Lab setup instructions

1 Objective

Set up, on your own computer, the tools we will use during the C Programming module: the GNU C compiler (gcc) and the Visual Studio Code editor. At the end of this guide you will have built and run a first C program in three different ways.

> Estimated time: 45–60 minutes, most of it waiting for downloads. Please complete this guide **before the first lab session**.

2 Requirements

- A personal computer with at least 5 GB of free disk space and 8 GB of RAM.
- Microsoft Windows 11 (W11). The course also works on Linux and macOS; see the section *Linux and macOS users* at the end of this guide.
- Administrative rights may be needed to run the MSYS2 installer.
- An Internet connection.

3 Install the GNU C compiler and associated tools

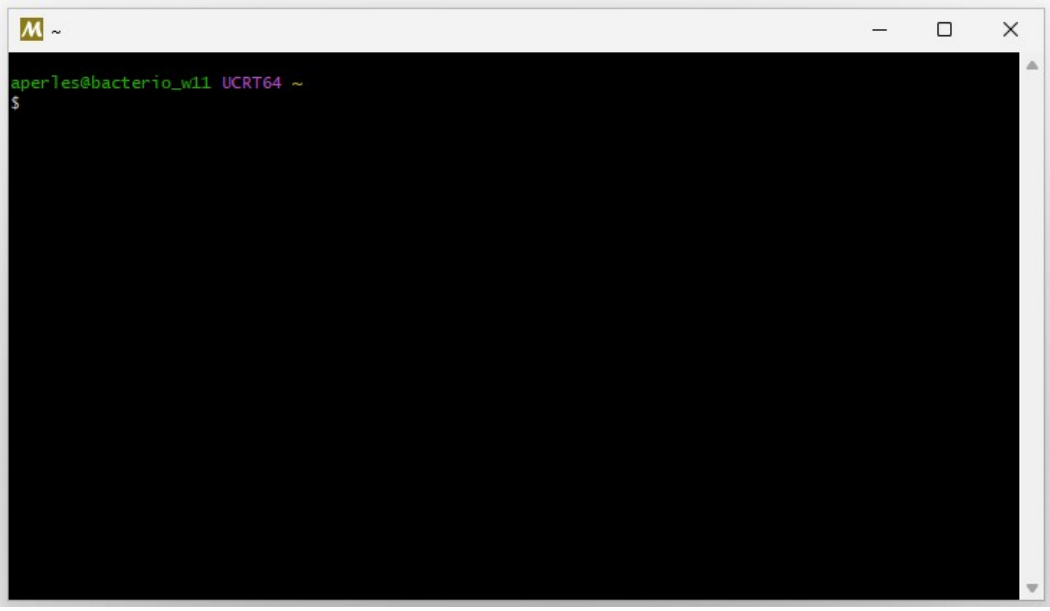
We will use the widely adopted, open and free GNU C/C++ compiler (gcc) and its toolset. GNU tools are native to Linux; on Windows we will use **MSYS2**, a distribution that provides the **MinGW-w64** version of gcc for Windows.

> Linux and macOS users can skip this section and go to *Linux and macOS users* at the end of this guide.

3.1 Install MSYS2

1. Go to the official MSYS2 web page: https://www.msys2.org/
2. Download the installer for Windows (named *msys2-x86_64-YYYYMMDD.exe*).
3. Run the installer (be patient), keep the default installation folder (`C:\msys64`) and, at the end, check the option *Run MSYS2 now*.

A UNIX-like terminal opens. Check that its title bar and prompt show **UCRT64**, as in the following image. You can run typical UNIX commands here.



> Always use the terminal called **MSYS2 UCRT64** (you can find it in the Start menu). Other MSYS2 terminals, such as "MSYS2 MSYS", will not find the right compiler.

3.2 Install the compiler

Install gcc, the gdb debugger and other useful tools with *pacman*, the MSYS2 package manager. Type the following command (do not type the `$` prefix) and press Enter:

```
$ pacman -S --needed base-devel mingw-w64-ucrt-x86_64-toolchain
```

pacman will ask you two questions:

- `Enter a selection (default=all):` press **Enter** to accept all packages.
- `Proceed with installation? [Y/n]` type **Y** and press **Enter**.

