Systems Programming

Laboratory 1 – Programming environment

During the semester the compiler that is going to be used is **gcc**. It compiles C/C++ source code into executables.

Although it can be executed from the command line it is advantageous to use an IDE (Integrated Development Environment).

The IDE seamless integrates the writing of the code the compilation, error finding, debugging and execution all inside a single environment.

The IDE that is suggested is the **Microsoft Visual Code (VSCode).**

VSCode has versions to Linux, MAC OS X and Windows.

In the case of Linux and Mac OS X, VSCode uses the installed compilers.

In Windows it can use the Windows Subsystem for Linux (that replicates a Linux OS inside Windows).

The installation of VSCode is straightforward and its configuration to compile C/C++ code is also simple.

It is only necessary to follow the documentations provided by Microsoft.

In the middle of this document there is two small exercises for you to solve.

Table of Contents

1 Install Visual Studio Code

The way to install Visual Studio Code is described in

- https://code.visualstudio.com/docs/setup/setup-overview

This simple tutorial guides the installation on the three different operating systems: Linux, MAC OS X and Windows

After installation, students should follow the GET STARTED guided tour to get acquainted with VSCode:

- https://code.visualstudio.com/docs/getstarted/introvideos

2 Install C++ Extensions

In order to compile C programs it is necessary to install the C++ extension.

2.1 Linux

If the student is using Linux, it is necessary to install the gcc compiler and the necessary libraries. In **Ubuntu** this is done with the command:

```
    sudo apt-get install build-essential gdb
```

in other Linux distributions the command can be different :/

All the necessary installation steps are presented in this guide:

- https://code.visualstudio.com/docs/cpp/config-linux

After installing all the dependencies guarantee that everything is running smoothly by executing the example presented in the previous guides.

This example is in C++, but if you can run it, you can run C programs also.

2.2 Windows

https://learn.microsoft.com/pt-pt/windows/wsl/install

https://marketplace.visualstudio.com/items?itemName=ms-vscode-remote.remote-wsl

https://code.visualstudio.com/docs/cpp/config-wsl

In Windows it is necessary to install:

- Windows Subsystem for Linux
  - https://learn.microsoft.com/pt-pt/windows/wsl/install
  - install and configure one Linux distribution inside the WSL (for instance Ubuntu)
- Install vscode in windows
  - https://code.visualstudio.com/docs/cpp/config-wsl
  - install the **Remote-WSL** extension inside VSCode

All the necessary installation steps are presented in this guide:

- https://code.visualstudio.com/docs/cpp/config-wsl

After installing all the dependencies guarantee that everything is running smoothly by executing the example presented in the previous guides.

This example is in C++, but if you can run it, you can run C programs also.

2.3 MAC OS X

https://code.visualstudio.com/docs/cpp/config-clang-mac

In MAC OS X it is necessary to guarantee that you have the Xcode installed

All the necessary installation steps are presented in this guide:

- https://code.visualstudio.com/docs/cpp/config-clang-mac

After installing all the dependencies guarantee that everything is running smoothly by executing the example presented in the previous guides.

This example is in C++, but if you can run it, you can run C programs also.

It is fundamental that, independently on the Operating System used (LINUX, WINDOWS+WSL or MAC OS X), students complete one of the following tutorials:

https://learn.microsoft.com/pt-pt/windows/wsl/install

https://code.visualstudio.com/docs/cpp/config-wsl

https://code.visualstudio.com/docs/cpp/config-clang-mac

3 Debug C programs

https://code.visualstudio.com/docs/cpp/cpp-debug

The VS Code allows debugging of C program. It allows the regular execution of applications with the added functionalities:

- If the application crashes it is possible to evaluate the state of the application
- it is possible to insert breakpoints
- it is possible to execute the application step by step
- it is possible to verify and change the values of variables
- it is possible to execute python expressions that will change the state of the applications

Don’t forget to follow the debugging guide:

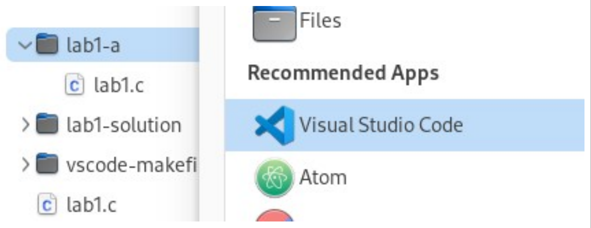
- https://code.visualstudio.com/docs/cpp/cpp-debug

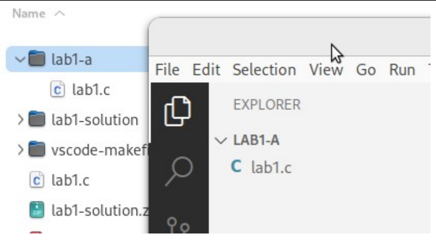
4 Exercise 1

In this exercise students will use VS Code to edit, compile and run a program.

