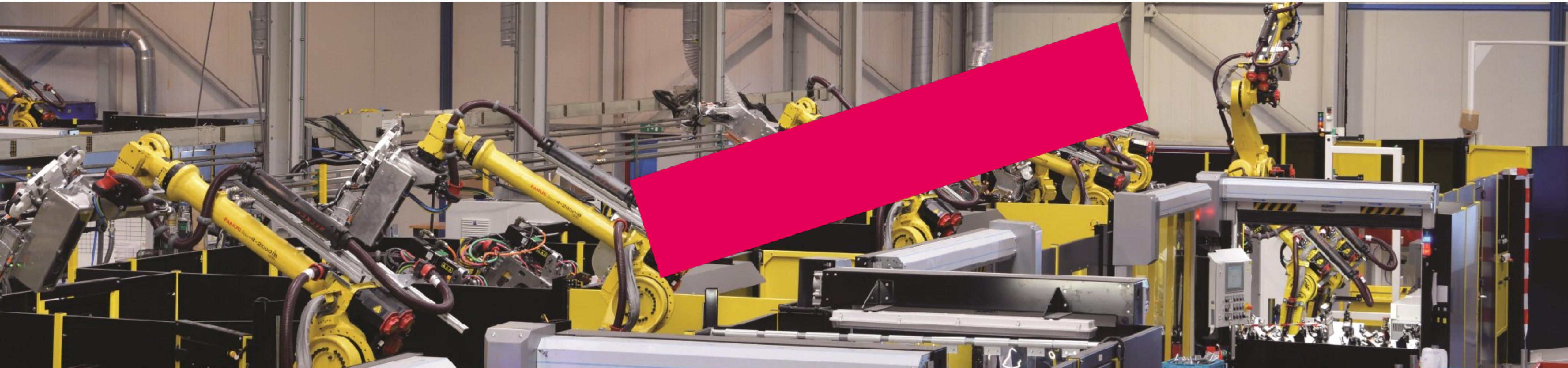


Programming 6



Workshop on CMake

johan.korten@han.nl

V1.0 Nov 2025

Workshop Topics overview

Clean code (beyond SOLID)

CMake

Unit testing

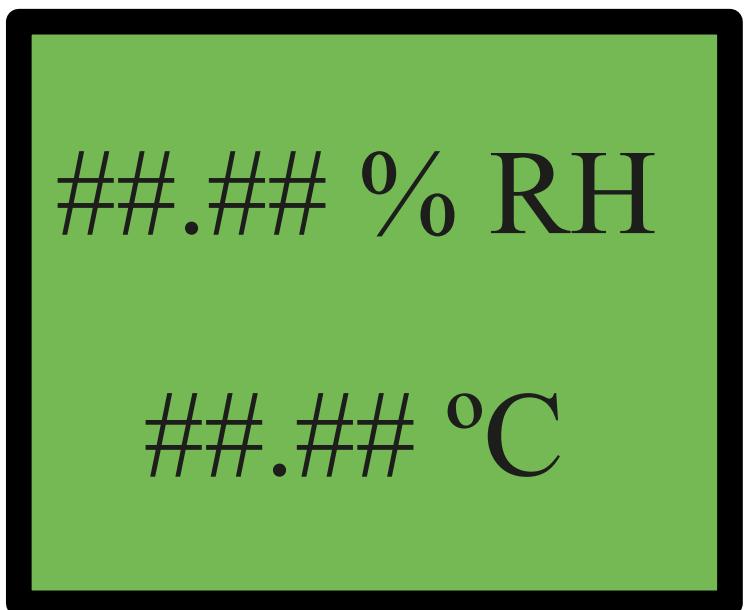
Commentaar

Patterns (Clean architecture)