Be patient: gcc will be installed among other supporting packages. Then check that the compiler works:

```
$ gcc --version
```

You should see a message starting with `gcc.exe (Rev..., Built by MSYS2 project)` followed by a version number.

3.3 Build a first C program in the MSYS2 terminal

We will keep all the course projects in a folder named `c_projects` inside your Windows user folder (`C:\Users\<your user name>`). This way, the same files are visible from MSYS2, from Windows and from Visual Studio Code.

1. In the MSYS2 UCRT64 terminal, run the following commands. The first one moves you to your Windows user folder.

```
$ cd "$(cygpath -u "$USERPROFILE")"
$ mkdir c_projects
$ cd c_projects
$ mkdir test
$ cd test
$ pwd
```

The last command shows the current folder; it should be something like `/c/Users/<your user name>/c_projects/test`.

2. Create a file named `main.c` with the following code. You can use the *nano* editor (`nano main.c`, then save with Ctrl+O, Enter, and exit with Ctrl+X).

```
#include <stdio.h>

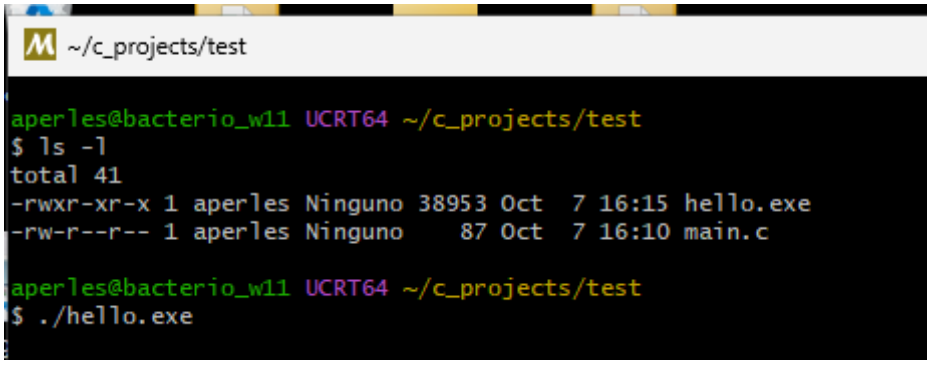
int main(void)
{
    printf("Hello ESIGELEC.\n");
    return 0;
}
```

> If you prefer, you can create the file with Windows Notepad in `C:\Users\<your user name>\c_projects\test`. In the *Save as* dialog, choose *Save as type: All files (*.*)*; otherwise Notepad saves the file as `main.c.txt`. To see file extensions in Windows, open File Explorer and select *View → Show → File name extensions*.

3. Build the program. `hello` is the name we choose for the executable file. The options `-Wall -Wextra` ask the compiler to warn you about suspicious code: always read the warnings, they usually point to real bugs.

```
$ gcc -Wall -Wextra -o hello main.c
```

4. List the files in the folder with `ls -l` and run the program by typing `./hello.exe`:



> Use the MSYS2 terminal to install tools, but run your programs from the Windows command prompt or from Visual Studio Code. In the MSYS2 terminal, text printed with `printf` may not appear before a `scanf` waits for input.

3.4 Make gcc available from Windows

To use gcc from any Windows program (the command prompt, Visual Studio Code...), add its folder to the `Path` environment variable of your user account:

1. Open the Start menu, type *environment* and select **Edit environment variables for your account**.
2. In the *User variables* list, select `Path` and click **Edit...**.
3. Click **New** and type `C:\msys64\ucrt64\bin` exactly as shown, without quotation marks.
4. Click **OK** in both dialogs.
5. Close any open command prompt and Visual Studio Code window. Programs only see the new `Path` when they are started again; there is no need to reboot.