Follow the next steps:

- Create a folder and inside it edit the lab1.c file
    - you can create a file called **lab1.c** and copy to it the C code
    - copy the **lab1.c** code into the suitable directory
- open the VSCode in such directory





- Compile the program
- Run the program
    - Type your student number
    - Take note of the printed value

Why does the random function always return the same number?
Is **random()** really random?

Debug the program line by line to understand if everything is ok!

5 Use of makefiles

https://marketplace.visualstudio.com/items?itemName=ms-vscode.makefile-tools

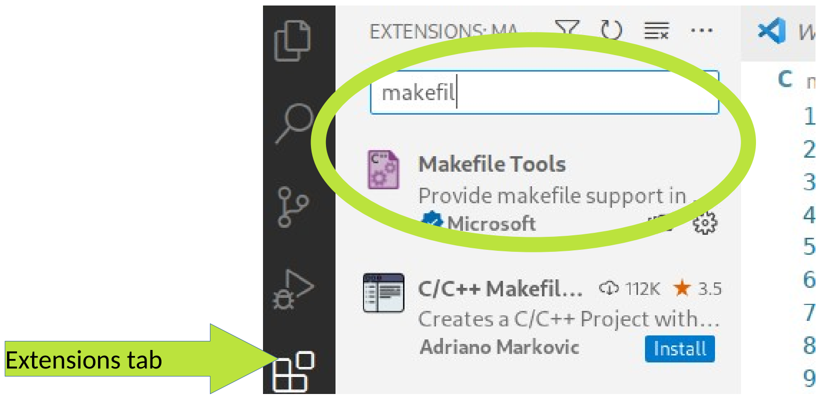
The Visual Studio Code can use a standard makefiles to help compile a project and run applications in a project. To accomplish this, it is necessary first install an extension, then write the suitable makefile and finally configure the extension.

5.1 Installation of Makefile extension

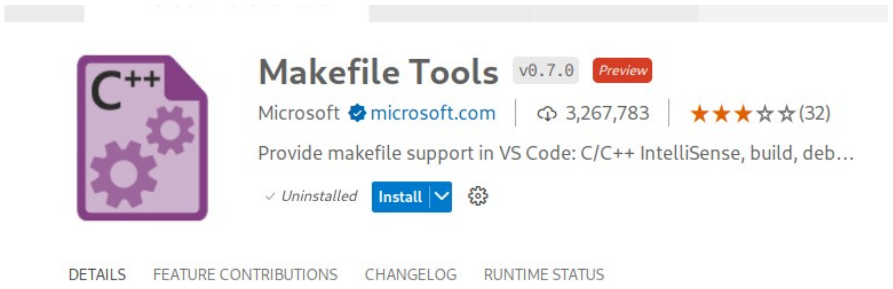
In order to activate the make and correct debug buttons on VSCode, it is necessary to install the **Makefile Tools** extension from Microsoft.

https://marketplace.visualstudio.com/items?itemName=ms-vscode.makefile-tools

Select the **Extensions tab** and search for the **Makefile Tools** from Microsoft

