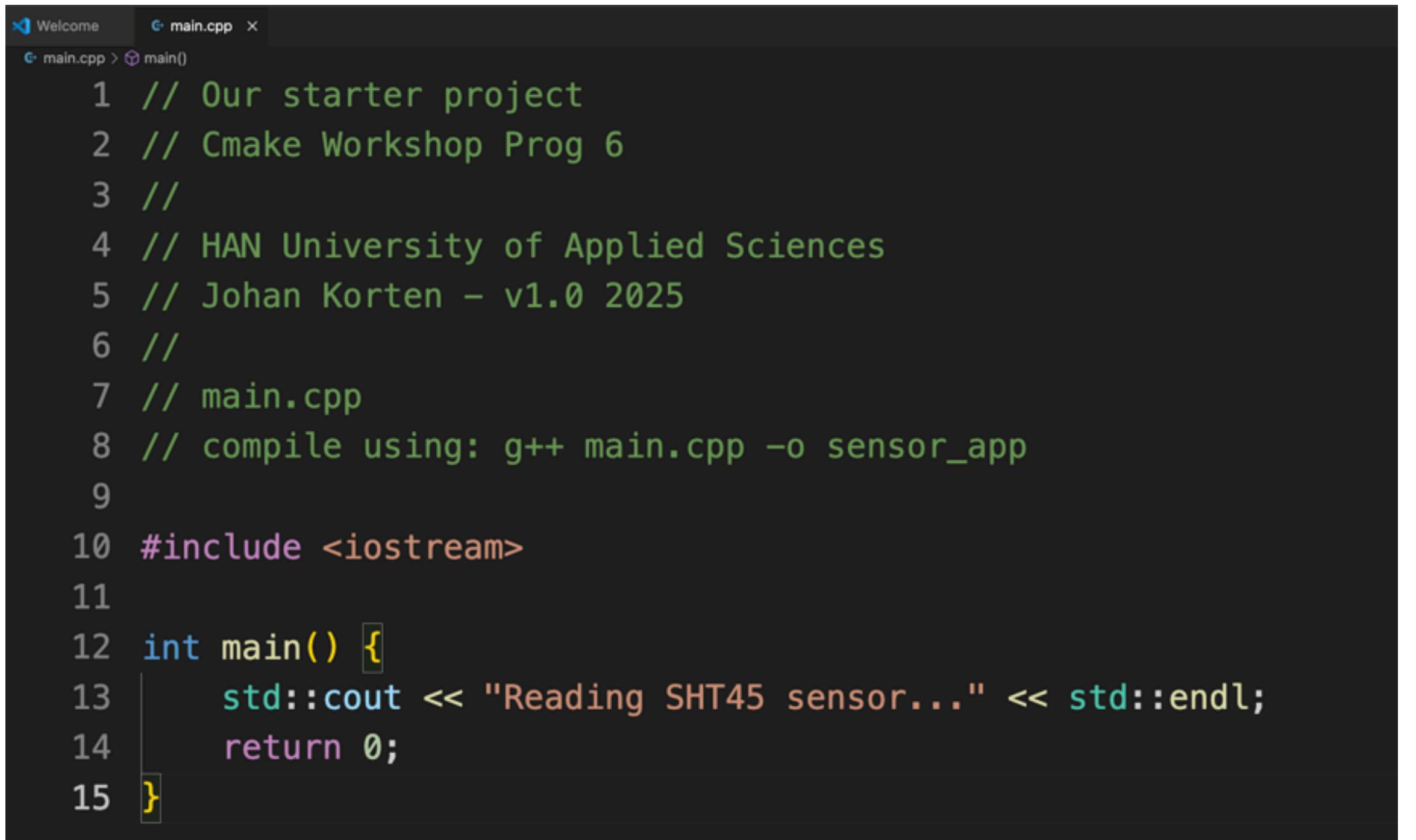
1 // Our starter project
2 // Cmake Workshop Prog 6
3 //
4 // HAN University of Applied Sciences
5 // Johan Korten - v1.0 2025
6 //
7 // main.cpp
8 // compile using: g++ main.cpp -o sensor_app
9
10 #include <iostream>
11
12 int main() {
13     std::cout << "Reading SHT45 sensor..." << std::endl;
14     return 0;
15 }
```

Compile and test: (Whoops...)

```
[jakorten@device-35 SimpleLibrary % g++ main.cpp -o sensor_app
Undefined symbols for architecture arm64:
  "SHT45::read_temperature()", referenced from:
    _main in main-ecf635.o
ld: symbol(s) not found for architecture arm64
clang++: error: linker command failed with exit code 1 (use -v to see invocation
)]
```

Towards a library - Stepping stone 2

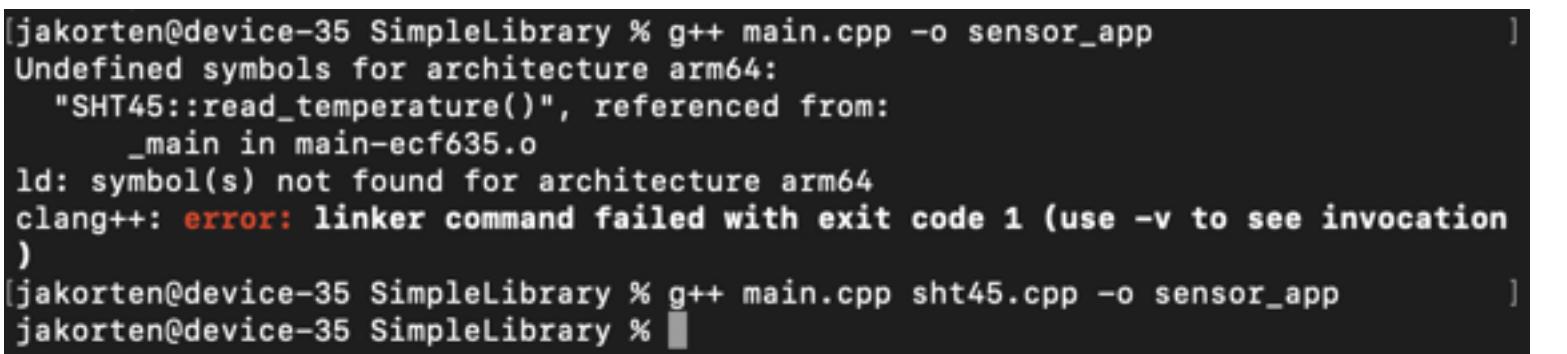
Build Nightmare



```
>Welcome  main.cpp x
G· main.cpp > ⚙ main()