3.5 Build the program from the Windows command prompt

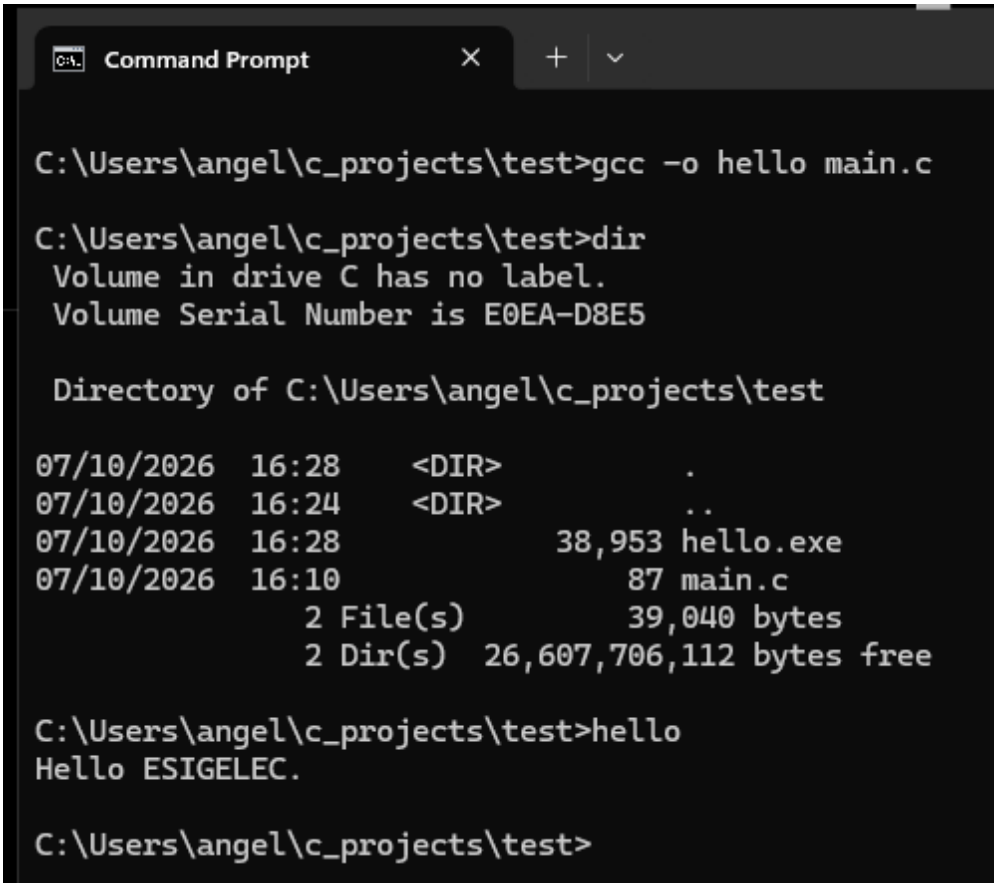
1. Open a Windows command prompt: Start menu, type *cmd*, press Enter.
2. Check that gcc is found:

```
C:\...> gcc --version
```

3. Go to the folder of the previous example, delete the old executable and build it again:

```
C:\...> cd %USERPROFILE%\c_projects\test
C:\...> del hello.exe
C:\...> gcc -Wall -Wextra -o hello main.c
```

4. List the files with `dir` and run the program by typing `hello`:



Done! The compiler is ready.

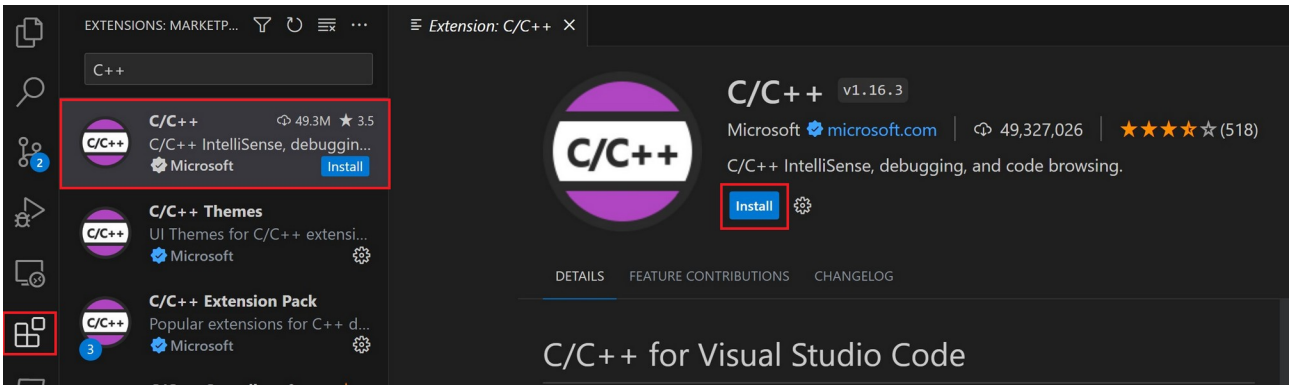
4 Install Microsoft Visual Studio Code (VS Code)

VS Code is a full-featured code editor widely used by the programming community. It has many options, but we will only need a few of them.