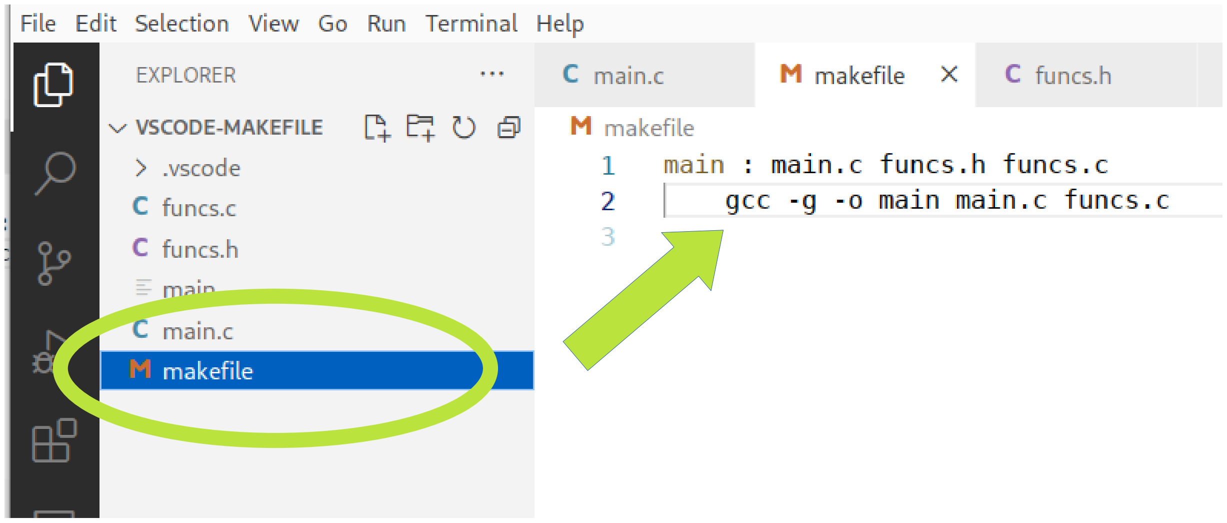


After selecting the correct extension, install it



5.2 Makefile

After this extension is installed, students should create a Makefile file with rules to compile the various components and applications.



```
main : main.c funcs.h funcs.c
	gcc -g -o main main.c funcs.c
```

To ease the development the following required rules should also be defined:

- all
- clean

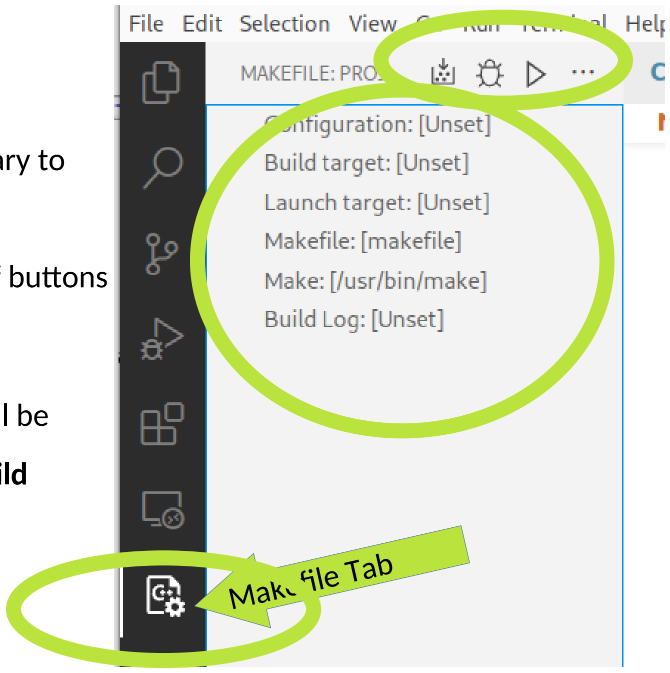
In order to allow the debug of the applications, it is necessary to use the **-g** compilation flag.

5.3 Configuration of extension

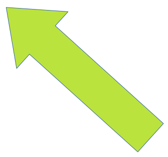
To configure the Makefile extension and compile/debug the applications it is necessary to access the Makefile tab.

This tab show the configuration and a set of buttons to compile and run the applications.