1 // Our starter project
2 // Cmake Workshop Prog 6
3 //
4 // HAN University of Applied Sciences
5 // Johan Korten - v1.0 2025
6 //
7 // main.cpp
8 // compile using: g++ main.cpp -o sensor_app
9
10 #include <iostream>
11
12 int main() {
13     std::cout << "Reading SHT45 sensor..." << std::endl;
14     return 0;
15 }
```

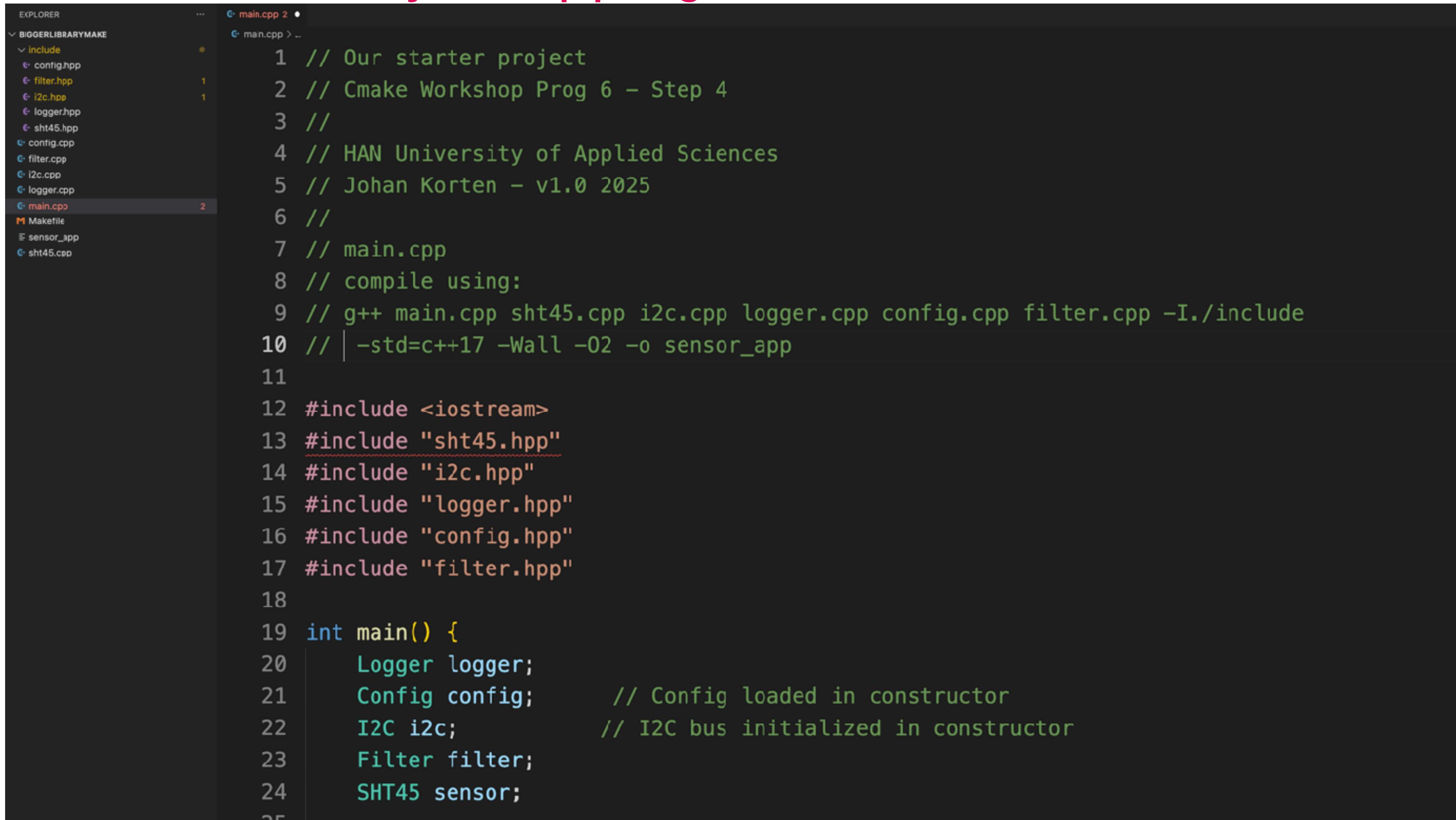
Compile and test: (Pfwiew, we got away for now...)



```
[jakorten@device-35 SimpleLibrary % g++ main.cpp -o sensor_app
Undefined symbols for architecture arm64:
  "SHT45::read_temperature()", referenced from:
    _main in main-ecf635.o
ld: symbol(s) not found for architecture arm64
clang++: error: linker command failed with exit code 1 (use -v to see invocation
)
[jakorten@device-35 SimpleLibrary % g++ main.cpp sht45.cpp -o sensor_app
[jakorten@device-35 SimpleLibrary % ]
```

Towards a library - Stepping stone 3

Build Nightmare

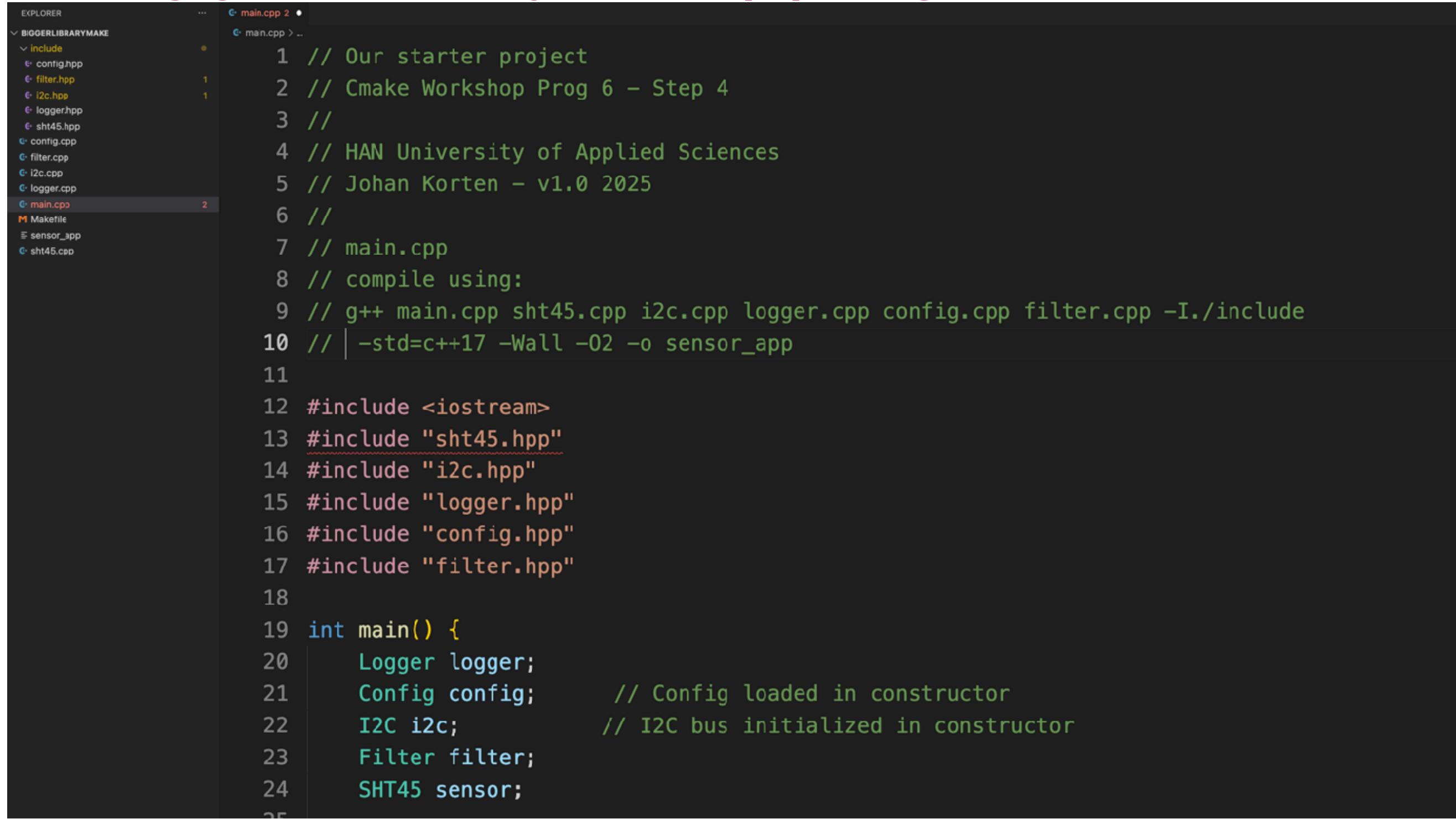


The screenshot shows a code editor with a dark theme. On the left is an Explorer sidebar showing a project structure under 'BIGGERLIBRARYMAKE'. The 'include' folder contains files: config.hpp, filter.hpp, i2c.hpp, logger.hpp, and sht45.hpp. The 'src' folder contains files: config.cpp, filter.cpp, i2c.cpp, logger.cpp, and main.cpp. A 'Makefile' file is also present. The main editor area displays the content of main.cpp. The code is a C++ program with the following content:

