

Ambiente de trabalho

Este guia serve para ajudar a criar o ambiente de trabalho para a realização dos exercícios práticos e do projeto. Para tal são necessárias as seguintes ferramentas:

- Editor de texto. Qualquer editor de texto pode ser usado. p.e. VSCode, Notepad++, Gedit, etc.
- gcc – Compilador e “Linker” - O compilador é um programa que transforma o programa escrito em linguagem C (código fonte) em código máquina que pode ser executado no CPU. O “Linker” é o programa que agrega o código compilado com bibliotecas do sistema operativo para produzir o programa executável.
- make – É uma ferramenta que controla a produção de executáveis a partir do código fonte. Esta ferramenta é particularmente útil para programas com múltiplos ficheiros de código fonte que devem ser compilados individualmente e posteriormente “linkados”.

Ambas as ferramentas são abertas e disponibilizadas para Linux (ver ponto 4). Para executar em computadores com SO Windows 10 é necessário utilizar o WSL (Windows Subsystem for Linux). Em alternativa também se pode utilizar uma máquina virtual com uma distribuição de Linux instalada ou o cygwin.

Este guia é para computadores com o **Windows 10 versão 2004 ou posterior (Build 19041 ou posterior)** ou Windows 11 e necessita de uma ligação à rede. (Em computadores com versões anteriores do Windows 10, podem seguir as instruções em <https://docs.microsoft.com/en-us/windows/wsl/install-manual>)

1) Abrir uma linha de comando ou powershell como administrador.

No menu WinX (pressionando as teclas Windows+x) seleccionar Windows PowerShell (Admin)

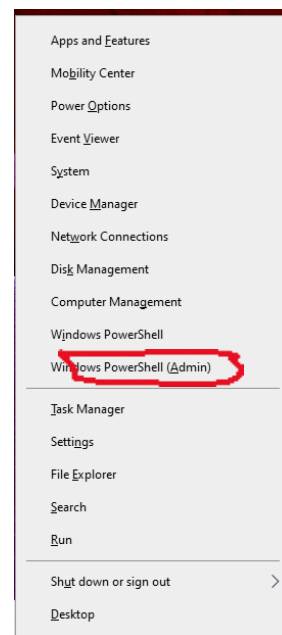
2) Instalar o WSL

Execute o comando “wsl –install”.

```
Administrator: Windows PowerShell
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\WINDOWS\system32> wsl --install
Installing: Virtual Machine Platform
Virtual Machine Platform has been installed.
Installing: Windows Subsystem for Linux
Windows Subsystem for Linux has been installed.
Downloading: WSL Kernel
Installing: WSL Kernel
WSL Kernel has been installed.
Downloading: Ubuntu
The requested operation is successful. Changes will not be effective until the system is rebooted.
PS C:\WINDOWS\system32>
```



3) Reiniciar o computador

Após reiniciar a instalação continua com o Linux Ubuntu que pode demorar alguns minutos.



Introduzir o UNIX username e definir a password. (Não tem de ser o username nem password do utilizador do Windows 10)

Na linha de comandos execute “sudo apt update” para atualizar o Ubuntu.

4) Instalar GCC e Make

Na linha de comandos do WSL execute o comando:

```
$ sudo apt install build-essential gdb make wget ca-certificates
```

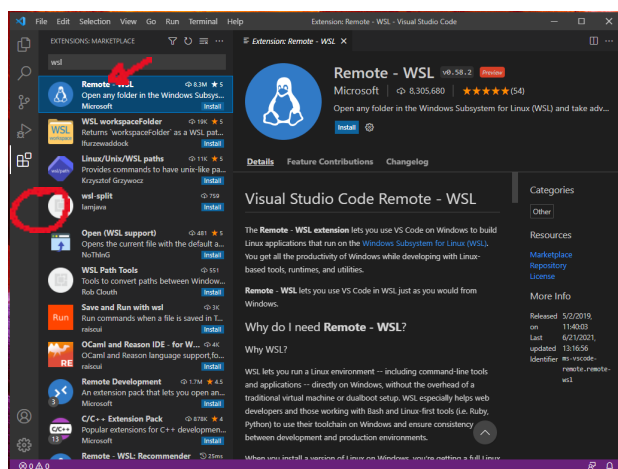
Confirme que o gcc está instalado executando o comando:

```
$ gcc --version
```