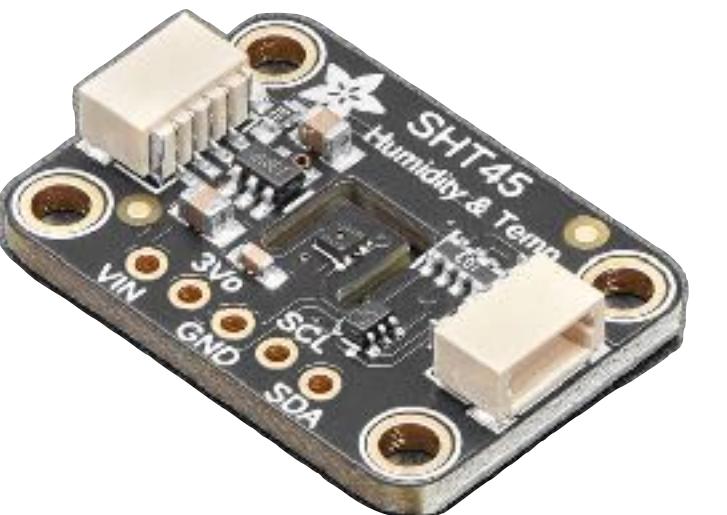
Example project for today

SHT45

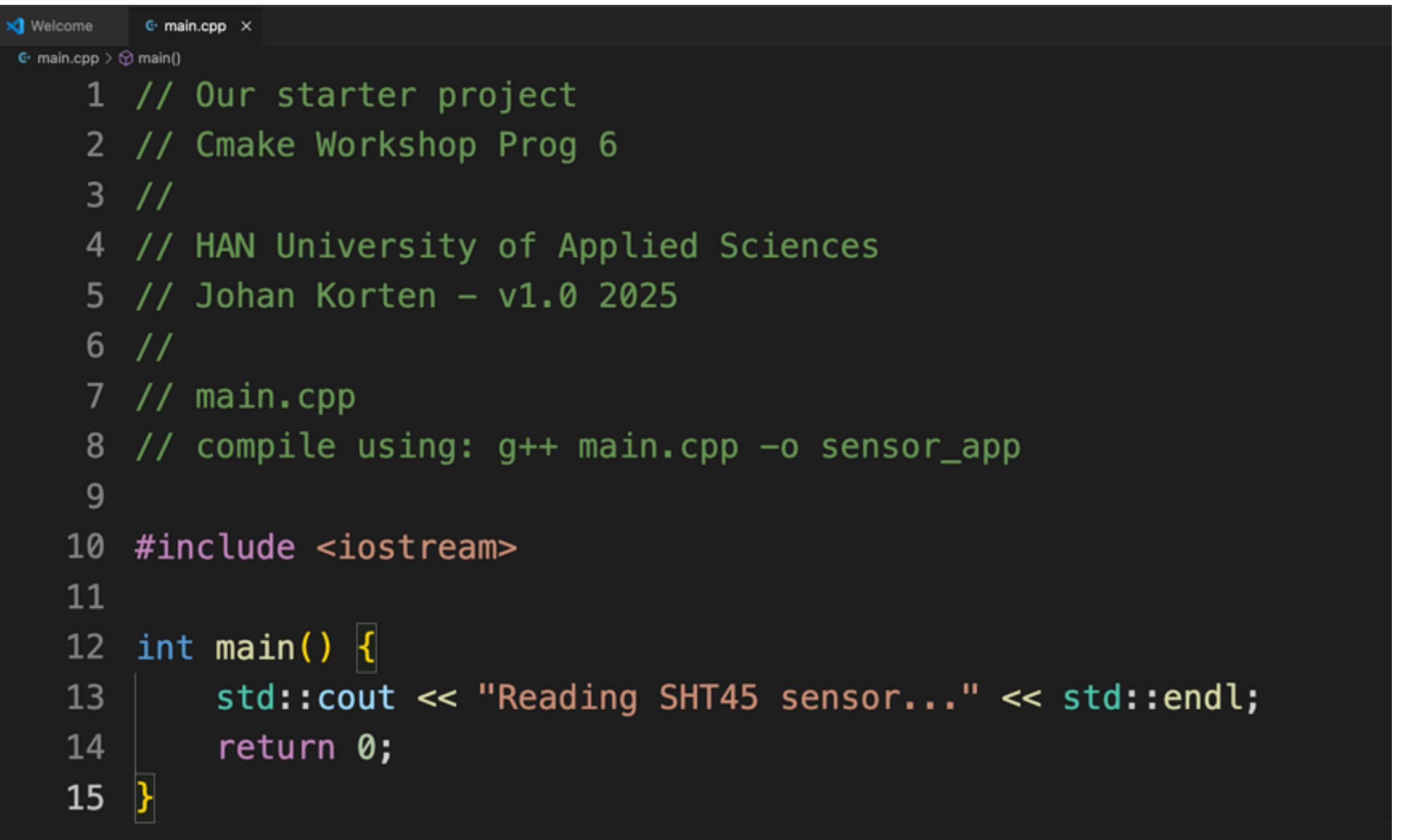
- Temperature
- Humidity Sensor



Lab
Environment
Monitor



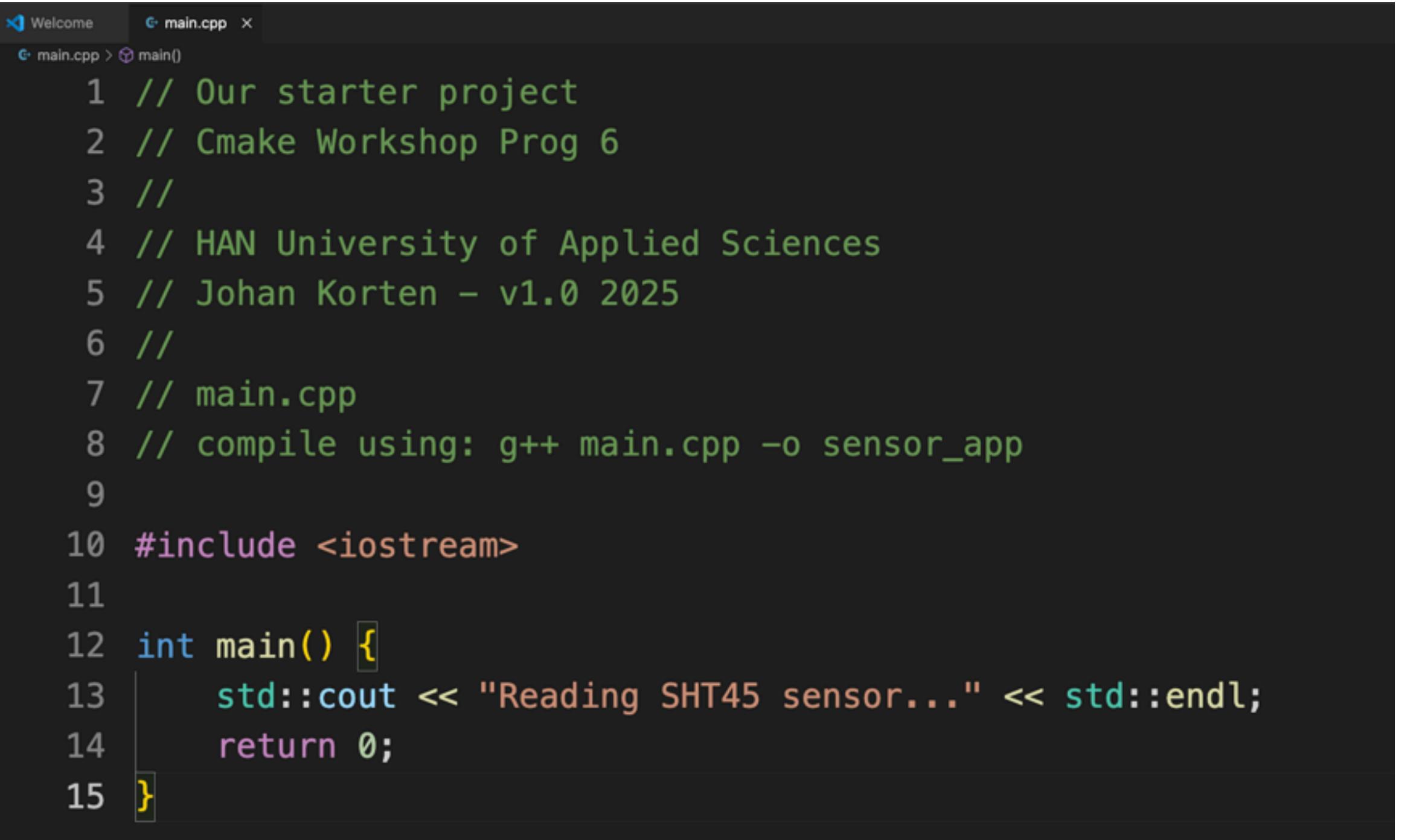
Pragmatic start - Stepping stone 1



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 explaining