

C programming: installing the programming environment

Version: 2021-01

This document details the process of getting started creating console-based C programs. Console-based programs are quite simple - they take input from the keyboard (or a file stored on disk) and they produce output to the text-based console window.

You will need two things to create C programs:

- A text editor to write the source code.
- A compiler to convert the source code into an executable file so the program can be run.

This document is a compilation from information found on the internet. No originality is claimed.

The focus of this document is on computers running Windows.

Contents

Windows (Visual Studio approach)	2
Command line	3
Visual Studio IDE	5
Visual Studio 2019 flow	5
WSL (Windows Subsystem for Linux)	10
Get WSL	10
X (running graphical programs on WSL)	11
MobaXterm	12
Xming	13
Installing the compiler	14
Optional packages	14
Windows and WSL (Visual Studio Code approach)	16
Install WSL	16
Install VS Code	16
Remote – WSL extension	16
Test the installation	18

Windows (Visual Studio approach)

Source: <https://www.cs.auckland.ac.nz/~paul/C/Windows/> (accessed 2020-12-11)

An easy way to get started with C programming on Windows is to install Visual Studio. There are different versions of Microsoft's Visual Studio - the Community version is free and includes all of the necessary libraries and tools that we need to develop C programs.

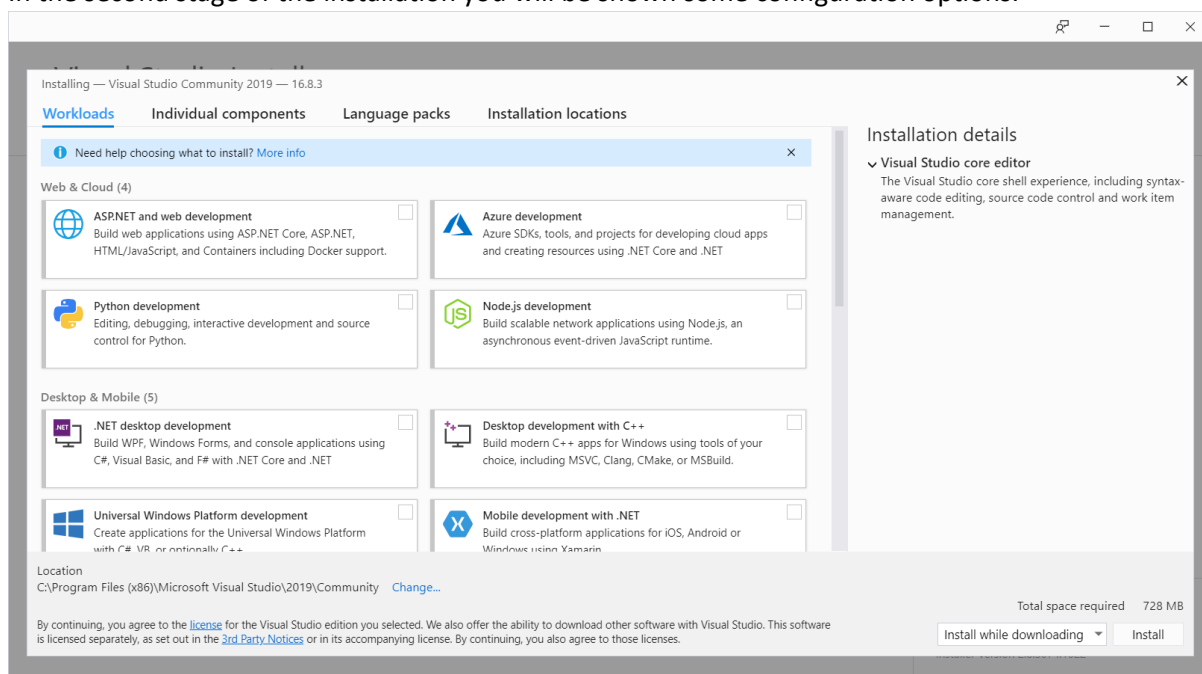


Search for *Microsoft Visual Studio Community version*

(<https://visualstudio.microsoft.com/vs/community/>)

Steps:

- Save the file (called `vs_community_<nnnnnn>.exe`) somewhere convenient.
- Double-click this downloaded file to launch the installer program
- In the second stage of the installation you will be shown some configuration options.



There are several tabs along the top - you only need to look at the default tab which is called *Workloads*.