5) Instalar e configurar VSCode no Windows

Descarregar o VSCode de <https://code.visualstudio.com>, aceitar a licença e instalar com a opções “Add to PATH” selecionada.

Abrir o VSCode e instalar a extensão “Remote – WSL”



6) Fechar o VSCode e o Ubuntu

Hello World, GCC and VSCode

Com o ambiente de trabalho preparado. Podemos fazer um pequeno programa para verificar que o compilador e editor de texto estão devidamente configurados.

- Abrir o Ubuntu no start menu do Windows 10
- Criar uma pasta para guardar o trabalho

Na linha de comandos:

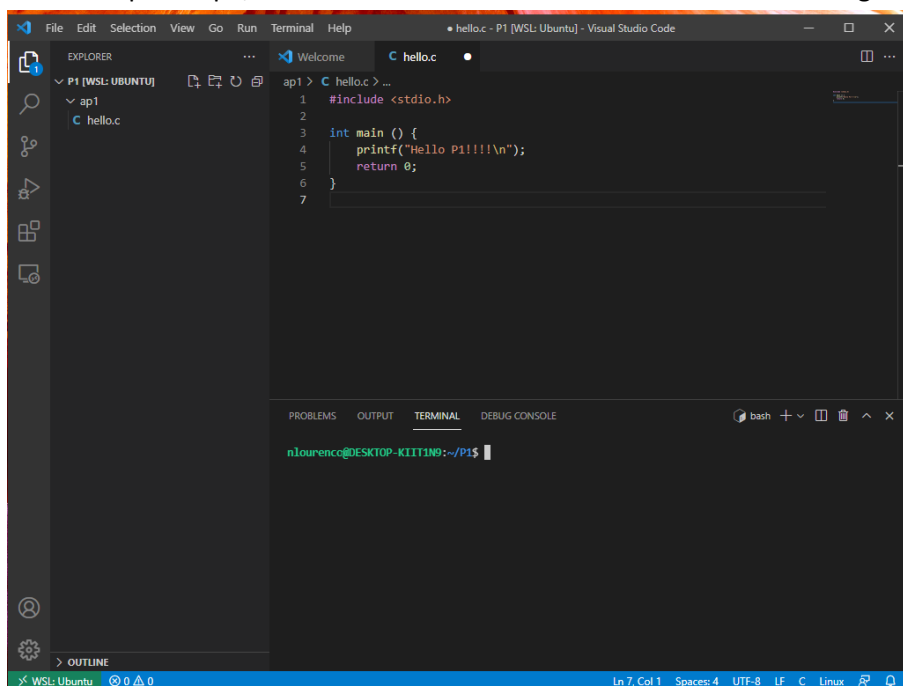
```
$mkdir p1
```

Executar o vscode

```
$code p1
```

```
nlourenco@DESKTOP-KIIT1N9: ~
nlourenco@DESKTOP-KIIT1N9:~$ mkdir p1
nlourenco@DESKTOP-KIIT1N9:~$ code p1
nlourenco@DESKTOP-KIIT1N9:~$
```

- No VSCode abra uma janela de terminal (Terminal->New Terminal).
- Crie uma pasta chamada "ap1"
 - No terminal: mkdir ap1
 - No menu de contexto no painel Explorer (botão direito do rato nesse painel):
- Dentro da pasta ap1, criar um novo ficheiro chamado "hello.c" com o seguinte conteúdo:



- Compilar, linkar e executar o programa hello.
No terminal, criar uma pasta para guardar o código compilado chamada "bin"
\$mkdir bin

Compilar o código fonte guardando o código compilado (*object code*) na pasta “bin”

```
$gcc -c -o bin/hello.o ap1/hello.c
```

Linkar o programa e criar o executável hello

```
$gcc -o hello bin/hello.o
```

Executar o programa

```
$. /hello
```

```

1 #include <stdio.h>
2
3 int main () {
4     printf("Hello P1!!!!\n");
5     return 0;
6 }
7
nlorenco@DESKTOP-KIIT1M9:~/P1$ mkdir bin
nlorenco@DESKTOP-KIIT1M9:~/P1$ gcc -c -o bin/hello.o ap1/hello.c
nlorenco@DESKTOP-KIIT1M9:~/P1$ gcc -o hello bin/hello.o
nlorenco@DESKTOP-KIIT1M9:~/P1$ ./hello
Hello P1!!!!
nlorenco@DESKTOP-KIIT1M9:~/P1$

```

g) Criar uma Makefile para simplificar a compilação.

Criar um ficheiro chamado Makefile com o seguinte conteúdo (as linhas precedidas com # são comentários e não necessitam de ser copiadas)

```

# Makefile defines dependencies and rules to build the targets.
# Basic syntax:
# target : dependency(ies)
#     build rule
#
# $@ is the target
# $^ is the dependencies

#builds executable hello from bin/hello.o
#if hello.o is newer that hello
hello: bin/hello.o
    gcc -o $@ $^

#compiles ap1/hello.c into object bin/hello.o
#if hello.c is newer that hello.o
bin/hello.o: ap1/hello.c | bin
    gcc -c -o $@ $^

#creates bin folder if it does not exists
bin:
    mkdir $@

clean:
    rm -rf hello bin

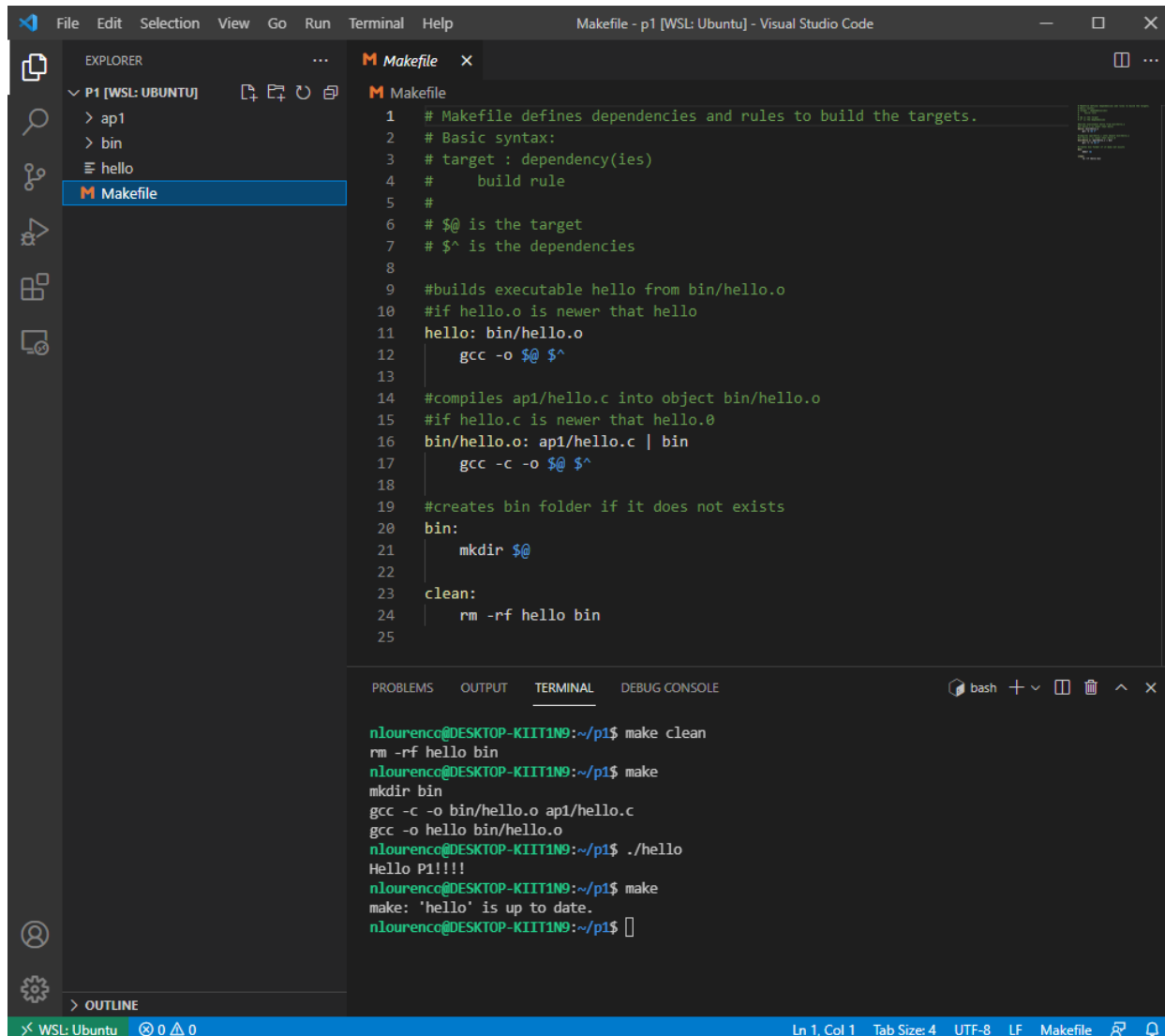
```