its purpose. It includes a header file, defines the main function, and outputs a message to the console.

```
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



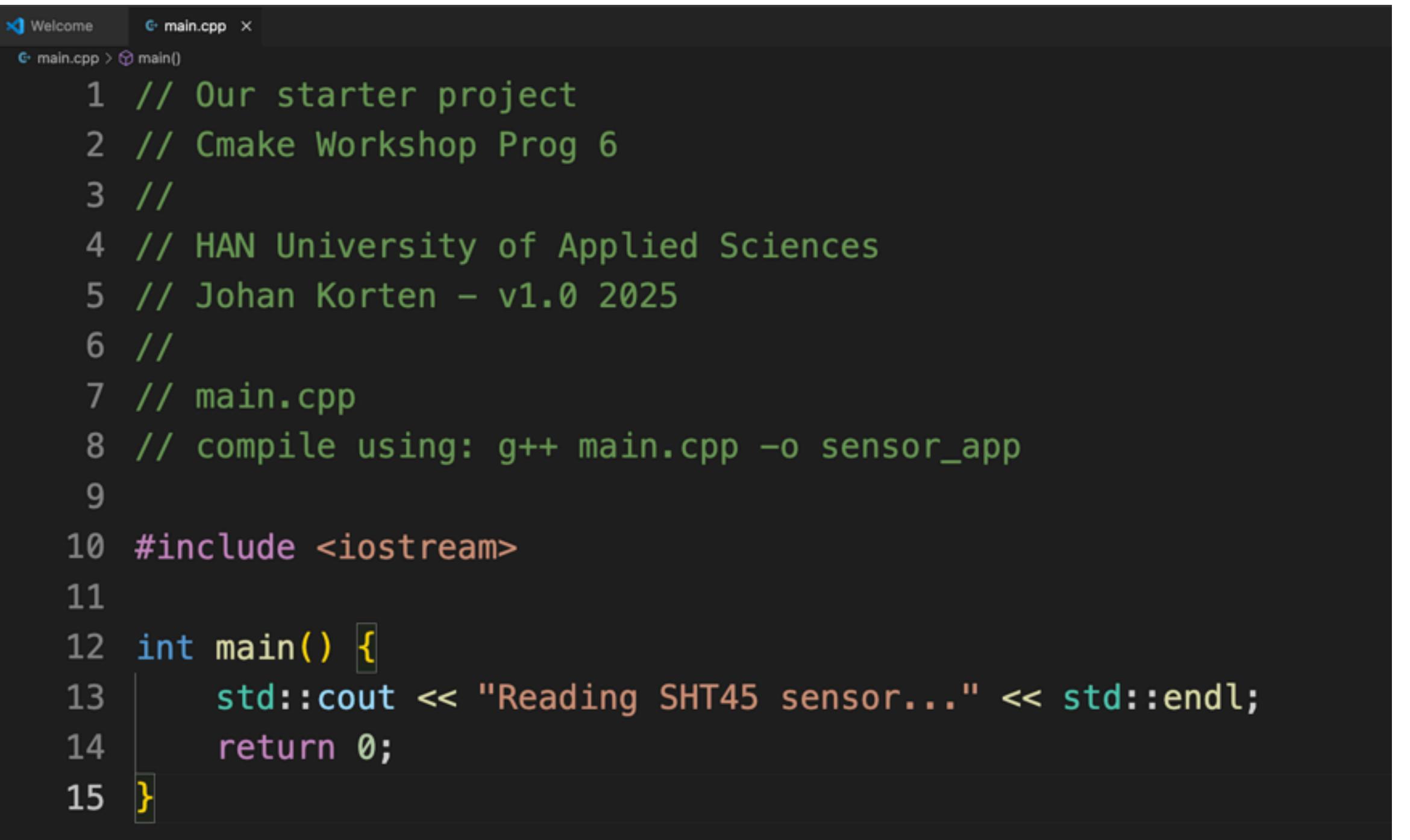
A screenshot of a code editor window titled "main.cpp". The code is a simple C++ program with the following content:

```
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

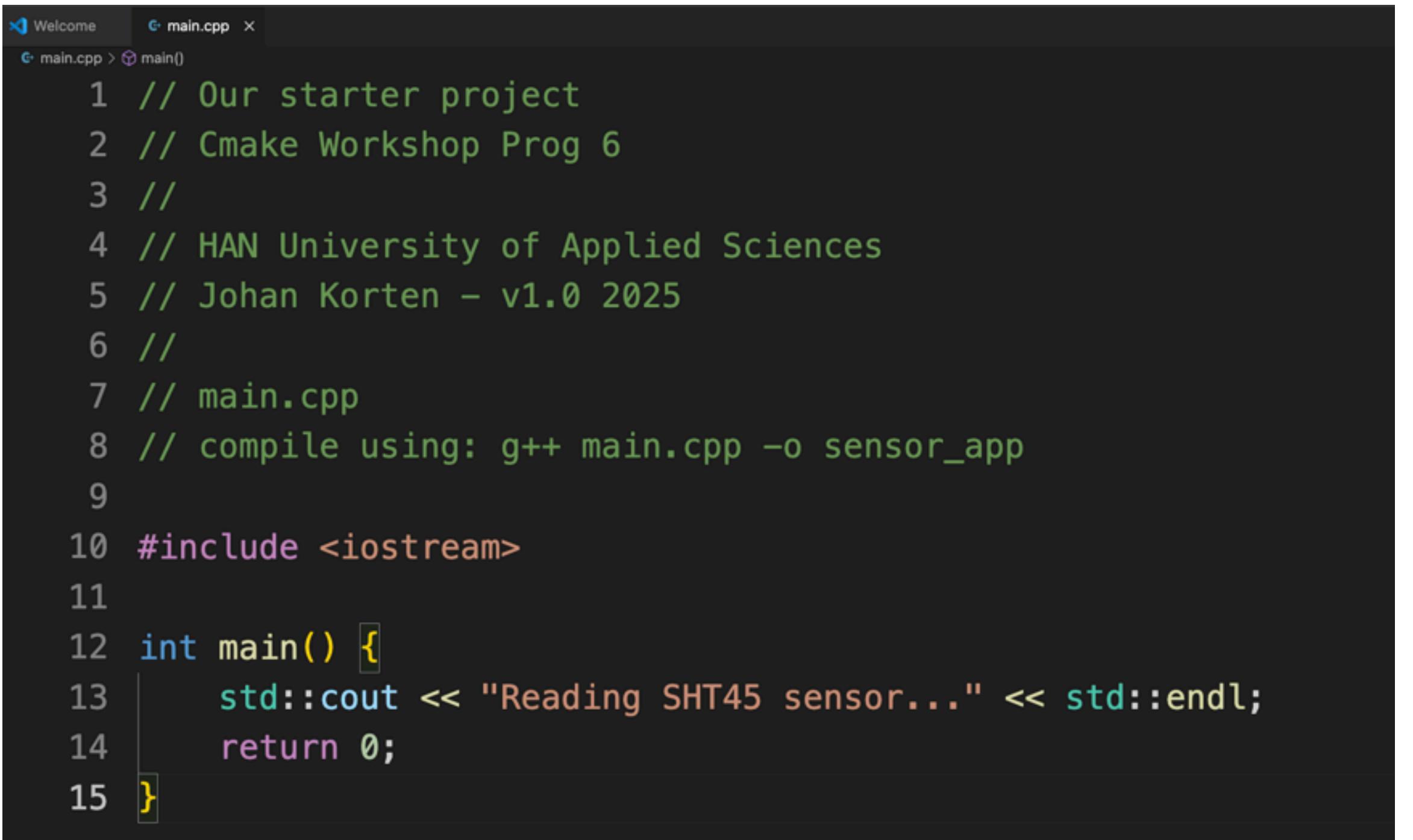


A screenshot of a code editor window titled "main.cpp". The code is a simple C++ program with the following content:

```