```
1 // Our starter project
2 // Cmake Workshop Prog 6 – Step 4
3 //
4 // HAN University of Applied Sciences
5 // Johan Korten – v1.0 2025
6 //
7 // main.cpp
8 // compile using:
9 // g++ main.cpp sht45.cpp i2c.cpp logger.cpp config.cpp filter.cpp -I./include
10 // | -std=c++17 -Wall -O2 -o sensor_app
11
12 #include <iostream>
13 #include "sht45.hpp"
14 #include "i2c.hpp"
15 #include "logger.hpp"
16 #include "config.hpp"
17 #include "filter.hpp"
18
19 int main() {
20     Logger logger;
21     Config config;      // Config loaded in constructor
22     I2C i2c;           // I2C bus initialized in constructor
23     Filter filter;
24     SHT45 sensor;
25 }
```

A bigger library - Stepping stone 4

Build Nightmare



The screenshot shows a code editor with a dark theme. On the left is the Explorer sidebar showing a project structure named 'BIGGERLIBRARYMAKE' with subfolders 'include' containing files like config.hpp, filter.hpp, i2c.hpp, logger.hpp, and sht45.hpp, and source files config.cpp, filter.cpp, i2c.cpp, logger.cpp, main.cpp, Makefile, sensor_app, and sht45.cpp. The main editor area displays the content of main.cpp:

```
1 // Our starter project
2 // Cmake Workshop Prog 6 – Step 4
3 //
4 // HAN University of Applied Sciences
5 // Johan Korten – v1.0 2025
6 //
7 // main.cpp
8 // compile using:
9 // g++ main.cpp sht45.cpp i2c.cpp logger.cpp config.cpp filter.cpp -I./include
10 // | -std=c++17 -Wall -O2 -o sensor_app
11
12 #include <iostream>
13 #include "sht45.hpp"
14 #include "i2c.hpp"
15 #include "logger.hpp"
16 #include "config.hpp"
17 #include "filter.hpp"
18
19 int main() {
20     Logger logger;
21     Config config;      // Config loaded in constructor
22     I2C i2c;           // I2C bus initialized in constructor
23     Filter filter;
24     SHT45 sensor;
```

```
g++ main.cpp sht45.cpp i2c.cpp logger.cpp config.cpp filter.cpp -I./include -std=c++17 -Wall -O2 -o sensor_app
```

Fighting Commandline Nightmares - Stepping stone 5

Build Nightmare

```
g++ main.cpp sht45.cpp i2c.cpp logger.cpp config.cpp filter.cpp -I./include -std=c++17 -Wall -O2 -o sensor_app
```

Alternative: Makefile

```
# Makefile for Sensor Application
# CMake Workshop - HAN University of Applied Sciences
# Johan Korten - v1.0 2025

# Compiler and flags
CXX = g++
CXXFLAGS = -std=c++17 -Wall -Wextra -O2 -I./include
LDFLAGS =

# Target executable
TARGET = sensor_app

# Source files
SOURCES = main.cpp sht45.cpp i2c.cpp logger.cpp config.cpp filter.cpp

# Object files (replace .cpp with .o)
OBJECTS = main.o sht45.o i2c.o logger.o config.o filter.o

# Header files (for dependency tracking)
HEADERS = include/sht45.hpp include/i2c.hpp include/logger.hpp include/config.hpp include/filter.hpp

# Default target
all: $(TARGET)

# Link object files to create executable
$(TARGET): $(OBJECTS)
    $(CXX) $(OBJECTS) $(LDFLAGS) -o $(TARGET)
    @echo "Build complete: $(TARGET)"