- Make sure that you click the *Desktop development with C++* option
- Make sure that the *VC++ 20nn Version ... tools* and *Windows 10 SDK* options are both selected before continuing
- Click *Install*

A complete description of the installation can be found at <https://docs.microsoft.com/en-us/visualstudio/install/install-visual-studio?view=vs-2019> (accessed 2020-12-11)

Once installed, the C compiler can be used.

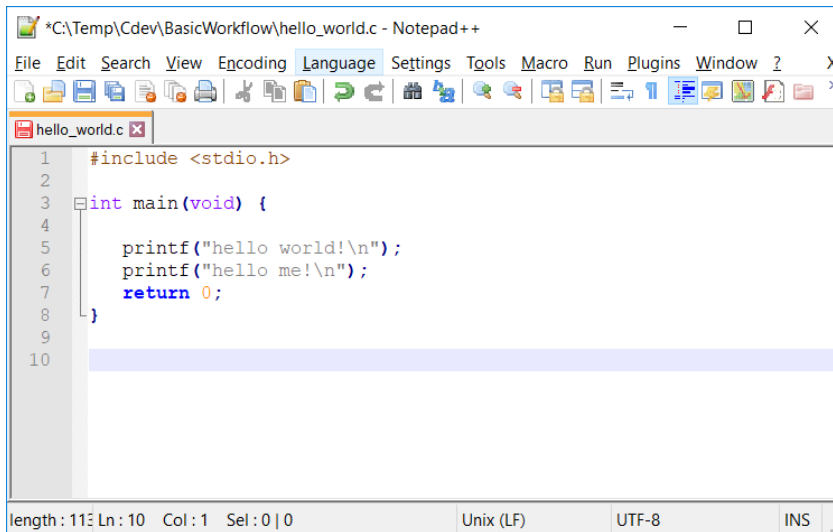
Prerequisites:

- Visual Studio (Community version)

Command line

In the first approach:

1. Use a basic text editor (you can use any editor you like: Notepad, Notepad++, etc.) to write the source code.

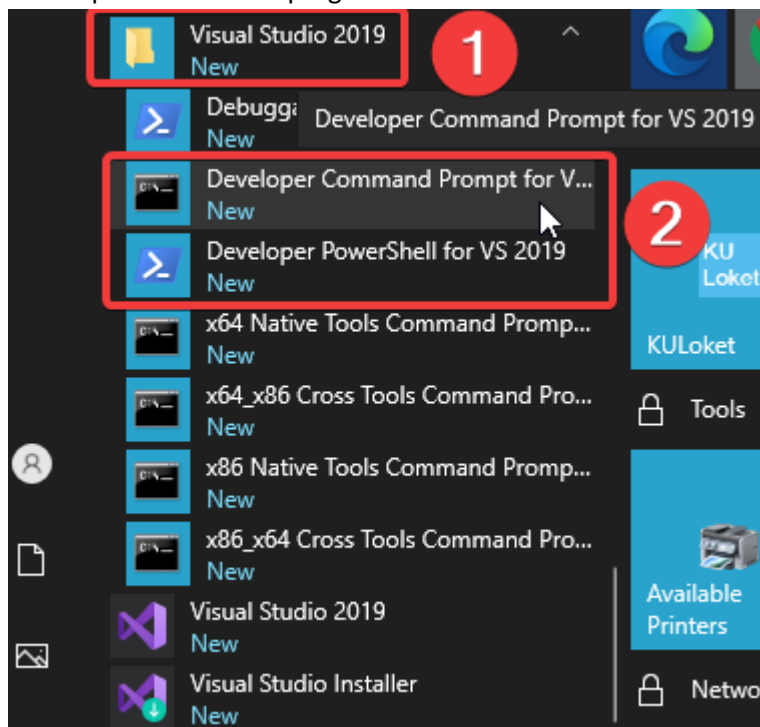


The screenshot shows the Notepad++ application window with the file 'hello_world.c' open. The code is as follows:

```
1 #include <stdio.h>
2
3 int main(void) {
4     printf("hello world!\n");
5     printf("hello me!\n");
6     return 0;
7 }
8
9
10
```

The status bar at the bottom indicates 'length: 113 Ln: 10 Col: 1 Sel: 0 | 0', 'Unix (LF)', 'UTF-8', and 'INS'.