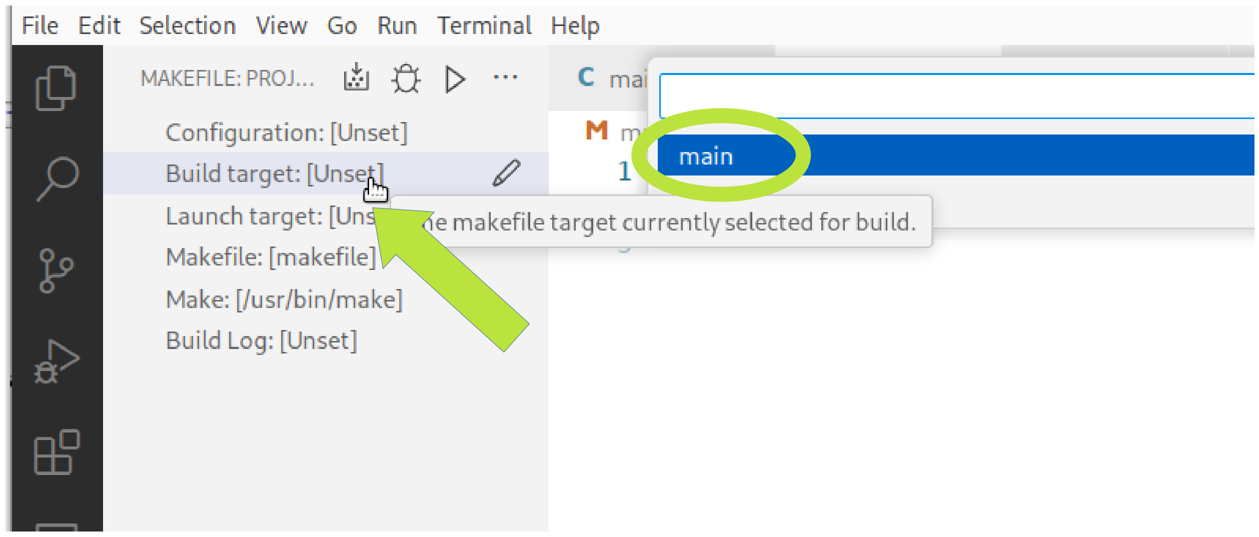
In order to define what what application will be compiled, the student should define the **Build Target** configuration:



Configuration: [Unset]
Build target: [Unset]
Launch target: [Unset]
Makefile: [makefile]
Make: [/usr/bin/make]
Build Log: [Unset]



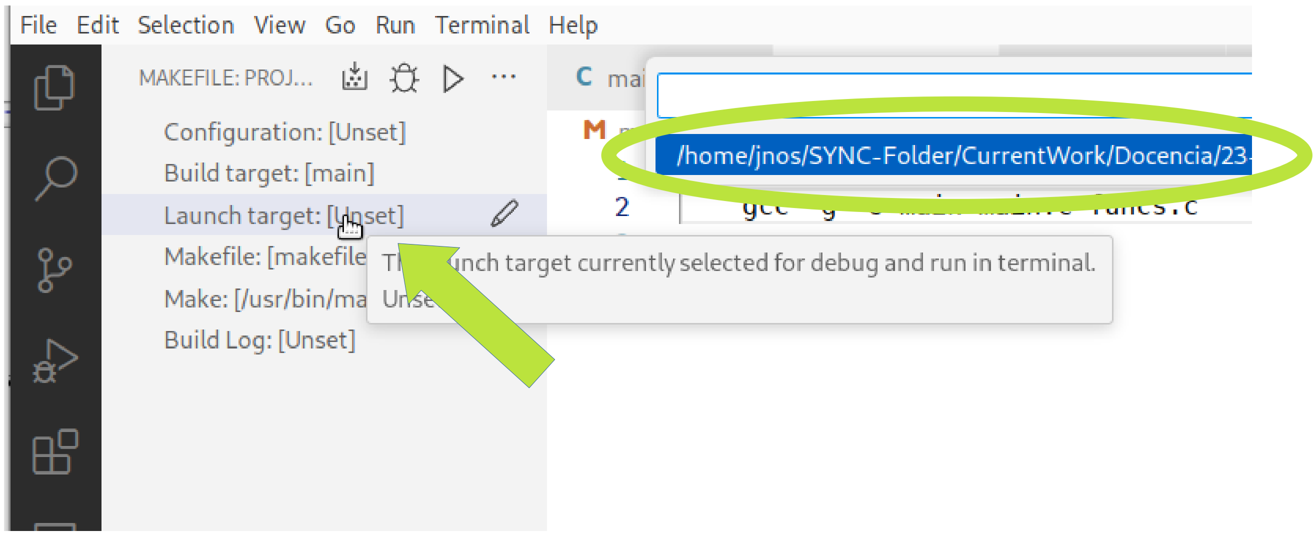
Configure Build target:



Later, when the user clicks on the compile button, the Make command will try to execute the selected rule (in this case compile the **main** application.

To run the application inside VSCode or debug it it is also necessary to configure the Launch target.

Configure the Launch target:


File Edit Selection View Go Run Terminal Help
MAKEFILE: PROJ...
Configuration: [Unset]
Build target: [main]
Launch target: [Unset]
Makefile: [makefile
Make: [/usr/bin/ma
Build Log: [Unset]
/home/jnos/SYNC-Folder/CurrentWork/Docencia/23-
unch target currently selected for debug and run in terminal.