1. Download VS Code from https://code.visualstudio.com/ for your operating system.
2. Run the installer and follow the instructions (the default options are fine).

To work with C, install the official C/C++ extension:

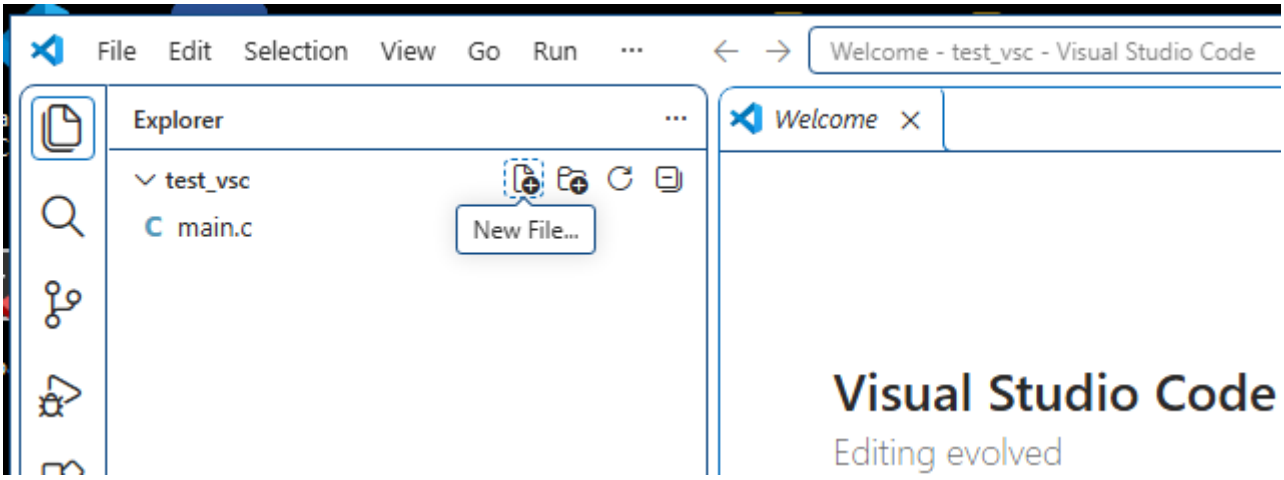
1. Open VS Code.
2. Open the Extensions view (Ctrl+Shift+X), search for *C++* and install **C/C++** by Microsoft.



4.1 Open the first project

In VS Code, a project is a folder that contains your code, the build instructions, etc. We will start from the previous example.

1. In File Explorer, copy the folder `c_projects\test` as `c_projects\test_vsc`. Inside `test_vsc`, delete `hello.exe` and keep `main.c`.
2. In VS Code, select *File → Open Folder...* and open the `test_vsc` folder. If VS Code asks whether you trust the authors of the files, select *Yes, I trust the authors*, since you created this folder yourself.