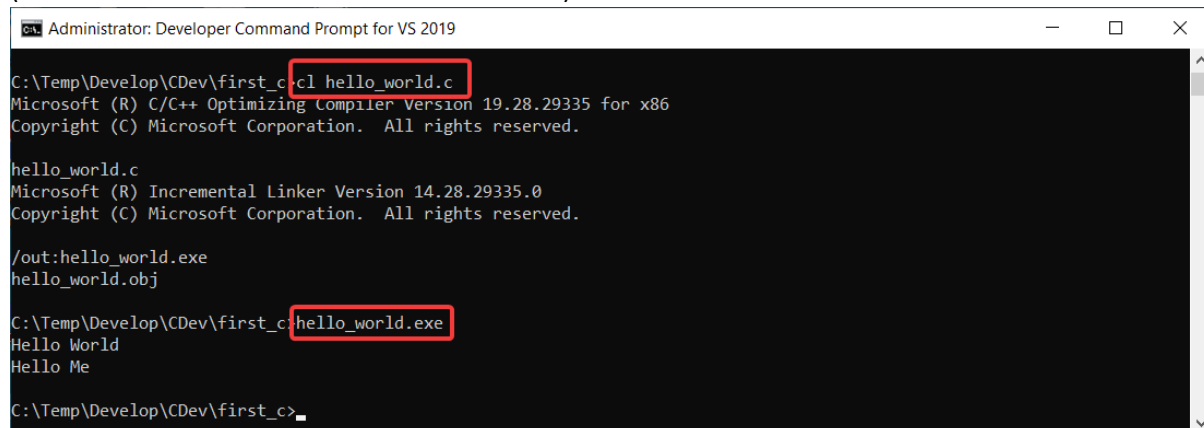
2. Use the *Developer Command Prompt* or *PowerShell* - a command line based environment - to compile and run the program.



Or search for *Developer Command Prompt*

The *Developer Command Prompt* is part of the Visual Studio Community software and automatically sets the correct paths for the compiler and tools, and for any required headers and libraries.

Much like you can use `gcc` on Linux, Visual Studio has a command to be used from the command prompt (it must be the Visual Studio Developer Command Prompt though): `cl` (make sure the source file has the extension `.c`)



```

Administrator: Developer Command Prompt for VS 2019

C:\Temp\Develop\CDev\first_c>cl hello_world.c
Microsoft (R) C/C++ Optimizing Compiler Version 19.28.29335 for x86
Copyright (C) Microsoft Corporation. All rights reserved.

hello_world.c
Microsoft (R) Incremental Linker Version 14.28.29335.0
Copyright (C) Microsoft Corporation. All rights reserved.

/out:hello_world.exe
hello_world.obj

C:\Temp\Develop\CDev\first_c>hello_world.exe
Hello World
Hello Me

C:\Temp\Develop\CDev\first_c>

```



On KU Leuven managed computers: run command line as administrator.



Get an overview of the different compiler options at <https://docs.microsoft.com/en-us/cpp/build/reference/compiler-options-listed-alphabetically?view=vs-2019> (accessed 2020-12-11) or type `cl -help` in the Developer Command Prompt.

Common options are:

- `/out:<file.exe>` - Set output file name.
- `/Zi` - Add debugging symbols to the executable
- `/c` - Doesn't link generating `*.exe` or `*.dll`, it creates only intermediate object code for further separate linking. It is useful for compiling large code bases where each compilation unit can be compiled separately.
- `/W4` - Set the level of warning to the highest.
- `/Wall` - Enables all warnings, including warnings that are disabled by default.

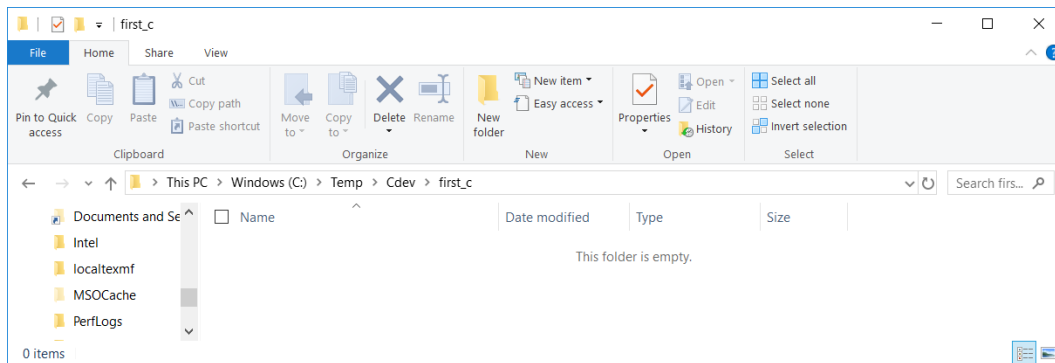
The source file can be edited, re-compiled and ran again.