After the configuration it is now possible to

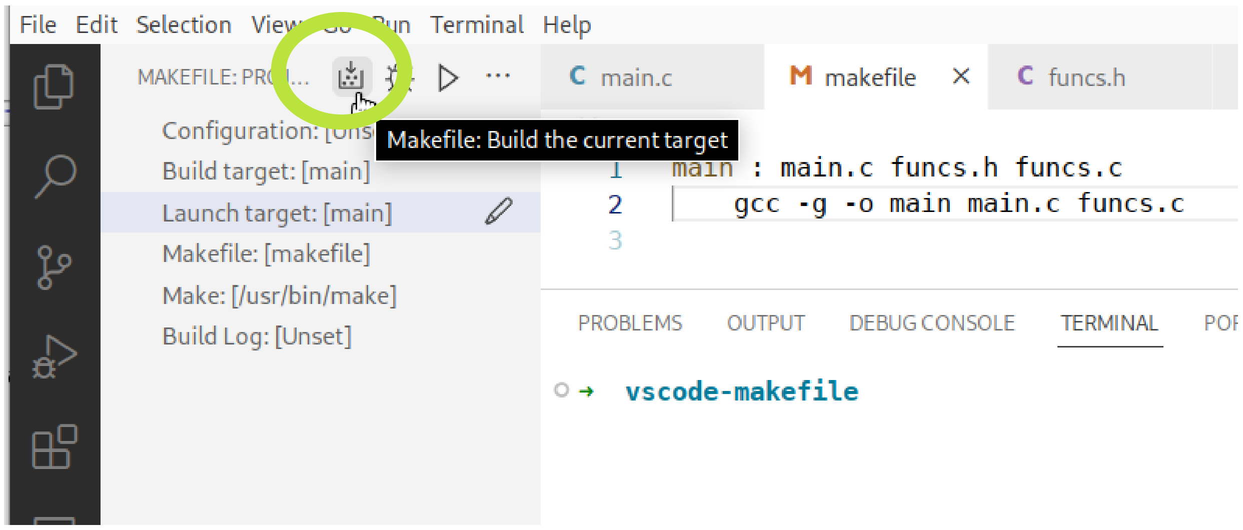
- compile the target application

- run (launch) the target application
- debug the target application

using the provided buttons.

Whe compiling it is possible to see the compilation/linking errors and correct them.

Build the current target:


File Edit Selection View Go Run Terminal Help
MAKEFILE: PR...
main.c
makefile
funcs.h
Makefile: Build the current target
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

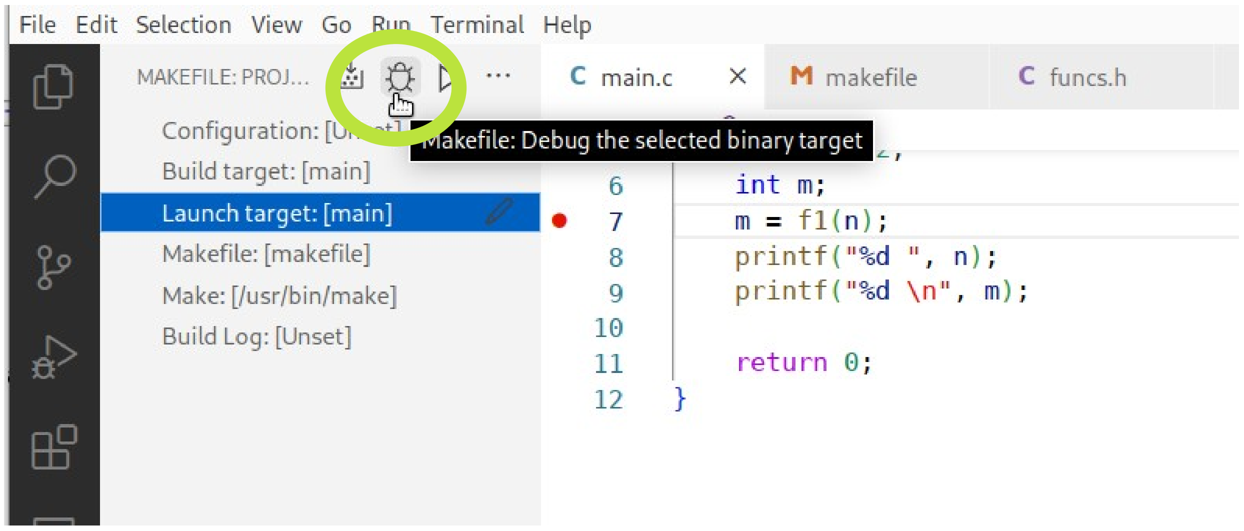
```
Configuration: [Uns
Build target: [main]
Launch target: [main]
Makefile: [makefile]
Make: [/usr/bin/make]
Build Log: [Unset]
```

```
main : main.c funcs.h funcs.c
	gcc -g -o main main.c funcs.c
```

```
→ vscode-makefile
```

To debug the target program, students should define the suitable breakpoints and run the program in the debugger.