# Compile main.cpp
main.o: main.cpp $(HEADERS)
    $(CXX) $(CXXFLAGS) -c main.cpp

# Compile sht45.cpp
sht45.o: sht45.cpp include/sht45.hpp
    $(CXX) $(CXXFLAGS) -c sht45.cpp

# Compile i2c.cpp
i2c.o: i2c.cpp include/i2c.hpp
    $(CXX) $(CXXFLAGS) -c i2c.cpp

# Compile logger.cpp
logger.o: logger.cpp include/logger.hpp
    $(CXX) $(CXXFLAGS) -c logger.cpp

# Compile config.cpp
config.o: config.cpp include/config.hpp
    $(CXX) $(CXXFLAGS) -c config.cpp

# Compile filter.cpp
filter.o: filter.cpp include/filter.hpp
    $(CXX) $(CXXFLAGS) -c filter.cpp

# Clean build artifacts
clean:
    rm -f $(OBJECTS) $(TARGET)
    @echo "Clean complete"

# Rebuild everything from scratch
rebuild: clean all

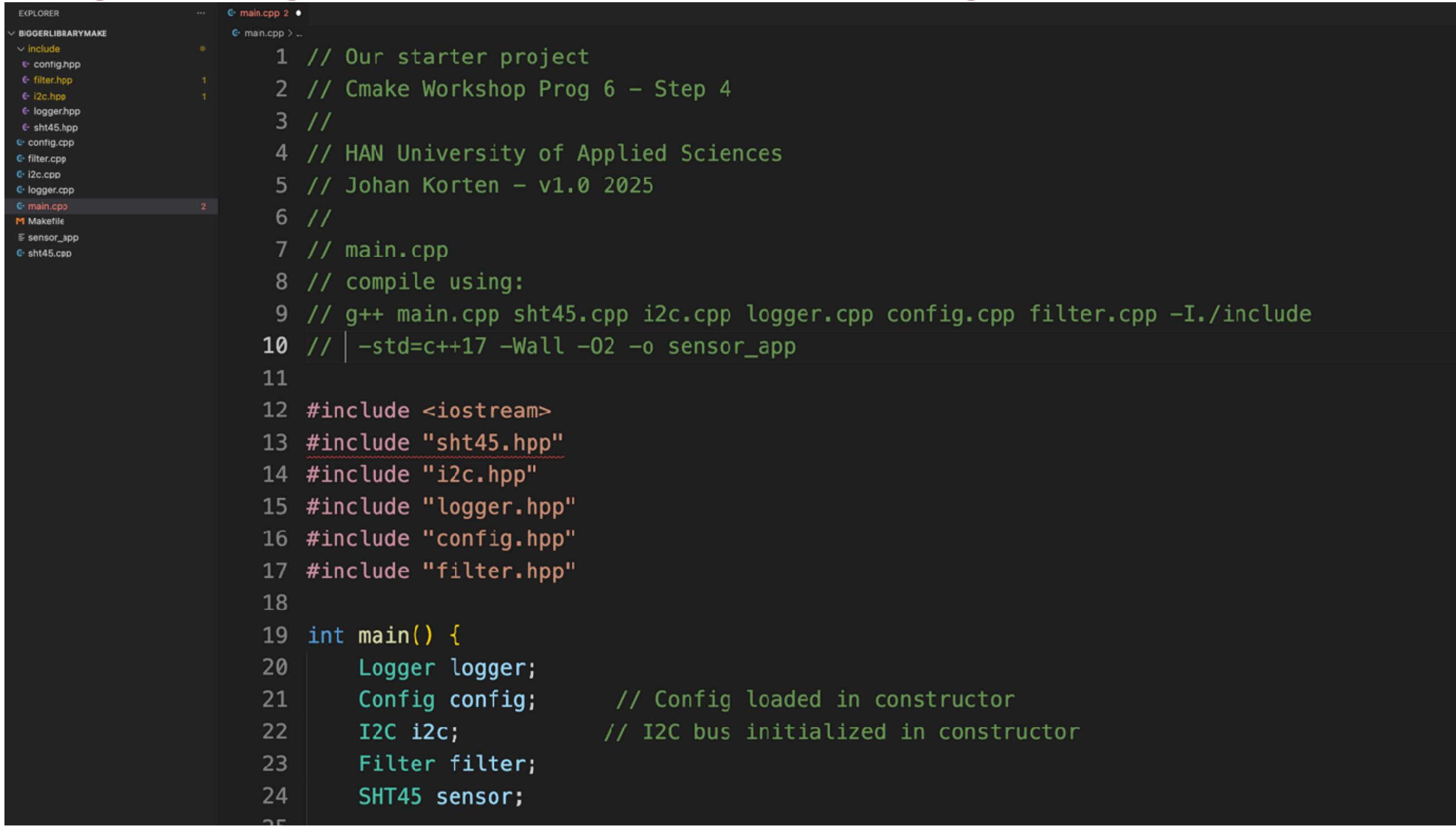
# Run the application
run: $(TARGET)
    ./$(TARGET)

# Show help
help:
    @echo "Available targets:"
    @echo " all - Build the application (default)"
    @echo " clean - Remove all build artifacts"
    @echo " rebuild - Clean and build from scratch"
    @echo " run - Build and run the application"
    @echo " help - Show this help message"

.PHONY: all clean rebuild run help
```

Fighting Commandline Nightmares - Stepping stone 5

Build Nightmare



The screenshot shows a code editor with the main.cpp file open. The code is a simple C++ program that includes several header files and initializes objects from them. The sidebar shows a CMake project structure with files like config.hpp, filter.hpp, i2c.hpp, logger.hpp, sht45.hpp, config.cpp, filter.cpp, i2c.cpp, logger.cpp, main.cpp, Makefile, sensor_app, and sht45.cpp.

```
1 // Our starter project
2 // Cmake Workshop Prog 6 - Step 4
3 //
4 // HAN University of Applied Sciences
5 // Johan Korten - v1.0 2025
6 //
7 // main.cpp
8 // compile using:
9 // g++ main.cpp sht45.cpp i2c.cpp logger.cpp config.cpp filter.cpp -I./include
10 // | -std=c++17 -Wall -O2 -o sensor_app
11
12 #include <iostream>
13 #include "sht45.hpp"
14 #include "i2c.hpp"
15 #include "logger.hpp"
16 #include "config.hpp"
17 #include "filter.hpp"
18
19 int main() {
20     Logger logger;
21     Config config;      // Config loaded in constructor
22     I2C i2c;           // I2C bus initialized in constructor
23     Filter filter;
24     SHT45 sensor;
```

```
jakorten@device-35 BiggerLibraryMake % make
g++ -std=c++17 -Wall -Wextra -O2 -I./include -c main.cpp
g++ -std=c++17 -Wall -Wextra -O2 -I./include -c sht45.cpp
g++ -std=c++17 -Wall -Wextra -O2 -I./include -c i2c.cpp
g++ -std=c++17 -Wall -Wextra -O2 -I./include -c logger.cpp
g++ -std=c++17 -Wall -Wextra -O2 -I./include -c config.cpp
g++ -std=c++17 -Wall -Wextra -O2 -I./include -c filter.cpp
g++ main.o sht45.o i2c.o logger.o config.o filter.o -o sensor_app
Build complete: sensor_app
jakorten@device-35 BiggerLibraryMake % make run
./sensor_app
[LOG] Starting sensor application...
Temperature: 20.25
[LOG] Application complete
jakorten@device-35 BiggerLibraryMake %
```

```
jakorten@device-35 BiggerLibraryMake % make clean
rm -f main.o sht45.o i2c.o logger.o config.o filter.o sensor_app
Clean complete
jakorten@device-35 BiggerLibraryMake %
```

```
cd BiggerLibraryMake
make          # Build the project
make run      # Build and run
make clean    # Clean up
make help     # See available commands
```

The problem of makefiles

Build Nightmare

‘Makefile hell’: 200+ lines for a simple project

IDE lock-in: Can't share code between STM32CubeIDE / VS Code / CLion / whatever.

Fighting Commandline Nightmares - Stepping stone 5

Build Nightmare

From the cli nightmare to the make nightmare...

Pro's:

1. Incremental compilation - Only recompiles changed files
2. Dependency tracking - Recompiles when headers change
3. Clean separation - Variables for compiler, flags, sources
4. Helper targets - clean, rebuild, run, help
5. Standard conventions - Uses CXX, CXXFLAGS, etc.

Con's:

1. Manual listing - Every .cpp and .o file must be listed
2. Repetitive rules - 6 nearly identical compilation rules
3. Manual dependency tracking - Have to specify which .cpp depends on which .hpp
4. Platform-specific - Won't work on Windows without changes
5. Hard-coded compiler - g++ might not be available everywhere
6. Tedious maintenance - Add new file = edit 3+ places in Makefile

Towards a library - Stepping stone 6

Build Nightmare

The screenshot shows a code editor interface with several tabs and panes. On the left, the Explorer pane shows a project structure for 'BIGGERLIBRARYCMAKE' containing files like 'build', 'include', 'CMakeLists.txt', 'config.cpp', 'filter.cpp', 'i2c.cpp', 'logger.cpp', 'main.cpp', 'Makefile', and 'sht45.cpp'. The main code editor pane displays the 'main.cpp' file with the following content:

```
1 // Our starter project
2 // Cmake Workshop Prog 6 – Step 6
3 //
4 // HAN University of Applied Sciences
5 // Johan Korten – v1.0 2025
6 //
7 // main.cpp
8 // compile using:
9 // g++ main.cpp sht45.cpp i2c.cpp logger.cpp config.cpp filter.cpp -I./include -std=c++17 -l
```

Below the code editor is a terminal window showing the build process:

```
[main] Building folder: /Users/jakorten/Library/Mobile Documents/com~apple~CloudDocs/Documents/Documenten – MacBook Pro van JA/HAN/HAN Engineering/S3 Programming – Software Engineering/Repo2425/ESE_PROG/Workshops/CMake/BiggerLibraryCMake/build
[build] Starting build
[proc] Executing command: /opt/homebrew/bin/cmake --build "/Users/jakorten/Library/Mobile Documents/com~apple~CloudDocs/Documents/Documenten – MacBook Pro van JA/HAN/HAN Engineering/S3 Programming – Software Engineering/Repo2425/ESE_PROG/Workshops/CMake/BiggerLibraryCMake/build" --config Debug
--target all --
[build] [6/7 14% :: 0.231] Building CXX object CMakeFiles/sensor_app.dir/sht45.cpp.o
[build] [6/7 28% :: 0.231] Building CXX object CMakeFiles/sensor_app.dir/i2c.cpp.o
[build] [6/7 42% :: 0.231] Building CXX object CMakeFiles/sensor_app.dir/config.cpp.o
[build] [6/7 57% :: 0.231] Building CXX object CMakeFiles/sensor_app.dir/filter.cpp.o
[build] [6/7 71% :: 0.602] Building CXX object CMakeFiles/sensor_app.dir/main.cpp.o
[build] [6/7 85% :: 0.603] Building CXX object CMakeFiles/sensor_app.dir/logger.cpp.o
[build] [7/7 100% :: 0.715] Linking CXX executable sensor_app
[driver] Build completed: 00:00:00.743
[build] Build finished with exit code 0
```

```
cd BiggerLibraryCMake
mkdir build
cd build
cmake ..          # Generate build system
make              # Build (or 'cmake --build .')
./sensor_app      # Run
```

Towards a library - Stepping stone 6

Build Nightmare

```
M Makefile
1 # Makefile for Sensor Application
2 # CMake Workshop – HAN University of Applied Sciences
3 # Johan Korten – v1.0 2025
4
5 # Compiler and flags
6 CXX = g++
7 CXXFLAGS = -std=c++17 -Wall -Wextra -O2 -I./include
8 LDFLAGS =
9
10 # Target executable
11 TARGET = sensor_app
12
13 # Source files
14 SOURCES = main.cpp sht45.cpp i2c.cpp logger.cpp config.cpp filter.cpp
15
16 # Object files (replace .cpp with .o)
17 OBJECTS = main.o sht45.o i2c.o logger.o config.o filter.o
```

```
cd BiggerLibraryCMake
mkdir build
cd build
cmake ..          # Generate build system
make              # Build (or 'cmake --build .')
./sensor_app      # Run
```

Makefile 1/5 1..17

Towards a library - Stepping stone 6

Build Nightmare

```
M Makefile
18
19 # Header files (for dependency tracking)
20 HEADERS = include/sht45.hpp include/i2c.hpp include/logger.hpp include/config.hpp include/
21
22 # Default target
23 all: $(TARGET)
24
25 # Link object files to create executable
26 $(TARGET): $(OBJECTS)
27     $(CXX) $(OBJECTS) $(LDFLAGS) -o $(TARGET)
28     @echo "Build complete: $(TARGET)"
29
30 # Compile main.cpp
31 main.o: main.cpp $(HEADERS)
32     $(CXX) $(CXXFLAGS) -c main.cpp
33
34 # Compile sht45.cpp
```

```
cd BiggerLibraryCMake
mkdir build
cd build
cmake ..      # Generate build system
make          # Build (or 'cmake --build .')
./sensor_app  # Run
```

Makefile 2/5 18..33

Towards a library - Stepping stone 6

Build Nightmare

```
M Makefile
34 # Compile sht45.cpp
35 sht45.o: sht45.cpp include/sht45.hpp
36     $(CXX) $(CXXFLAGS) -c sht45.cpp
37
38 # Compile i2c.cpp
39 i2c.o: i2c.cpp include/i2c.hpp
40     $(CXX) $(CXXFLAGS) -c i2c.cpp
41
42 # Compile logger.cpp
43 logger.o: logger.cpp include/logger.hpp
44     $(CXX) $(CXXFLAGS) -c logger.cpp
45
46 # Compile config.cpp
47 config.o: config.cpp include/config.hpp
48     $(CXX) $(CXXFLAGS) -c config.cpp
49
50 # Compile filter.cpp
```

```
cd BiggerLibraryCMake
mkdir build
cd build
cmake ..      # Generate build system
make          # Build (or 'cmake --build .')
./sensor_app  # Run
```

Makefile 3/5 34..49

Towards a library - Stepping stone 6

Build Nightmare

```
M Makefile
47 config.o: config.cpp include/config.hpp
48 # Compile filter.cpp
49 filter.o: filter.cpp include/filter.hpp
50 $(CXX) $(CXXFLAGS) -c filter.cpp
51
52 # Clean build artifacts
53 clean:
54 rm -f $(OBJECTS) $(TARGET)
55 @echo "Clean complete"
56
57 # Rebuild everything from scratch
58 rebuild: clean all
59
60 # Run the application
61 run: $(TARGET)
62 ./$(TARGET)
63
64
65
```

```
cd BiggerLibraryCMake
mkdir build
cd build
cmake ..      # Generate build system
make          # Build (or 'cmake --build .')
./sensor_app # Run
```

Makefile 4/5 47..64

Towards a library - Stepping stone 6

Build Nightmare

```
M Makefile
63 run: $(TARGET)
65
66 # Show help
67 help:
68     @echo "Available targets:"
69     @echo " all      - Build the application (default)"
70     @echo " clean    - Remove all build artifacts"
71     @echo " rebuild  - Clean and build from scratch"
72     @echo " run      - Build and run the application"
73     @echo " help     - Show this help message"
74
75 .PHONY: all clean rebuild run help
76
```

```
cd BiggerLibraryCMake
mkdir build
cd build
cmake ..          # Generate build system
make              # Build (or 'cmake --build .')
./sensor_app      # Run
```

Makefile 5/5 47..64

Towards a library - Stepping stone 6

Build Nightmare

```
M CMakeLists.txt
1 # CMakeLists.txt for Sensor Application
2 # CMake Workshop – HAN University of Applied Sciences
3 # Johan Korten – v1.0 2025
4
5 # Minimum CMake version required
6 cmake_minimum_required(VERSION 3.10)
7
8 # Project name and version
9 project(SensorApp VERSION 1.0 LANGUAGES CXX)
10
11 # Set C++ standard
12 set(CMAKE_CXX_STANDARD 17)
13 set(CMAKE_CXX_STANDARD_REQUIRED ON)
14 set(CMAKE_CXX_EXTENSIONS OFF)
15
16 # Compiler flags
17 add_compile_options(-Wall -Wextra -O2)
```

```
cd BiggerLibraryCMake
mkdir build
cd build
cmake ..          # Generate build system
make              # Build (or 'cmake --build .')
./sensor_app      # Run
```

CMakeLists.txt
1/2 1..17

Towards a library - Stepping stone 6

Build Nightmare

```
M CMakeLists.txt
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17 add_compile_options(-Wall -Wextra -O2)
18
19 # Include directories
20 include_directories(include)
21
22 # Source files
23 set(SOURCES
24     main.cpp
25     sht45.cpp
26     i2c.cpp
27     logger.cpp
28     config.cpp
29     filter.cpp
30 )
31
32 # Create executable
33 add_executable(sensor_app ${SOURCES})
34
35 # Print build information
36 message(STATUS "Building ${PROJECT_NAME} version ${PROJECT_VERSION}")
37 message(STATUS "C++ Standard: C++${CMAKE_CXX_STANDARD}")
38
```

```
cd BiggerLibraryCMake
mkdir build
cd build
cmake ..      # Generate build system
make          # Build (or 'cmake --build .')
./sensor_app # Run
```

CMakeLists.txt
2/2 17..37

Makefile versus CMakeLists.txt

Build Nightmare

1. No manual .o file listing - CMake figures it out
2. No repetitive rules - One add_executable() does it all
3. Automatic dependency tracking - CMake scans headers
4. Cross-platform - Same file works on Windows/Linux/Mac
5. Modern build system - Generates Makefiles, Visual Studio projects, Ninja, etc.

Makefile versus CMakeLists.txt

Build Nightmare