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



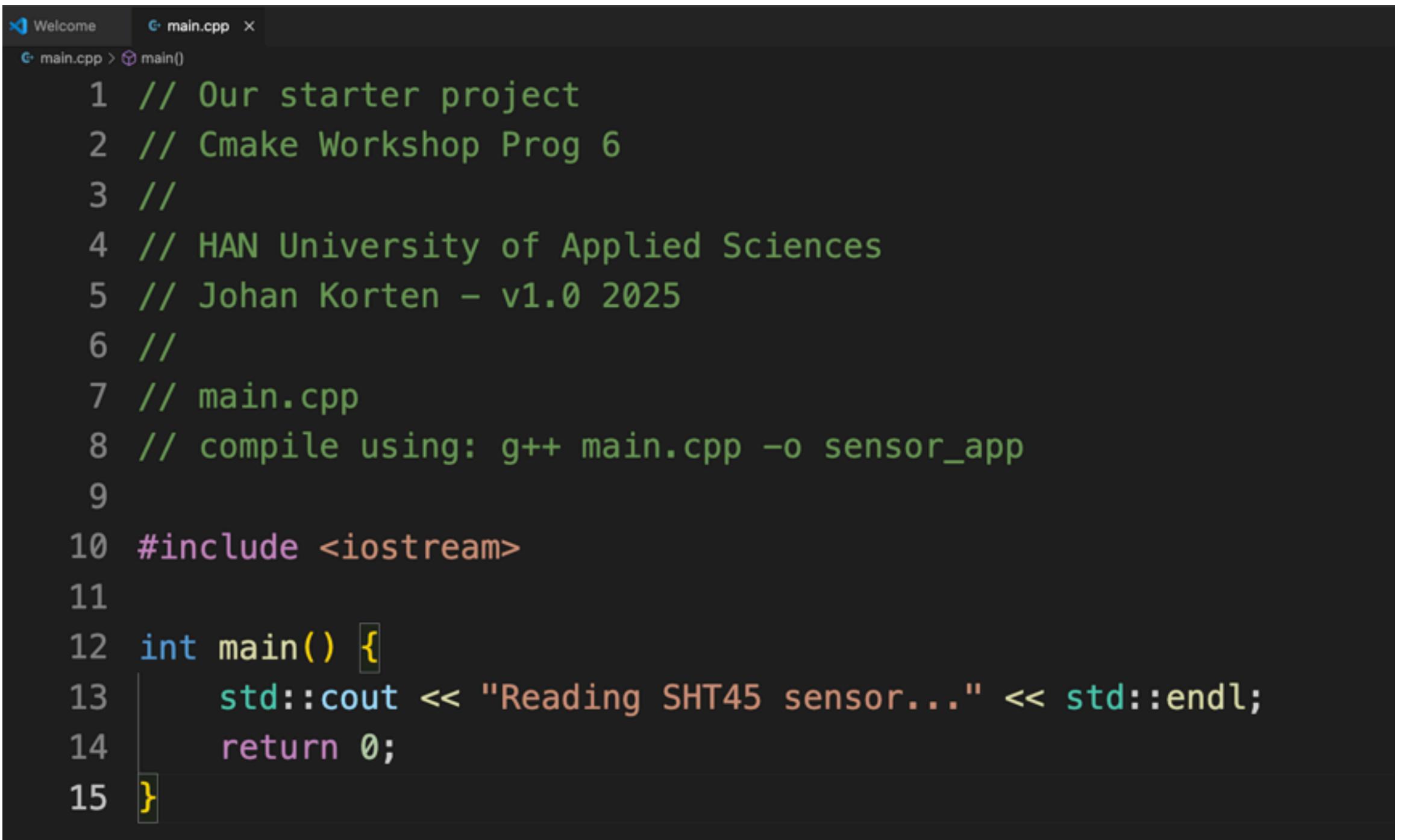
A screenshot of a code editor window titled "main.cpp". The code is a simple C++ program with the following content:

```
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...)