Debug the current target:

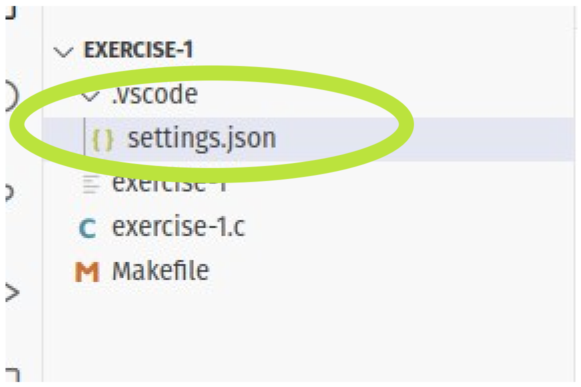

File Edit Selection View Go Run Terminal Help
MAKEFILE: PROJ...
main.c
makefile
funcs.h
Makefile: Debug the selected binary target

```
Configuration: [Unset]
Build target: [main]
Launch target: [main]
Makefile: [makefile]
Make: [/usr/bin/make]
Build Log: [Unset]
```

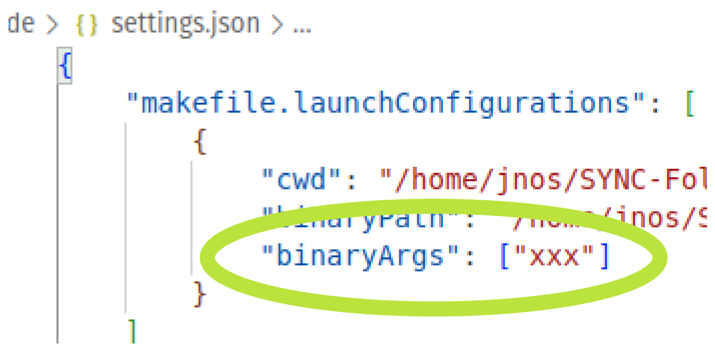
```
    int m;
    m = f1(n);
    printf("%d ", n);
    printf("%d \n", m);

    return 0;
}
```

To debug programms that require command line arguments it is necessary to define them in the settings.json file:



The arguments for the debugged applications should be writte in the **binaryArgs** array:



```
{
    "makefile.launchConfigurations": [
        {
            "cwd": "/home/jnos/SYNC-Fol
            "binaryPath": "/home/jnos/S
            "binaryArgs": ["xxx"]
        }
    ]
```

6 Exercise 2

Create a Makefile for the program of Exercise 1 and verify the various commands to compile, execute and debug inside VSCode.

7 Collaborate while programming

https://docs.microsoft.com/en-us/visualstudio/liveshare/use/vscode

https://docs.microsoft.com/en-us/visualstudio/liveshare/quickstart/share

https://docs.microsoft.com/en-us/visualstudio/liveshare/quickstart/join

https://docs.microsoft.com/en-us/visualstudio/liveshare/quickstart/browser-join

The Visual Studio Code offers a set of tools that allow multiple user to interact in the same workspace and files.

This allows remote users (in different computers) to control the same workspace:

- All users see the same interface
- All users can change in real time the same file.
- All users can execute the applications and interact with them

To activate this functionality it is necessary to install the Visual studio live share.

Follow the instructions in:

- https://docs.microsoft.com/en-us/visualstudio/liveshare/use/vscode

You will be required to sign in. You can use your TEAMS/Técnico account to do so.

After sharing a session, the remote partner can access your work from his installed VSCode or even from the browser:

- https://docs.microsoft.com/en-us/visualstudio/liveshare/quickstart/join
- https://docs.microsoft.com/en-us/visualstudio/liveshare/quickstart/browser-join

Now all participants can cooperate on the same workspace in real time.

If necessary, an audio connection can be established to help partners to communicate.