C++ introduction

C++ for Developers

C++ introduction

- History of C++
- C++ today
- Building a C++ program
- Installing Tools

History of C++

C++ for Developers

Background of programming

- Before modern programming
- Creation of OOP
- 1972: C brings programming into modern age
- 1979: C++ / C with classes
- Standardisation of C and C++
- C and C++ classed as ‘bad languages’

Pre-modern programming

- Complicated syntax
- High threshold
- Prone to errors
- Hard to maintain
- Not portable

```
          global    _start

          section   .text
_start:   mov       rax, 1
          mov       rdi, 1
          mov       rsi, message
          mov       rdx, 13
          syscall
          mov       rax, 60
          xor       rdi, rdi
          syscall

          section   .data
message:  db        "Hello, World", 10
```

lafftale for developers

Object-oriented programming

- Way of modelling programs around real world objects
- Early signs found at MIT 1950/1960 AI group
- Some OOP function in ALGOL
- Norwegian Computing Center 1960s - Simula

```
Begin
    OutText ("Hello, World!");
    Outimage;
End;
```

1972: The birth of C

- 1972 at Bell Labs by Dennis Ritchie to create utilities running on Unix.
- Access to memory through pointers instead of commands
- Imperative programming
- Compiled to machine code
- Portability

```c
int main(void) {
    printf("Hello World\n");
    return 0;
}
```

1979: C++ / C with classes

- 1979 Bjarne Stroustrup, Danish Scientist at AT&T Bell Labs
- Extension of C with classes
- Object-oriented programming
- Overloading, References, type-safe memory

lafftale for developers

Standardisation of C and C++

- C standardised 1989/1990
- C++ standardised 1998

Why is standardisation important?

- Agreement on syntax and standard functions
- Portability through compilers

C and C++ classed as ‘bad languages’

- Memory leaks (failing to free allocated memory)
- Undefined errors (trust the programmer)
- Errors in the program could crash the computer
- Java and C# with garbage collector and JIT

C++ modernized

- STL provides code reusability and memory management through containers, iterators and algorithms
- Auto keyword, safe pointers and type secure validation
- Better developed OS makes it less prone to crashes

C++ today

Modern C++

- Popular programming language (Tiobe second place)
- Top 5 stack overflow (programming languages)

Pros and Cons

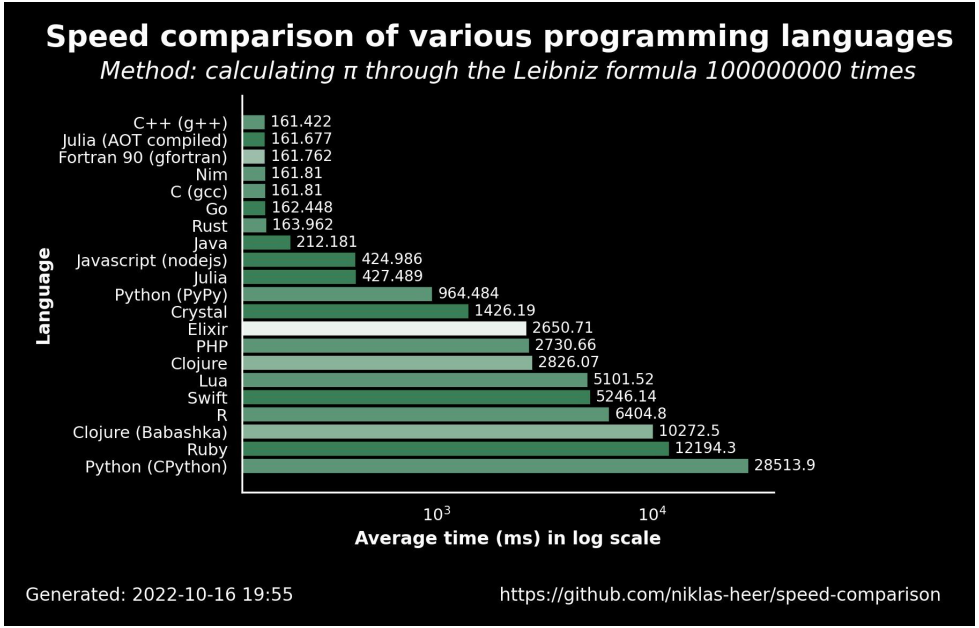
Pros

- Incredibly fast
- Scalable
- Mature
- Both usable in low- and high-level applications