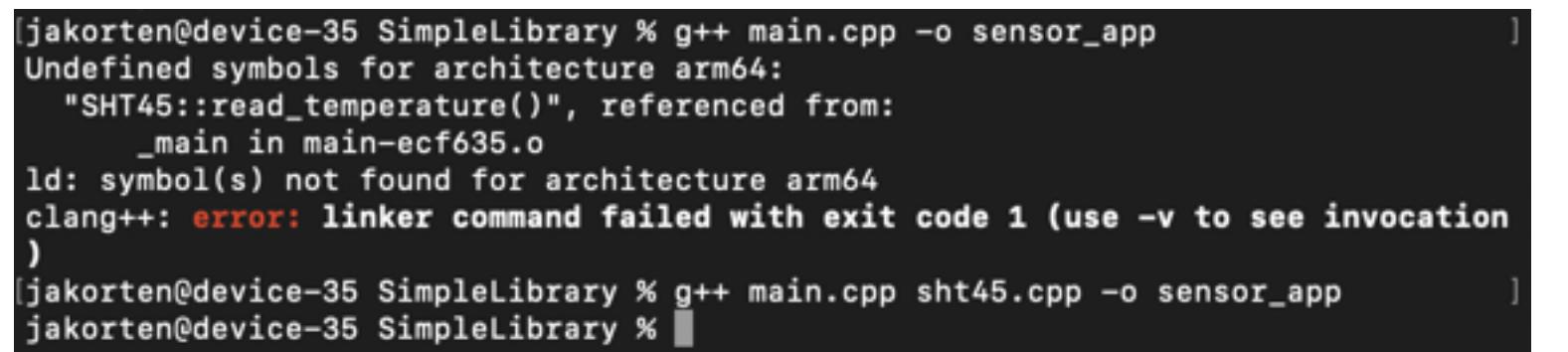
```
[jakorten@device-35 SimpleLibrary % g++ main.cpp -o sensor_app
Undefined symbols for architecture arm64:
  "SHT45::read_temperature()", referenced from:
    _main in main-ecf635.o
ld: symbol(s) not found for architecture arm64
clang++: error: linker command failed with exit code 1 (use -v to see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

Towards a library - Stepping stone 2



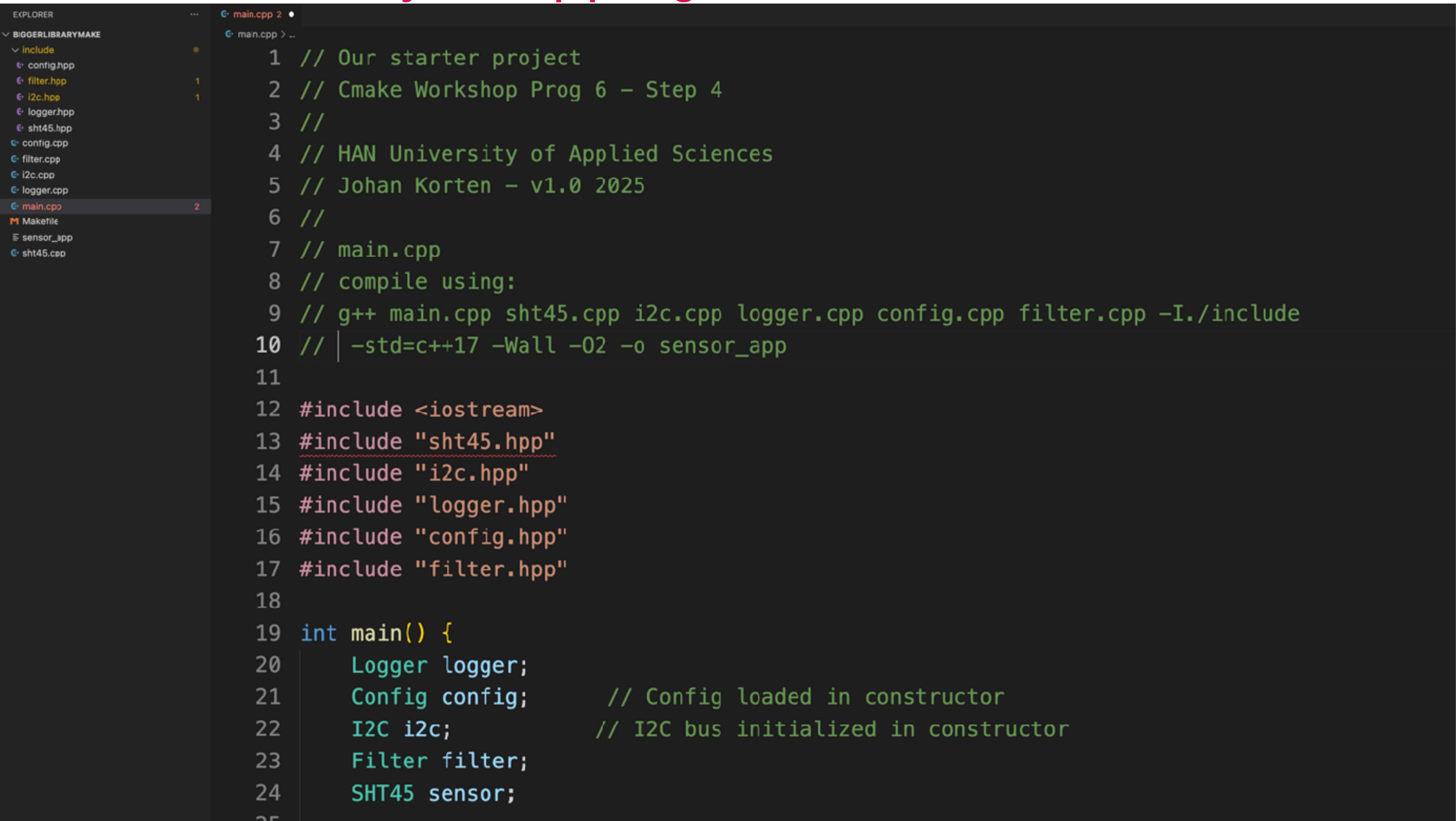
```
>Welcome  main.cpp x
main.cpp > main()
1 // Our starter project
2 // Cmake Workshop Prog 6
3 //
4 // HAN University of Applied Sciences
5 // Johan Korten - v1.0 2025
6 //
7 // main.cpp
8 // compile using: g++ main.cpp -o sensor_app
9
10 #include <iostream>
11
12 int main() {
13     std::cout << "Reading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