```
# Just list the sources
set(SOURCES
    main.cpp
    sht45.cpp
    i2c.cpp
    logger.cpp
    config.cpp
    filter.cpp
)
# CMake figures out the rest!
add_executable(sensor_app ${SOURCES})
```

Makefile versus CMakeLists.txt

Build Nightmare

```
# Just list the sources
set(SOURCES
    main.cpp
    sht45.cpp
    i2c.cpp
    logger.cpp
    config.cpp
    filter.cpp
)
# CMake figures out the rest!
add_executable(sensor_app ${SOURCES})
```

Workshop Topics for Today

Up-to-date CMake

- Overcoming the Build Nightmare
- **Cmake ins- and outs**
- Cmake for Embedded targets
- Alternatives to make

Towards a library - Stepping stone 6

Up-to-date CMake

BiggerLibraryCMakeV2:

Stage 1 - What we currently have

Stage 2 - Updated CMake

Stage 3 - Separate libraries for reusability

Stage 4 - FetchContent for external libraries

Stage 5 - How real projects configure CMake

Stage 6 - CMake for Embedded Systems targets

CMake V2 stage2

Up-to-date CMake

BiggerLibraryCMakeV2:

3.20+ to use ‘modern’ features.

```
target_include_directories(sht45_lib PUBLIC include) # vs
target_include_directories(sht45_lib PRIVATE include) # vs
target_include_directories(sht45_lib INTERFACE include)
```

Tip: always use “PRIVATE” unless you need
INTERFACE or PUBLIC

```
8 cmake_minimum_required(VERSION 3.20) # Updated for modern features
9 project(SensorApp VERSION 1.0 LANGUAGES CXX)
10
11 # Set C++ standard
12 set(CMAKE_CXX_STANDARD 17)
13 set(CMAKE_CXX_STANDARD_REQUIRED ON)
14 set(CMAKE_CXX_EXTENSIONS OFF)
15
16 # Source files
17 set(SOURCES
18   main.cpp
19   sht45.cpp
20   i2c.cpp
21   logger.cpp
22   config.cpp
23   filter.cpp
24 )
25
26 # Create executable
27 add_executable(sensor_app ${SOURCES})
28
29 # MODERN: Use target-specific commands with PRIVATE visibility
30 # This means ONLY sensor_app uses these settings, not everything in the project!
31 target_include_directories(sensor_app PRIVATE include)
32 target_compile_options(sensor_app PRIVATE -Wall -Wextra -O2)
33
34 # Print build information
35 message(STATUS "Building ${PROJECT_NAME} version ${PROJECT_VERSION}")
36 message(STATUS "C++ Standard: C++${CMAKE_CXX_STANDARD}")
37 message(STATUS "Using modern target-specific commands!")
38
```

CMake V2 stage3

BiggerLibraryCMakeV2:

Using Libraries:

```
# Create sensor hardware library (SHT45 + I2C)
add_library(sht45_lib STATIC
    sht45.cpp
    i2c.cpp
)
target_include_directories(sht45_lib PUBLIC include)
target_compile_options(sht45_lib PRIVATE -Wall -Wextra -O2)
```

```
# Link libraries to executable
target_link_libraries(sensor_app
    PRIVATE
        sht45_lib
        utils_lib
)
```

Up-to-date CMake

```
8 cmake_minimum_required(VERSION 3.20)
9 project(SensorApp VERSION 1.0 LANGUAGES CXX)
10
11 set(CMAKE_CXX_STANDARD 17)
12 set(CMAKE_CXX_STANDARD_REQUIRED ON)
13 set(CMAKE_CXX_EXTENSIONS OFF)
14
15 # Create sensor hardware library (SHT45 + I2C)
16 add_library(sht45_lib STATIC
17     sht45.cpp
18     i2c.cpp
19 )
20 target_include_directories(sht45_lib PUBLIC include)
21 target_compile_options(sht45_lib PRIVATE -Wall -Wextra -O2)
22
23 # Create utilities library (Logger, Config, Filter)
24 add_library(utils_lib STATIC
25     logger.cpp
26     config.cpp
27     filter.cpp
28 )
29 target_include_directories(utils_lib PUBLIC include)
30 target_compile_options(utils_lib PRIVATE -Wall -Wextra -O2)
31
32 # Create main executable
33 add_executable(sensor_app main.cpp)
34
35 # Link libraries to executable
36 target_link_libraries(sensor_app
37     PRIVATE
38         sht45_lib
39         utils_lib
40 )
41
42 # Compile options for the executable
43 target_compile_options(sensor_app PRIVATE -Wall -Wextra -O2)
```

CMake V2 stage4

Up-to-date CMake

BiggerLibraryCMakeV2:

External Dependencies
(Automatic Dependency Management)

```
16 # External dependencies with FetchContent
17 include(FetchContent)
18
19 # Add fmt library for better output formatting
20 FetchContent_Declare(
21     fmt
22     GIT_REPOSITORY https://github.com/fmtlib/fmt.git
23     GIT_TAG 10.1.1
24 )
25 FetchContent_MakeAvailable(fmt)
```

CMake V2 stage5

Up-to-date CMake

BiggerLibraryCMakeV2:

More advanced options including conditions

```
24 # Build type with sensible default
25 if(NOT CMAKE_BUILD_TYPE)
26   set(CMAKE_BUILD_TYPE Release)
27   message(STATUS "Build type not specified, defaulting to Release")
28 endif()
29
30 # Compiler-specific warnings
31 if(CMAKE_CXX_COMPILER_ID MATCHES "GNU|Clang")
32   add_compile_options(-Wall -Wextra -Wpedantic)
33 elseif(CMAKE_CXX_COMPILER_ID MATCHES "MSVC")
34   add_compile_options(/W4)
35 endif()
```

CMake V2 stage5

Up-to-date CMake

BiggerLibraryCMakeV2:

More advanced options including conditions

```
37 # Project options (user-configurable)
38 option(BUILD_TESTS "Build unit tests" OFF)
39 option(USE_FMT "Use fmt library for formatting" ON)
40 option(ENABLE_INSTALL "Enable installation rules" ON)

42 # External dependencies (conditional)
43 if(USE_FMT)
44     include(FetchContent)
45     message(STATUS "Fetching fmt library...")
46     FetchContent_Declare(
47         fmt
48         GIT_REPOSITORY https://github.com/fmtlib/fmt.git
49         GIT_TAG 10.1.1
50     )
51     FetchContent_MakeAvailable(fmt)
52 endif()
```

CMake V2 stage5

Up-to-date CMake

BiggerLibraryCMakeV2:

More advanced options including conditions

```
37 # Project options (user-configurable)
38 option(BUILD_TESTS "Build unit tests" OFF)
39 option(USE_FMT "Use fmt library for formatting" ON)
40 option(ENABLE_INSTALL "Enable installation rules" ON)

42 # External dependencies (conditional)
43 if(USE_FMT)
44     include(FetchContent)
45     message(STATUS "Fetching fmt library...")
46     FetchContent_Declare(
47         fmt
48         GIT_REPOSITORY https://github.com/fmtlib/fmt.git
49         GIT_TAG 10.1.1
50     )
51     FetchContent_MakeAvailable(fmt)
52 endif()