See Also: <https://github.com/MicrosoftDocs/cpp-docs/blob/master/docs/build/walkthrough-compile-a-c-program-on-the-command-line.md>

Visual Studio IDE

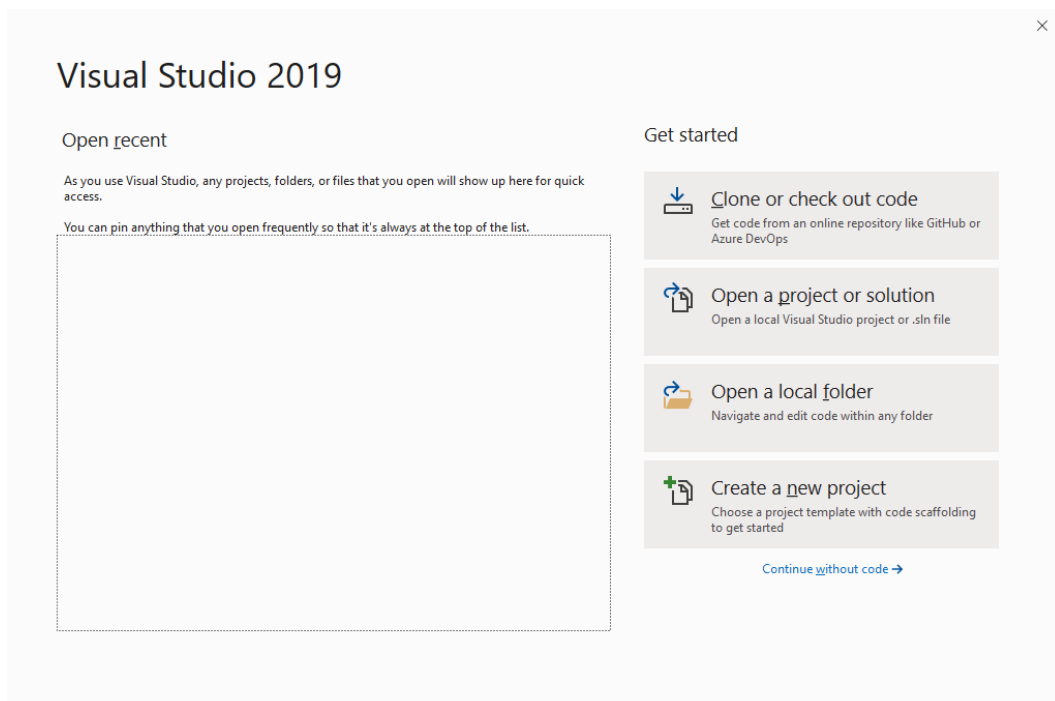
In this approach, the integrated features of Visual Studio Community will be used. The source code will be written using the Visual Studio Community code editor and then the code will be compiled and ran from within the Visual Studio Community environment.

Visual Studio organizes programs into *Projects*. So we are going to create a new Visual Studio project in a folder called *first_c* on the computer. This folder can be created in Windows explorer.