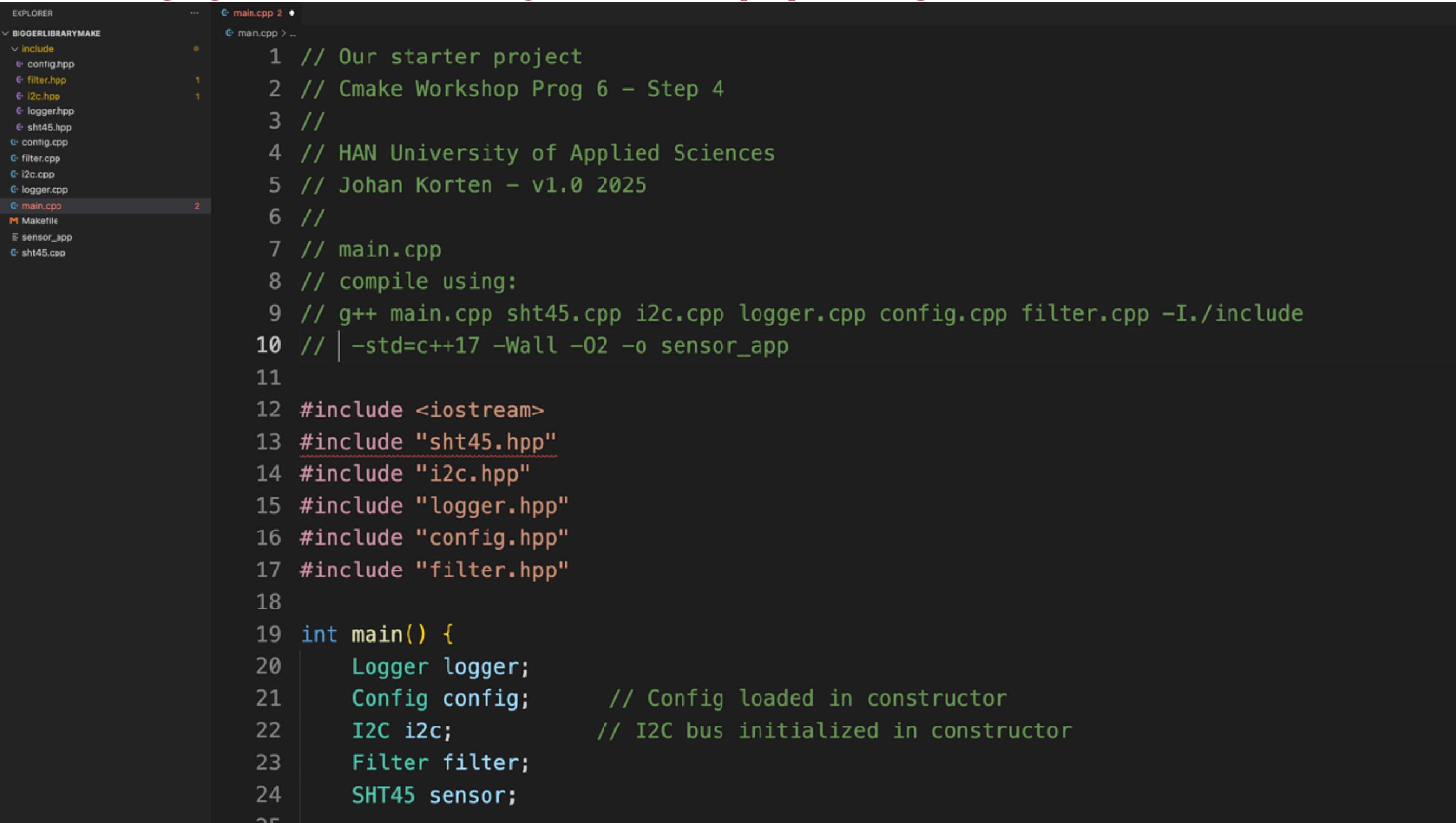
Towards a library - Stepping stone 3



The screenshot shows a code editor with a dark theme. On the left is the Explorer sidebar, which lists 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 that includes headers for std::iostream, sht45.hpp, i2c.hpp, logger.hpp, config.hpp, and filter.hpp. It defines a main() function that creates instances of Logger, Config, I2C, Filter, and SHT45.

```
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;
```

A bigger library - Stepping stone 4



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 '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 'main.cpp' file 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;
```