97 # Link fmt if enabled
98 if(USE_FMT)
99     target_link_libraries(utils PUBLIC fmt::fmt)
100    target_compile_definitions(utils PUBLIC USE_FMT)
101 endif()
```

CMake V2 stage5

BiggerLibraryCMakeV2:

Useful for e.g. CI/CD:

```
121 #
122 # Installation rules
123 #
124 if(ENABLE_INSTALL)
125     include(GNUInstallDirs)
126
127     # Install executable
128     install(TARGETS ${PROJECT_NAME}
129             RUNTIME DESTINATION ${CMAKE_INSTALL_BINDIR}
130             COMPONENT Runtime
131         )
132
133     # Install libraries
134     install(TARGETS sht45 utils
135             ARCHIVE DESTINATION ${CMAKE_INSTALL_LIBDIR}
136             LIBRARY DESTINATION ${CMAKE_INSTALL_LIBDIR}
137             COMPONENT Development
138         )
139
140     # Install headers
141     install(DIRECTORY include/
142             DESTINATION ${CMAKE_INSTALL_INCLUDEDIR}/${PROJECT_NAME}
143             COMPONENT Development
144             FILES_MATCHING PATTERN "*.hpp"
145         )
146
147     # Install README and LICENSE (if they exist)
148     install(FILES
149             ${CMAKE_CURRENT_SOURCE_DIR}/README.md
150             DESTINATION ${CMAKE_INSTALL_DOCDIR}
151             COMPONENT Documentation
152             OPTIONAL
153         )
154 endif()
```

Up-to-date CMake

CMake V2 stage5

Up-to-date CMake

BiggerLibraryCMakeV2:

Build config summary:

```
165 #
166 # Build configuration summary
167 #
168 message(STATUS "")
169 message(STATUS "=====")
170 message(STATUS "${PROJECT_NAME} v${PROJECT_VERSION}")
171 message(STATUS "=====")
172 message(STATUS "Build type:      ${CMAKE_BUILD_TYPE}")
173 message(STATUS "C++ Standard:    C++${CMAKE_CXX_STANDARD}")
174 message(STATUS "Compiler:        ${CMAKE_CXX_COMPILER_ID} ${CMAKE_CXX_COMPILER_VERSION}")
175 message(STATUS "Build tests:     ${BUILD_TESTS}")
176 message(STATUS "Use fmt:         ${USE_FMT}")
177 message(STATUS "Enable install:  ${ENABLE_INSTALL}")
178 message(STATUS "Install prefix:   ${CMAKE_INSTALL_PREFIX}")
179 message(STATUS "=====")
180 message(STATUS "")
```

CMake V2 stage5

Up-to-date CMake

BiggerLibraryCMakeV2:

More on this next week:

```
156 #
157 # Testing support
158 #
159 if(BUILD_TESTS)
160   enable_testing()
161   message(STATUS "Building with tests enabled")
162   # add_subdirectory(tests) # Uncomment when tests exist
163 endif()
164
```

Workshop Topics for Today

- Overcoming the Build Nightmare
- Cmake ins- and outs
- Cmake for Embedded targets**
- Alternatives to Make

CMake V2 stage6

BiggerLibraryCMakeV2:

arm-toolchain.cmake

```
1 # arm-toolchain.cmake
2 # ARM GCC Toolchain for STM32 Cross-Compilation
3 # CMake Workshop – HAN University of Applied Sciences
4
5 # Tell CMake we're cross-compiling (not building for host system)
6 set(CMAKE_SYSTEM_NAME Generic)
7 set(CMAKE_SYSTEM_PROCESSOR arm)
8
9 # Specify the cross compiler
10 # These tools are from ARM GCC toolchain (arm-none-eabi-gcc)
11 set(CMAKE_C_COMPILER arm-none-eabi-gcc)
12 set(CMAKE_CXX_COMPILER arm-none-eabi-g++)
13 set(CMAKE_ASM_COMPILER arm-none-eabi-gcc)
14 set(CMAKE_OBJCOPY arm-none-eabi-objcopy)
15 set(CMAKE_OBJDUMP arm-none-eabi-objdump)
16 set(CMAKE_SIZE arm-none-eabi-size)
17 set(CMAKE_DEBUGGER arm-none-eabi-gdb)
18
19 # Don't try to link during CMake configuration
20 # (embedded targets need special linker scripts)
21 set(CMAKE_TRY_COMPILE_TARGET_TYPE STATIC_LIBRARY)
22
23 # Search for programs only in the build host directories
24 set(CMAKE_FIND_ROOT_PATH_MODE_PROGRAM NEVER)
25
26 # Search for libraries and headers only in the target directories
27 set(CMAKE_FIND_ROOT_PATH_MODE_LIBRARY ONLY)
28 set(CMAKE_FIND_ROOT_PATH_MODE_INCLUDE ONLY)
29 set(CMAKE_FIND_ROOT_PATH_MODE_PACKAGE ONLY)
30 |
```

Embedded CMake

CMake V2 stage6

BiggerLibraryCMakeV2:

CMakeLists.txt for an embedded target:

Embedded CMake

```
1 # CMakeLists.txt - Stage 6: Real-World STM32 Project
2 # CMake Workshop - HAN University of Applied Sciences
3 # Johan Korten - v1.0 2025
4 #
5 # STAGE 6: Real Embedded Example (STM32 ARM Cortex-M)
6 # Shows how workshop concepts apply to actual embedded development
7
8 cmake_minimum_required(VERSION 3.20)
9
10 # CRITICAL: Toolchain file MUST be set before project()
11 # This tells CMake we're cross-compiling for ARM, not the host system
12 set(CMAKE_TOOLCHAIN_FILE ${CMAKE_CURRENT_SOURCE_DIR}/arm-toolchain.cmake)
13
14 # Project definition
15 project(STM32_SHT45
16   VERSION 1.0.0
17   DESCRIPTION "SHT45 Sensor on STM32F4"
18   HOMEPAGE_URL "https://github.com/AEAEEmbedded/ESE\_PROG"
19   LANGUAGES C CXX ASM # ASM needed for startup code!
20 )
21
22 #
23 # MCU-Specific Settings
24 #
25 set(MCU_FAMILY STM32F4xx)
26 set(MCU_MODEL STM32F411xE)
27 set(MCU_LINKER_SCRIPT ${CMAKE_CURRENT_SOURCE_DIR}/STM32F411RETx_FLASH.ld)
28
```

CMake V2 stage6

BiggerLibraryCMakeV2:

CMakeLists.txt for an embedded target:

```
29 # ARM Cortex-M4 compiler flags
30 set(MCU_FLAGS
31     -mcpu=cortex-m4          # Target CPU
32     -mthumb                  # Thumb instruction set
33     -mfpu=fpv4-sp-d16        # Hardware FPU
34     -mfloat-abi=hard         # Use hardware FP
35 )
36
37 # Common compile definitions
38 add_compile_definitions(
39     ${MCU_MODEL}
40     USE_HAL_DRIVER
41     ARM_MATH_CM4
42 )
43
44 #
45 # STM32 HAL Library (Hardware Abstraction Layer)
46 #
47 add_library(stm32_hal STATIC
48     # HAL Core
49     Drivers/STM32F4xx_HAL_Driver/Src/stm32f4xx_hal.c
50     Drivers/STM32F4xx_HAL_Driver/Src/stm32f4xx_hal_cortex.c
51     Drivers/STM32F4xx_HAL_Driver/Src/stm32f4xx_hal_rcc.c
52     Drivers/STM32F4xx_HAL_Driver/Src/stm32f4xx_hal_gpio.c
53
54     # I2C for sensor communication
55     Drivers/STM32F4xx_HAL_Driver/Src/stm32f4xx_hal_i2c.c
56
57     # DMA for efficient transfers (optional)
58     Drivers/STM32F4xx_HAL_Driver/Src/stm32f4xx_hal_dma.c
59
60     # Add more HAL modules as needed:
61     # Drivers/STM32F4xx_HAL_Driver/Src/stm32f4xx_hal_uart.c
62     # Drivers/STM32F4xx_HAL_Driver/Src/stm32f4xx_hal_tim.c
63 )
```

Embedded CMake

CMake V2 stage6

BiggerLibraryCMakeV2:

CMakeLists.txt for an embedded target:

```
65 target_include_directories(stm32_hal PUBLIC
66     Drivers/STM32F4xx_HAL_Driver/Inc
67     Drivers/CMSIS/Device/ST/${MCU_FAMILY}/Include
68     Drivers/CMSIS/Include
69     Core/Inc # User configuration headers
70 )
71
72 target_compile_options(stm32_hal PRIVATE ${MCU_FLAGS} -Wall -O2)
73
74 #
75 # Sensor Library (Same concepts from workshop!)
76 #
77 add_library(sht45 STATIC
78     src/sht45.cpp
79     src/i2c_wrapper.cpp # Wraps STM32 HAL I2C functions
80 )
81
82 target_include_directories(sht45 PUBLIC
83     ${<BUILD_INTERFACE:${CMAKE_CURRENT_SOURCE_DIR}/include>}
84 )
85
86 # Link to HAL (sensor needs I2C)
87 target_link_libraries(sht45 PUBLIC stm32_hal)
88 target_compile_options(sht45 PRIVATE ${MCU_FLAGS} -Wall -Wextra)
89
90 #
91 # System Files
92 #
93 add_library(system OBJECT
94     Core/Src/system_stm32f4xx.c      # System clock configuration
95     Core/Src/stm32f4xx_it.c        # Interrupt handlers
96     startup_stm32f411xe.s         # Assembly startup code
97 )
```

Embedded CMake

CMake V2 stage6

BiggerLibraryCMakeV2:

CMakeLists.txt for an embedded target:

```
98
99 target_include_directories(system PUBLIC Core/Inc)
100 target_compile_options(system PRIVATE ${MCU_FLAGS})
101
102 #
103 # Main Firmware Executable
104 #
105 add_executable(${PROJECT_NAME}.elf
106   src/main.cpp
107   ${TARGET_OBJECTS}:system) # Include system files
108 )
109
110 # Link all libraries
111 target_link_libraries(${PROJECT_NAME}.elf
112   PRIVATE
113   sht45      # Our sensor library
114   stm32_hal # STM32 HAL
115 )
116
117 target_include_directories(${PROJECT_NAME}.elf PRIVATE
118   Core/Inc
119   include
120 )
121
122 target_compile_options(${PROJECT_NAME}.elf PRIVATE
123   ${MCU_FLAGS}
124   -Wall
125   -Wextra
126   -fdata-sections # Enable garbage collection
127   -ffunction-sections
128 )
```

Embedded CMake

CMake V2 stage6

BiggerLibraryCMakeV2:

CMakeLists.txt for an
embedded target:

```
129
130 #
131 # Linker Settings (Critical for embedded!)
132 #
133 target_link_options(${PROJECT_NAME}.elf PRIVATE
134   -T${MCU_LINKER_SCRIPT}           # Memory layout
135   -Wl,-Map=${PROJECT_NAME}.map    # Generate map file
136   -Wl,--gc-sections             # Remove unused code
137   -Wl,--print-memory-usage      # Show memory usage
138   ${MCU_FLAGS}
139   --specs=nano.specs            # Use smaller newlib-nano
140   --specs=nosys.specs           # No system calls
141 )
142
143 #
144 # Post-Build: Generate Binary Files for Flashing
145 #
146 find_program(ARM_SIZE arm-none-eabi-size REQUIRED)
147 find_program(ARM_OBJCOPY arm-none-eabi-objcopy REQUIRED)
148
149 add_custom_command(TARGET ${PROJECT_NAME}.elf POST_BUILD
150   # Generate Intel HEX format
151   COMMAND ${ARM_OBJCOPY} -O ihex
152     ${PROJECT_NAME}.elf
153     ${PROJECT_NAME}.hex
154   COMMENT "Generating HEX file..."
155
156   # Generate binary format
157   COMMAND ${ARM_OBJCOPY} -O binary
158     ${PROJECT_NAME}.elf
159     ${PROJECT_NAME}.bin
160   COMMENT "Generating BIN file..."
161
162   # Show size information
163   COMMAND ${ARM_SIZE} --format=berkeley ${PROJECT_NAME}.elf
164   COMMENT "Size analysis:"
165 )
```

Embedded CMake

CMake V2 stage6

BiggerLibraryCMakeV2:

CMakeLists.txt for an
embedded target:

```
167 #
168 # Flash Target (Optional: Program the MCU)
169 #
170 find_program(OPENOCD openocd)
171 if(OPENOCD)
172     add_custom_target(flash
173         COMMAND ${OPENOCD}
174             -f interface/stlink.cfg
175             -f target/stm32f4x.cfg
176             -c "program ${PROJECT_NAME}.elf verify reset exit"
177         DEPENDS ${PROJECT_NAME}.elf
178         COMMENT "Flashing ${PROJECT_NAME}.elf to STM32..."
179     )
180 endif()
181 #
182 #
183 # Debug Target (Optional: Start GDB server)
184 #
185 if(OPENOCD)
186     add_custom_target(debug
187         COMMAND ${OPENOCD}
188             -f interface/stlink.cfg
189             -f target/stm32f4x.cfg
190         COMMENT "Starting OpenOCD debug server..."
191     )
192 endif()
```

Embedded CMake

Workshop Topics for Today

- Overcoming the Build Nightmare
- Cmake ins- and outs
- Cmake for Embedded targets
- Alternative to Make**

Alternative to Make: Ninja

Ninja is a build system (like Make) but designed to be fast.

CMake can generate Ninja build files instead of Makefiles.

Alternative to Make: Ninja

Ninja is a build system (like Make) but designed to be fast.

CMake can generate Ninja build files instead of Makefiles.

cmake -B build_make time

cmake --build build_make

Make: 2.3 seconds

cmake -B build_ninja -G Ninja time

cmake --build build_ninja

Ninja: 0.8 seconds

Alternative to Make: Ninja

Why Ninja is Faster:

No shell overhead - Make spawns shells, Ninja doesn't

Parallel by default - Ninja automatically uses all cores

Minimal features - Ninja only builds, Make has a lot of often not used features

Better dependency tracking - Rebuilds only what's needed

Embedded projects with 100+ files compile in seconds, not minutes

Faster iteration = more learning

Industry uses Ninja (Google Chrome, Android, LLVM)

Alternative to Make: Ninja and others

For your embedded systems:

- Ninja - Fast, simple, used by modern CMake
- Meson - Great cross-compilation support
- Bazel - If working with larger embedded projects
- SCons/Waf - Common in embedded open source

The trend is toward:

- Faster incremental builds (Ninja, tup)
- Better dependency tracking (Bazel, Buck2)
- Cross-platform support (Meson, CMake+generators)
- Reproducible builds (Bazel, Nix-based systems)

“

That's all...

Any questions?