Cons

- Higher threshold for good code
- Long compile times for large projects

Pros and Cons



lafftale for developers

Areas of use

- Heavily loaded transaction systems
  - Banks
  - Logistics
  - Stock Trading Systems
- Performance critical applications
  - Games
  - Graphics
- Embedded Systems
  - Aerospace
  - IoT
  - Automotive

Building a C++ application

C++ files

A program mainly consists of 3 types of files:

- .cpp-files: Contains the source code
- .h-files: Contains declared references used by our .cpp-files
- Build file: Describes how the program should be built

.h files

Declares the functions and variables used in the regarded .cpp file, that may or may not be available to other .cpp files

.cpp files

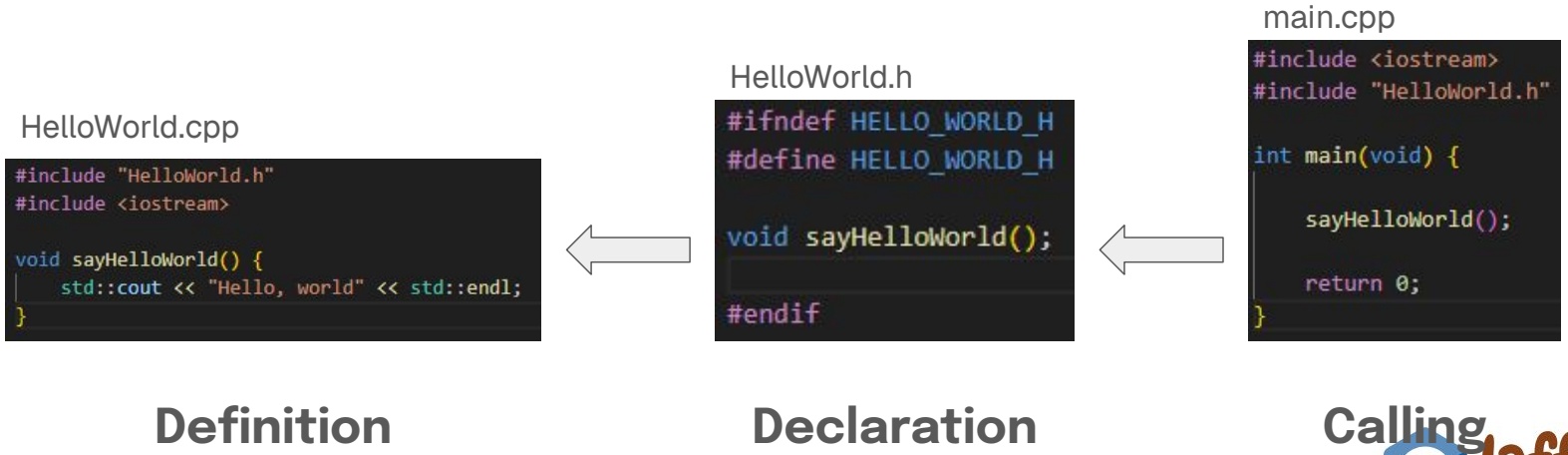
Contains the source code for your program. Amongst other things, defines functions and variables declared in the .h files.