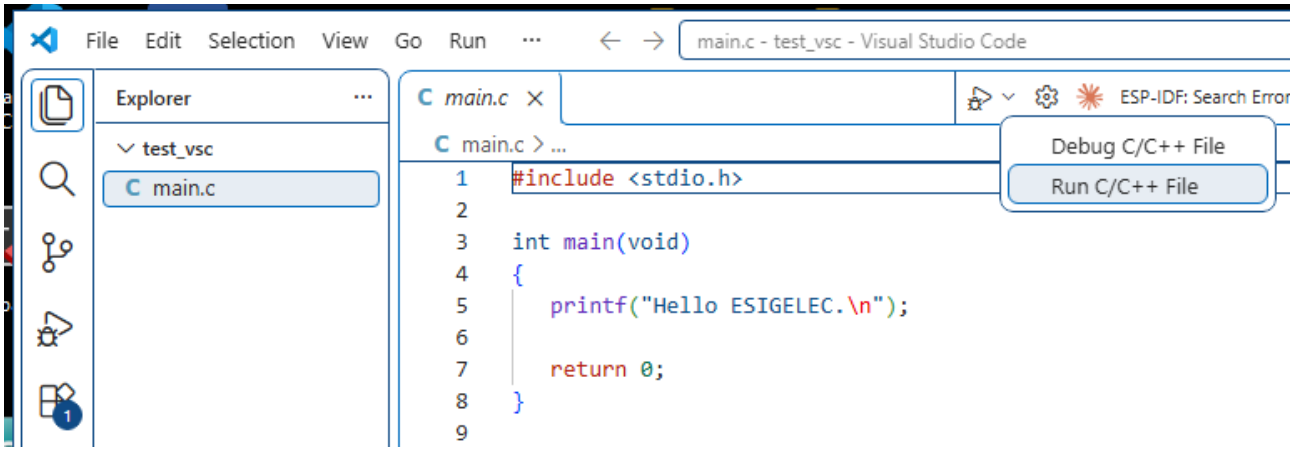


The *Explorer* panel shows the folder and the C file. You can create additional files with the *New File...* icon or from the *File* menu.

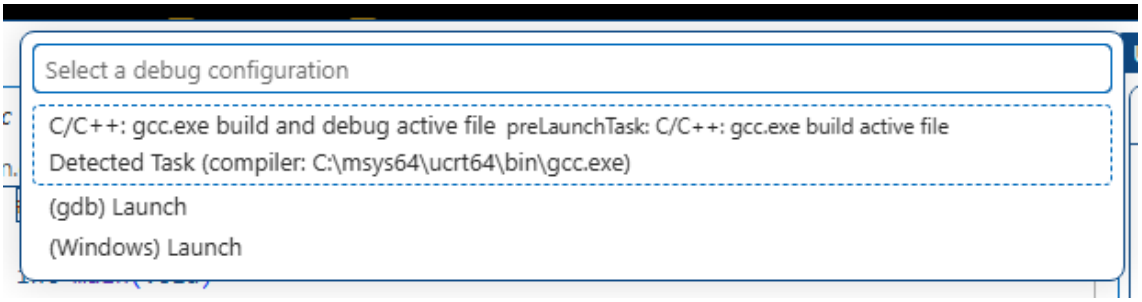
4.2 Build and run the program

To build and run the project, VS Code needs some extra information. We will let VS Code generate it automatically [1].

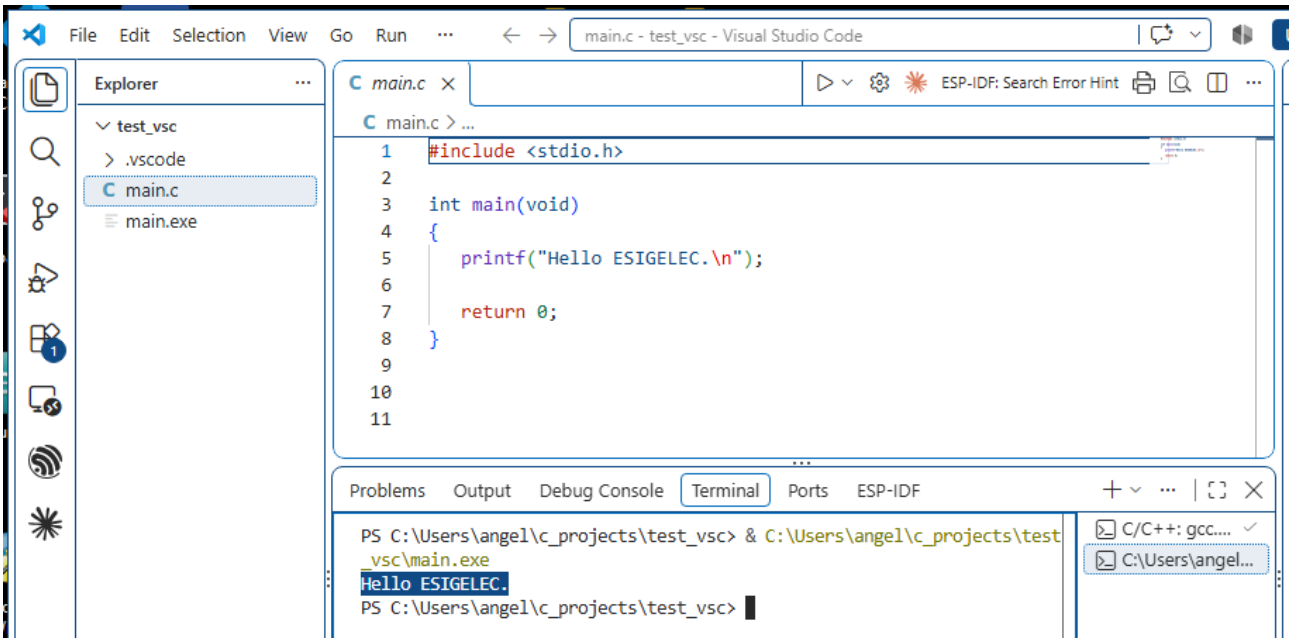
1. Open `main.c` so that it is the active file.
2. Click the arrow next to the play button in the top right corner of the editor and choose **Run C/C++ File**.