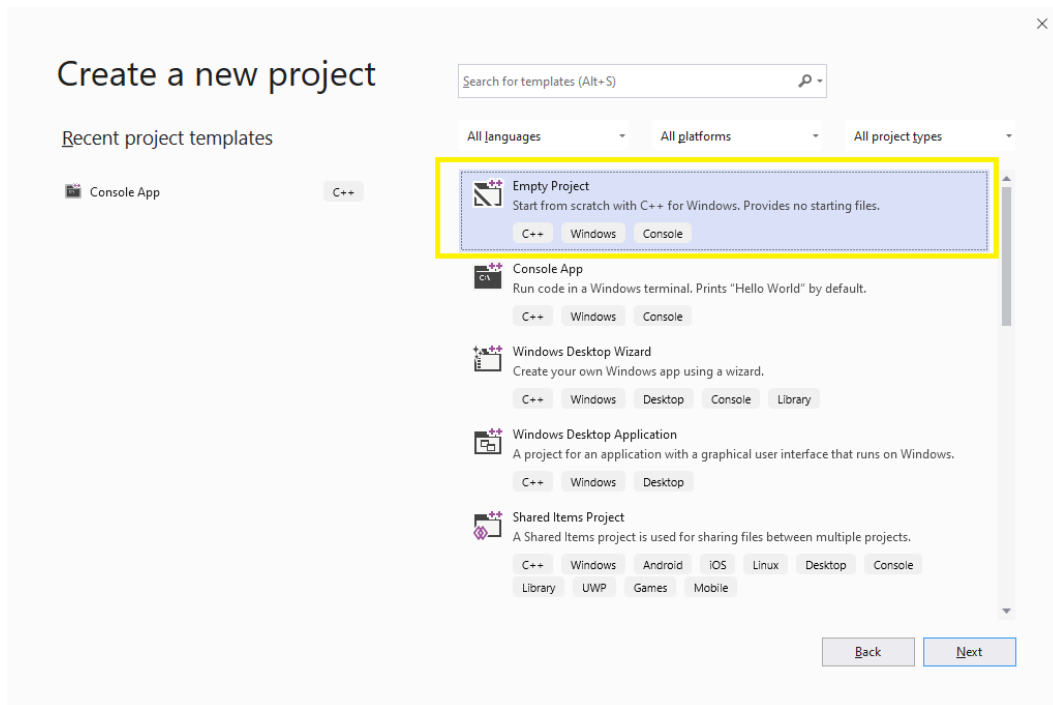


Visual Studio 2019 flow

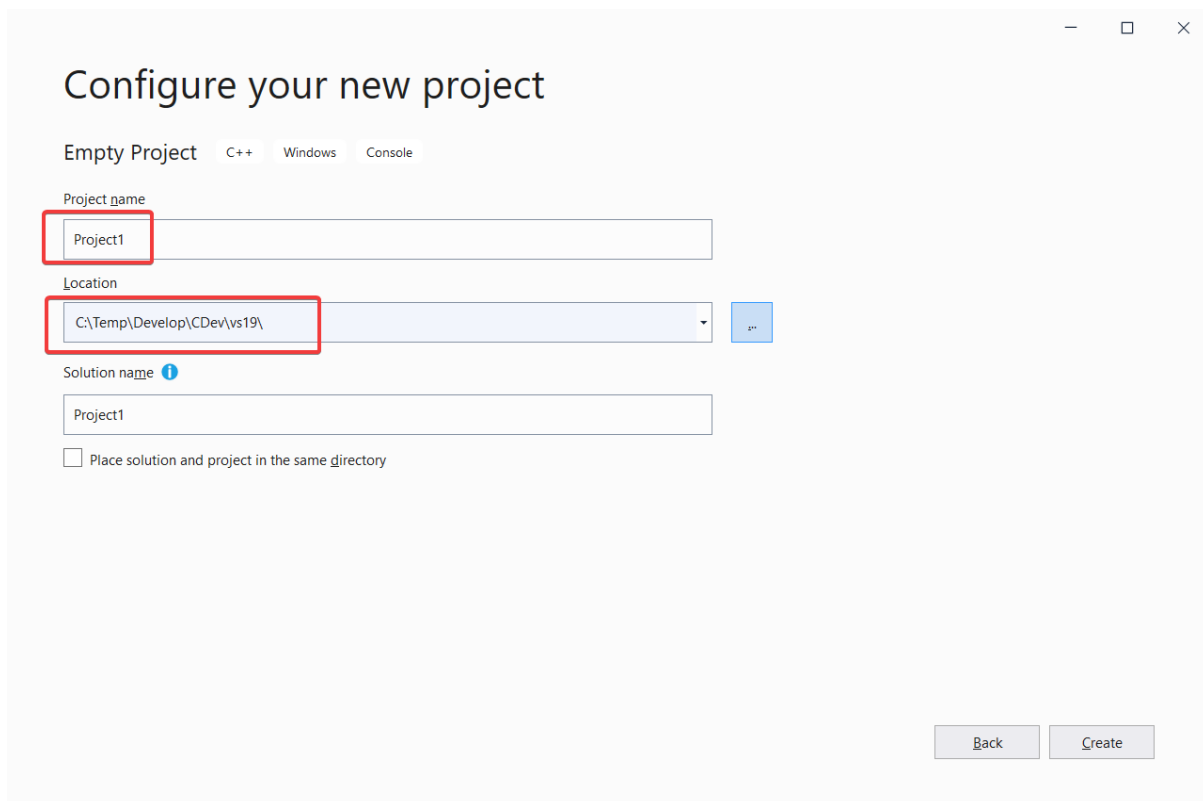
Launch Visual Studio and create a new project.