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

void sayHelloWorld() {
    std::cout << "Hello, world" << std::endl;
}
```

Connection

The main function calls the sayHelloWorld function declared in HelloWorld.h and defined in HelloWorld.cpp



lafftale for developers

C++ introduction

- History of C++
- C++ today
- Building a C++ program
- Installing Tools

Background of programming

- Before modern programming
- Creation of OOP
- 1972: C brings programming into modern age
- 1979: C++ / C with classes
- Standardisation of C and C++
- C and C++ classed as ‘bad languages’

Pre-modern programming

- Complicated syntax
- High threshold
- Prone to errors
- Hard to maintain
- Not portable

```
          global    _start

          section   .text
_start:   mov       rax, 1
          mov       rdi, 1
          mov       rsi, message
          mov       rdx, 13
          syscall
          mov       rax, 60
          xor       rdi, rdi
          syscall

          section   .data
message:  db        "Hello, World", 10
```

lafftale for developers

Object-oriented programming

- Way of modelling programs around real world objects
- Early signs found at MIT 1950/1960 AI group
- Some OOP function in ALGOL
- Norwegian Computing Center 1960s - Simula

```
Begin
    OutText ("Hello, World!");
    Outimage;
End;
```

1972: The birth of C

- 1972 at Bell Labs by Dennis Ritchie to create utilities running on Unix.
- Access to memory through pointers instead of commands
- Imperative programming
- Compiled to machine code
- Portability

```c
int main(void) {
    printf("Hello World\n");
    return 0;
}
```

1979: C++ / C with classes

- 1979 Bjarne Stroustrup, Danish Scientist at AT&T Bell Labs
- Extension of C with classes
- Object-oriented programming
- Overloading, References, type-safe memory

lafftale
for developers

Standardisation of C and C++

- C standardised 1989/1990
- C++ standardised 1998

Why is standardisation important?

- Agreement on syntax and standard functions
- Portability through compilers

C and C++ classed as ‘bad languages’

- Memory leaks (failing to free allocated memory)
- Undefined errors (trust the programmer)
- Errors in the program could crash the computer
- Java and C# with garbage collector and JIT

C++ modernized

- STL provides code reusability and memory management through containers, iterators and algorithms
- Auto keyword, safe pointers and type secure validation
- Better developed OS makes it less prone to crashes

Modern C++

- Popular programming language (Tiobe second place)
- Top 5 stack overflow (programming languages)

Pros and Cons

Pros

- Incredibly fast
- Scalable
- Mature
- Both usable in low- and high-level applications

Cons

- Higher threshold for good code
- Long compile times for large projects

Areas of use

- Heavily loaded transaction systems
  - Banks
  - Logistics
  - Stock Trading Systems
- Performance critical applications
  - Games
  - Graphics
- Embedded Systems
  - Aerospace
  - IoT
  - Automotive

C++ files

A program mainly consists of 3 types of files:

- .cpp-files: Contains the source code
- .h-files: Contains declared references used by our .cpp-files
- Build file: Describes how the program should be built

.h files

Declares the functions and variables used in the regarded .cpp file, that may or may not be available to other .cpp files

```
#ifndef HELLO_WORLD_H
#define HELLO_WORLD_H

void sayHelloWorld();