No terminal executar o comando `make clean` para eliminar os resultados da compilação anterior.

Compilar novamente com o comando `make`.

Executar o programa compilado com o comando `./hello`.

Se executar o comando `make` novamente, nada será compilado pois o código fonte não tem alterações.



The screenshot shows the Visual Studio Code interface with a Makefile open in the editor and a terminal window at the bottom. The Makefile defines a 'hello' target that compiles 'ap1/hello.c' into 'bin/hello.o' and then into an executable 'bin/hello'. It also includes a 'clean' target to remove the compiled files. The terminal shows the execution of 'make clean', 'make', and './hello', resulting in the output 'Hello P1!!!!'.

```

1 # Makefile defines dependencies and rules to build the targets.
2 # Basic syntax:
3 # target : dependency(ies)
4 #     build rule
5 #
6 # $@ is the target
7 # $^ is the dependencies
8
9 #builds executable hello from bin/hello.o
10 #if hello.o is newer that hello
11 hello: bin/hello.o
12 |
13 | gcc -o $@ $^
14
15 #compiles ap1/hello.c into object bin/hello.o
16 #if hello.c is newer that hello.o
17 bin/hello.o: ap1/hello.c | bin
18 |
19 | gcc -c -o $@ $^
20
21 #creates bin folder if it does not exists
22 bin:
23 |
24 | mkdir $@
25
26 clean:
27 |
28 | rm -rf hello bin
  
```

```

nlourenco@DESKTOP-KIIT1N9:~/p1$ make clean
rm -rf hello bin
nlourenco@DESKTOP-KIIT1N9:~/p1$ make
mkdir bin
gcc -c -o bin/hello.o ap1/hello.c
gcc -o hello bin/hello.o
nlourenco@DESKTOP-KIIT1N9:~/p1$ ./hello
Hello P1!!!!
nlourenco@DESKTOP-KIIT1N9:~/p1$ make
make: 'hello' is up to date.
nlourenco@DESKTOP-KIIT1N9:~/p1$
  
```