3. Choose **C/C++: gcc.exe build and debug active file**. This configuration was detected automatically; VS Code will not ask again for this project.



4. VS Code builds the program and runs it in the *Terminal* panel. You should see `Hello ESIGELEC.`



VS Code has created a `.vscode` folder in your project. It contains project-specific configuration files; here only `tasks.json`, which holds the instructions to build your project.

4.3 Enable compiler warnings in VS Code

Open `.vscode/tasks.json` and add `"-Wall"` and `"-Wextra"` to the `args` list, right after `"-g"`:

```
"args": [
    "-fdiagnostics-color=always",
    "-g",
    "-Wall",
    "-Wextra",
    "${file}",
    "-o",
    "${fileDirname}\\${fileBasenameNoExtension}.exe"
],
```

Save the file and run the program again to check that everything still works.

> This configuration builds only the active file. Later in the course, when a program is split into several `.c` files, we will modify `tasks.json` accordingly.

4.4 Try the debugger

The debugger lets you run a program line by line and inspect its variables. It will be very useful when we study pointers.

1. Click to the left of the line number of the `printf` line. A red dot (a *breakpoint*) appears.
2. Click the arrow next to the play button and choose **Debug C/C++ File**.
3. The program stops at the breakpoint. Press F10 to execute the next line and F5 to continue to the end.

5 Checklist

You are ready for the first lab session if:

- ☐ `gcc --version` works in a Windows command prompt.
- ☐ `hello.exe` builds and runs from the command prompt.
- ☐ `main.c` builds and runs from VS Code with the play button.

> Take a screenshot of each item and submit them as indicated by your instructor before the first lab session.

6 Troubleshooting

- **'gcc' is not recognized as an internal or external command.** Check that `C:\msys64\ucrt64\bin` is in your user `Path`, with no typing errors and no quotation marks, and that you closed and reopened the command prompt or VS Code.
- `gcc: command not found` **in MSYS2.** You are probably in the "MSYS2 MSYS" terminal. Open **MSYS2 UCRT64** instead.
- `hello.exe` **disappears or cannot be run.** Some antivirus programs block newly built executables. Add your `c_projects` folder to the antivirus exclusions.
- **Strange errors with paths.** Your Windows user name may contain spaces or accented characters (e.g. `C:\Users\Hélène Dupont`), or your user folder may be synchronized by OneDrive. In that case, create the `c_projects` folder directly in `C:\` (`C:\c_projects`) and use that location throughout this guide.
- **A command copied from this document fails.** Type it by hand: some PDF viewers alter quotes and dashes when copying.

7 Linux and macOS users

The tools are the same, but the compiler is installed differently. Then follow the steps of this guide using your system terminal and paths.

- **Linux (Debian/Ubuntu):** `sudo apt install build-essential gdb`
- **macOS:** `xcode-select --install`. On macOS, the `gcc` command is actually the clang compiler, which is fine for this course. Debugging in VS Code uses the lldb debugger.

On both systems, the executable has no `.exe` extension: run it with `./hello`.

8 References

[1] Using GCC with MinGW. https://code.visualstudio.com/docs/cpp/config-mingw
[2] MSYS2. https://www.msys2.org/