#endif
```

.cpp files

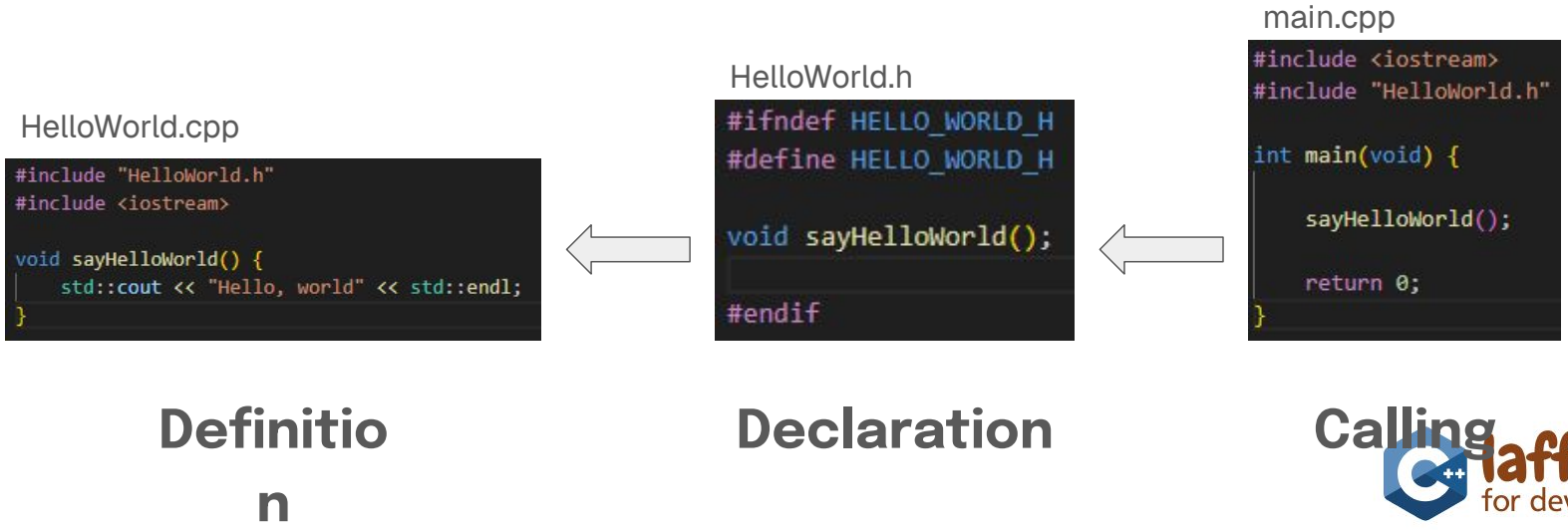
Contains the source code for your program. Amongst other things, defines functions and variables declared in the .h files.

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

void sayHelloWorld() {
    std::cout << "Hello, world" << std::endl;
}
```

Connection

The main function calls the sayHelloWorld function declared in HelloWorld.h and defined in HelloWorld.cpp



Build - Makefile

Dynamic creation of compile commands to build your program.

Speeds up compiling of larger projects.

```
1   PROG = main.exe
2   SRC = main.cpp
3   CFLAGS = -g -std=c++17 -Wall -Werror
4   LIBS =
5   CC=g++
6
7   all: $(PROG)
8
9   $(PROG): $(SRC)
10      $(CC) -o $@ $(CFLAGS) $(LDFLAGS) $(SRC)  $(LIBS)
11
12  clean:
13      rm -f $(PROG)
14
15  .PHONY: all clean
```

lafftale for developers

Link to examples of project structures

Simple project structure (good to start out with)
https://github.com/lafftale1999/cpp_for_developers/tree/main/project_templates/simple_project_template

Intermediate project structure (good for larger projects)
https://github.com/lafftale1999/cpp_for_developers/tree/main/project_templates/intermediate_project_template

Exercises project structure (good since there will be a lot of files):
https://github.com/lafftale1999/cpp_for_developers/tree/main/project_templates/exercise_project_template

lafftale for developers

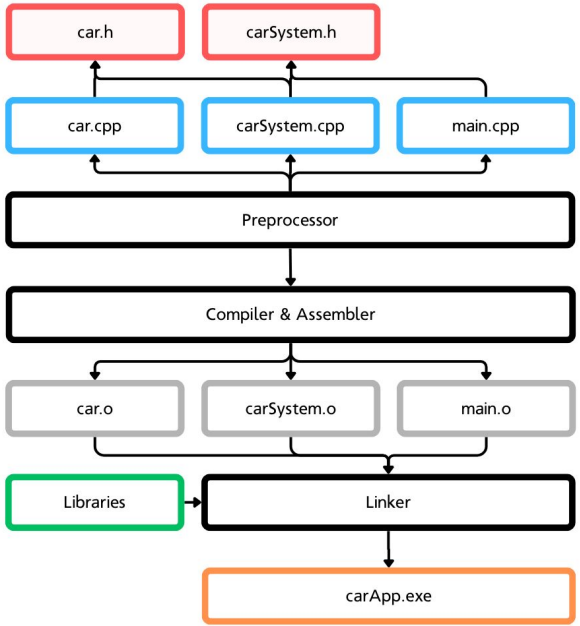
Compiler

C++ is a compiled language, which means it needs to be translated to machine code to run in an environment.

A compiler is a program made to perform the building process of our files, and translate them into runnable code.