```
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

```
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"

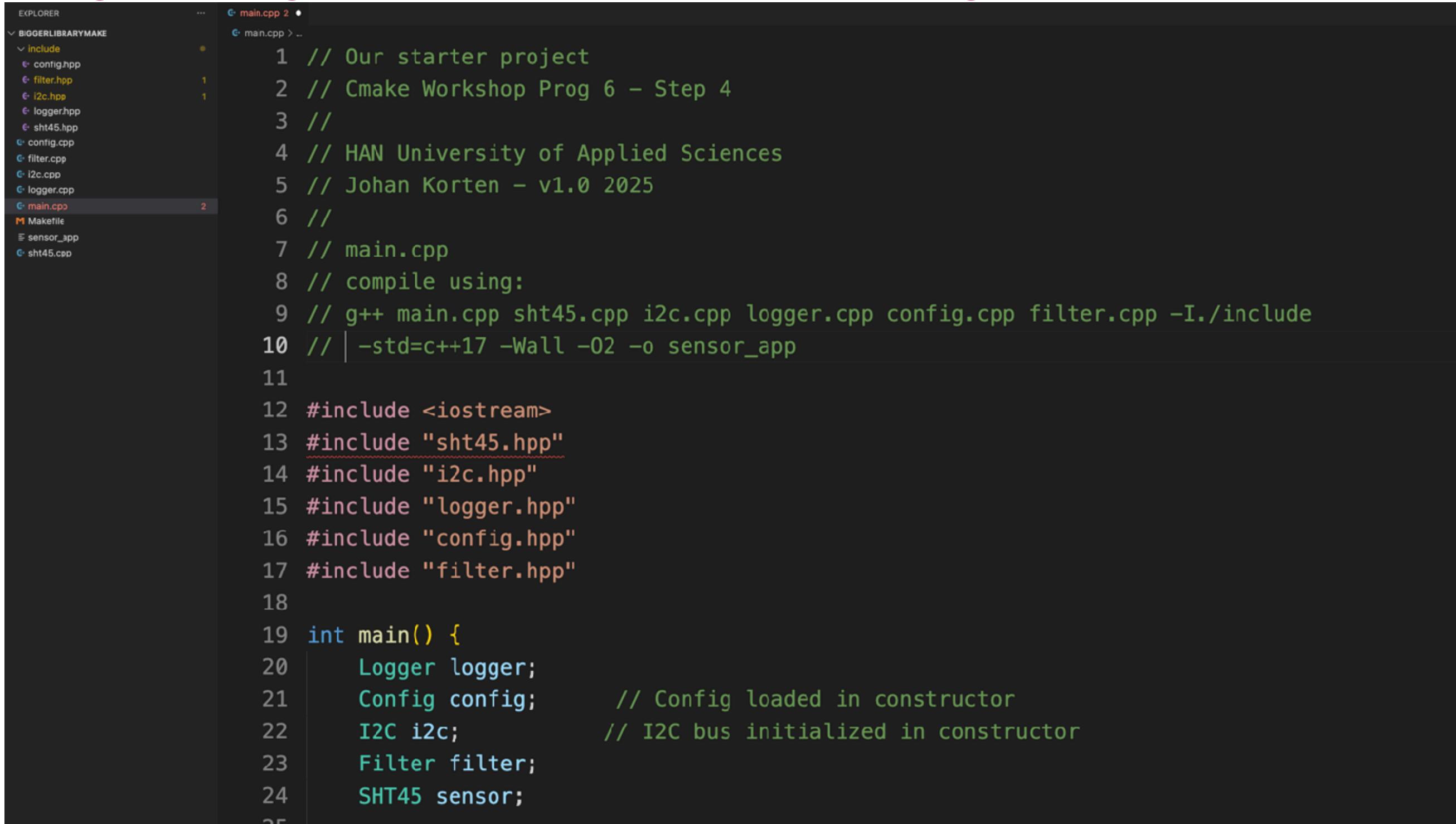
# Rebuild everything from scratch
rebuild: clean all

# 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



A screenshot of a code editor showing the main.cpp file. The code is a simple C++ program that includes headers for config, i2c, logger, sht45, filter, and main. It defines a main() function that creates instances of Logger, Config, I2C, Filter, and SHT45. The code editor also shows a sidebar with project 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

Makefiles: Easily 00+ lines for a simple project

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

Fighting Commandline Nightmares - Stepping stone 5

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

The screenshot shows a code editor interface with several panes. On the left is the Explorer pane, listing files in a directory structure. The main pane displays a C++ file named 'main.cpp' 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

```
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

```
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

```
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

```
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
53
54 # Clean build artifacts
55 clean:
56     rm -f $(OBJECTS) $(TARGET)
57     @echo "Clean complete"
58
59 # Rebuild everything from scratch
60 rebuild: clean all
61
62 # Run the application
63 run: $(TARGET)
64     ./$(TARGET)
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

```
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

```
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

```
M CMakeLists.txt
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 ..
make
./sensor_app
```

CMakeLists.txt
2/2 17..37

Makefile versus CMakeLists.txt

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

```
# 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})
```

Recap

- Without build system: Type long commands repeatedly
- With Makefile: Works on one platform, breaks on others
- With CMake: Write once, works everywhere

More on CMake

What is CMake

General Tips

More Advanced CMake

More on CMake

What is CMake

General Tips

More Advanced CMake

Next week

Unit Testing

Unit Testing + CMake

[https://github.com/AEAEmbedded/
ESE_PROG/tree/main/Workshops](https://github.com/AEAEmbedded/ESE_PROG/tree/main/Workshops)

Great Resources

C++ Core Guidelines (Stroustrup et al., ISO C++ Standards Committee)

Sutter & Alexandrescu (2005): C++ Coding Standards: 101 Rules, Guidelines, and Best Practices

JSF AV C++ (Lockheed Martin, 2005): Safety-critical embedded systems standard (221 rules)

MISRA C++:2023: Automotive/medical device coding guidelines for critical systems

Google C++ Style Guide: Industry best practices for large-scale development

Kolpackov's "Canonical Project Structure" (ISO C++ Paper P1204R0, 2018)

Sutter & Alexandrescu (2005): Coding standards promote consistency and international collaboration

Great Resources

C++ Core Guidelines (Stroustrup et al., ISO C++ Standards Committee)

Sutter & Alexandrescu (2005): C++ Coding Standards: 101 Rules, Guidelines, and Best Practices

JSF AV C++ (Lockheed Martin, 2005): Safety-critical embedded systems standard (221 rules)

MISRA C++:2023: Automotive/medical device coding guidelines for critical systems

Google C++ Style Guide: Industry best practices for large-scale development

Kolpackov's "Canonical Project Structure" (ISO C++ Paper P1204R0, 2018)

Sutter & Alexandrescu (2005): Coding standards promote consistency and international collaboration

“

That's all...

Any questions?