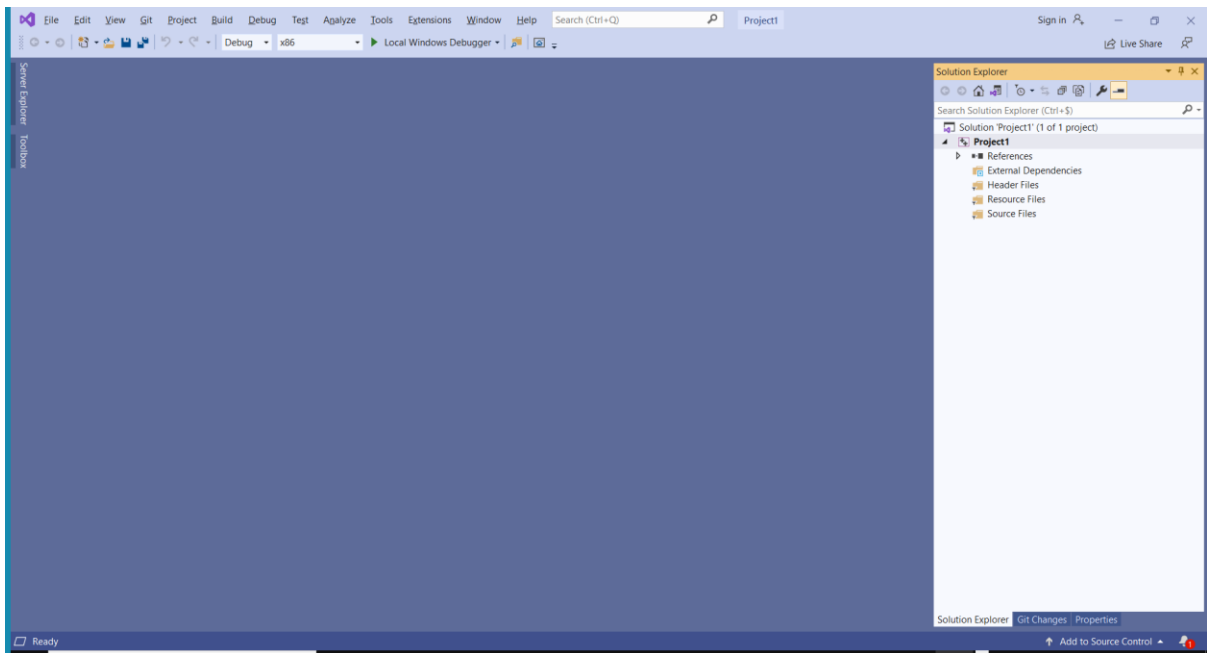
Select the Empty Project template



Click Next

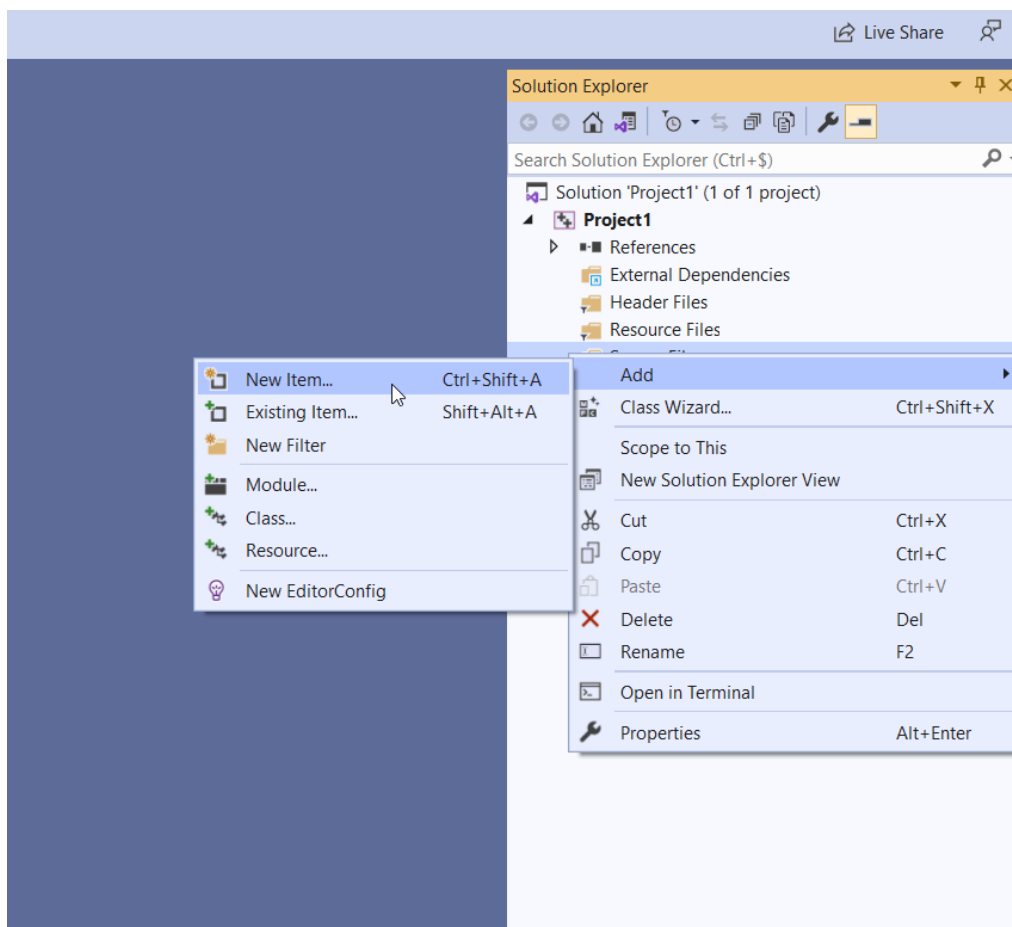


In this example the project is called Project1 (default), and the location is set