Build process

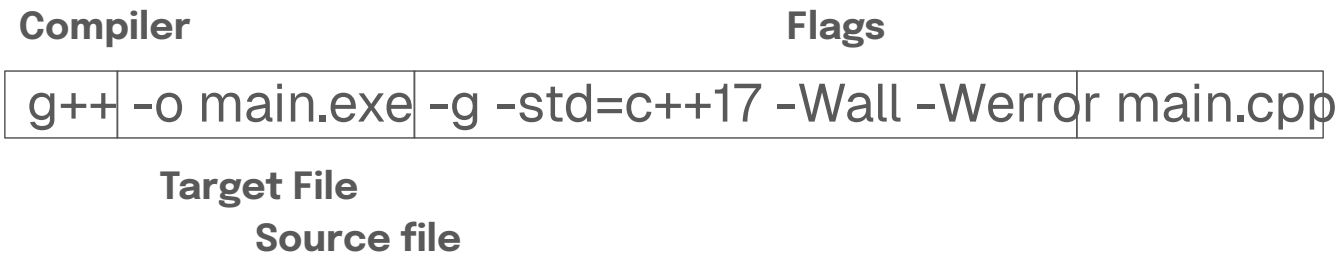
- Preprocessor - prepares the files
- Compiler - .cpp to assembly
- Assembler - assembly to binary .o
- Linker - Links all files together and produces an executable



lafftale for developers

Compile commands

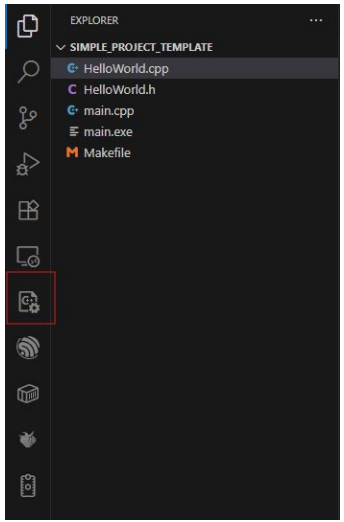
Describing how your program should be built.

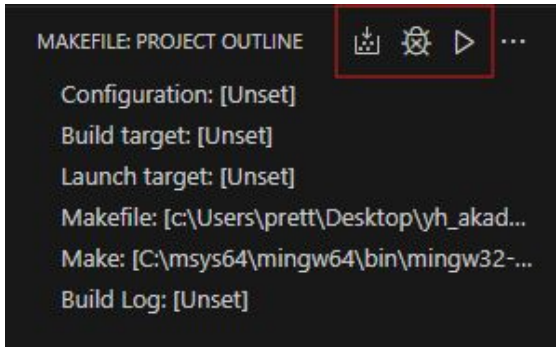


lafftale for developers

When running or debugging your file…

You shall now use the Makefile extension instead when running your code.





- Use “Bucket” for building the program.
- Use bug for debugging
- Use “play” to build and run program.

lafftale for developers

Installing the tools

Install Tools

Download:

- Visual Studio Code
  - Install Makefile Tools extension
  - Install C/C++ extension

Next slides have installation guide for Windows, Mac and Linux.

Windows Installation

Windows Step 1: Installing MSYS

We will be using GCC / G++ compiler, which is a Linux tool. Therefore we need to install tools to support running this in a Windows environment.

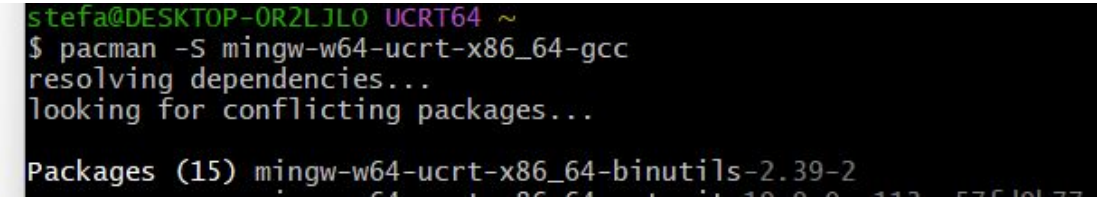
Go to: https://www.msys2.org/

Follow the installation guide for windows. Make sure that you install it on the path C:\msys64.

Windows Step 2: Download GCC

Start up MSYS2 MINGW-64

Run the command:
pacman -S mingw-w64-ucrt-x86_64-gcc

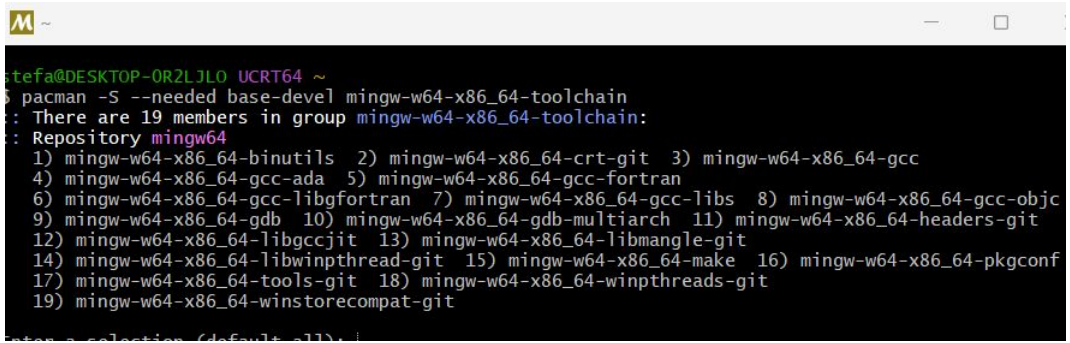


lafftale for developers

Windows Step 3: Download Toolchain

In the same terminal (MSYS2 MINGW-64)

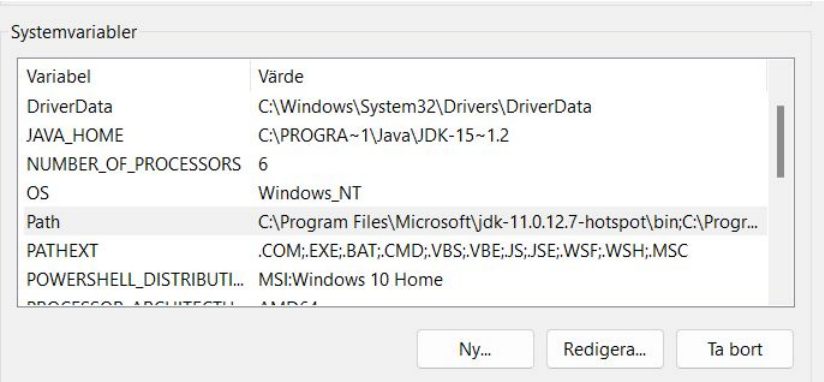
Run the command and accept default on all questions:
pacman -S --needed base-devel mingw-w64-x86_64-toolchain

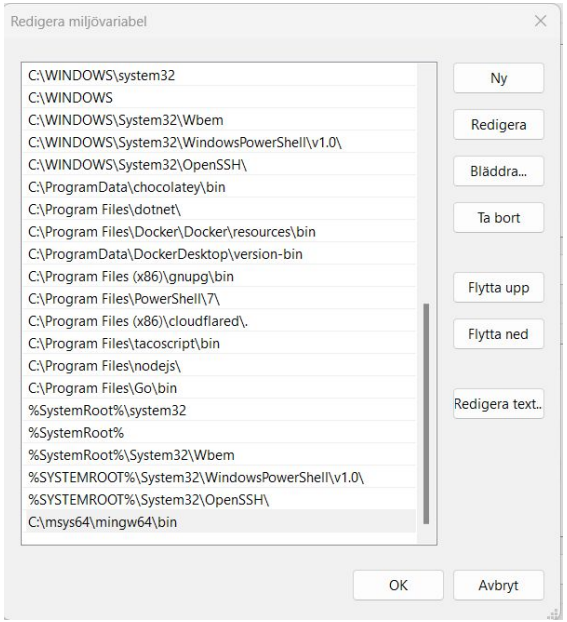


Windows Step 4.1: Add to Environment Variables

- Open “Environment Variables”. Make sure you open the systems and not users.
- Click “Environment Variables”.
- In the “Environment Variables”-tab, select PATH and then click “Edit”
- Add the following path: *C:\msys64\mingw64\bin*
- To make sure it is added properly, click “OK” in all windows you have opened.

Windows Step 4.2: Add to Environment Variables



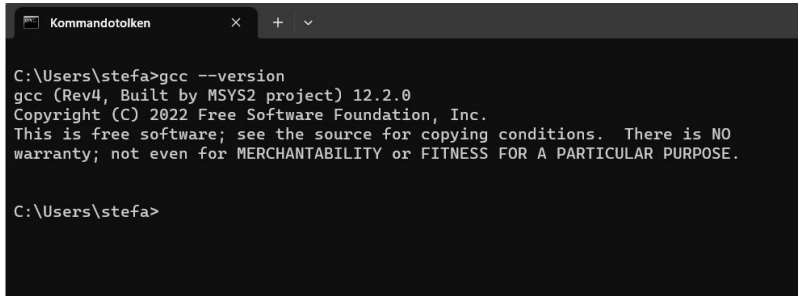


lafftale for developers

Windows Step 5: Check installation

Open your terminal and run the command:
gcc --version

This should give the following result:

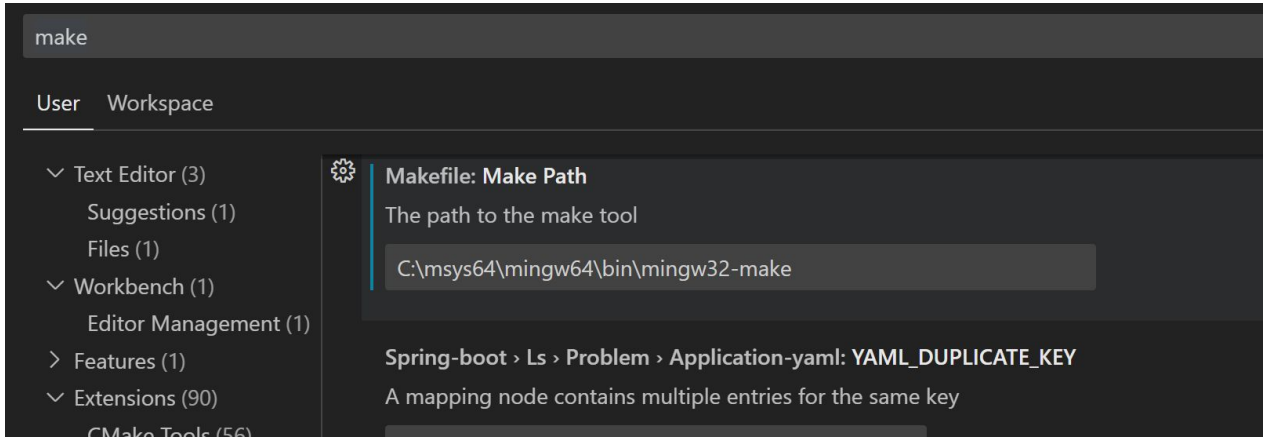


lafftale for developers

Windows Step 6: Add path to Makefile extension

File → Preferences → Settings → Search “make”

Add the path: C:\msys64\mingw64\bin\mingw32-make



lafftale for developers

Mac
Installation

Mac Install Tools

Installing GCC/G++ (Compiler)

1. Install xcode https://www.geeksforgeeks.org/techtips/how-to-install-xcode-command-line-tools/
2. Install Homebrew https://www.geeksforgeeks.org/installation-guide/homebrew-installation-on-macos/
3. Install GCC/G++ https://formulae.brew.sh/formula/gcc

Check by running the commands: “which gcc” and “which g++” to make sure the installation is complete.

lafftale for developers

Linux Installation

Linux Install Tools

Use your distro's package manager to install GCC/G++ and GDB. There are too many distros to specify all of them, search for your distro and "download GCC".

I believe in you!


lafftale
for developers