to the folder C:\Temp\Develop\CDev\vs19. Click create to continue

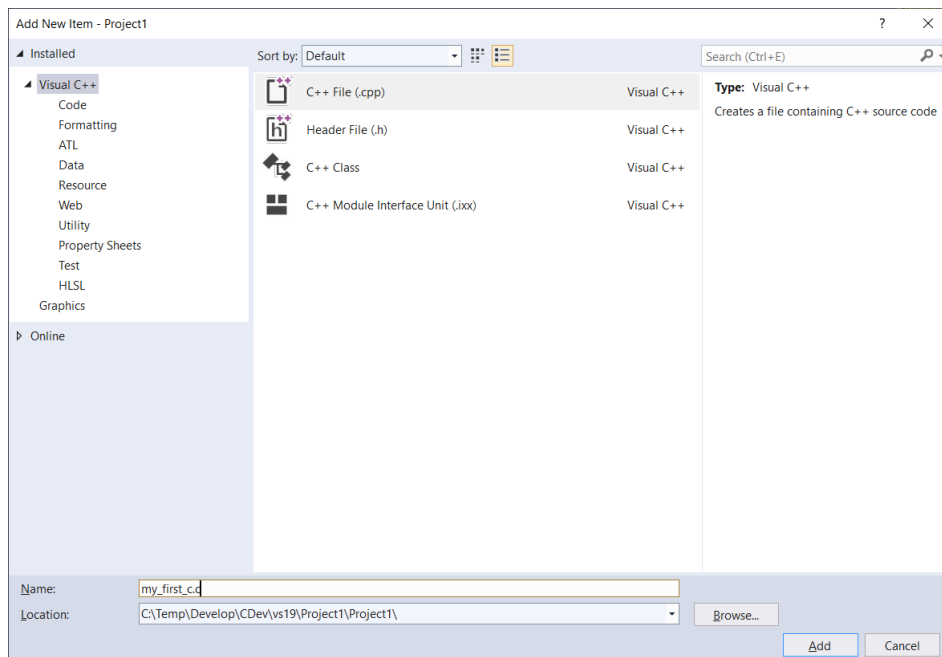


The project will then be created, the project is empty, the first thing to do is to add a source file.

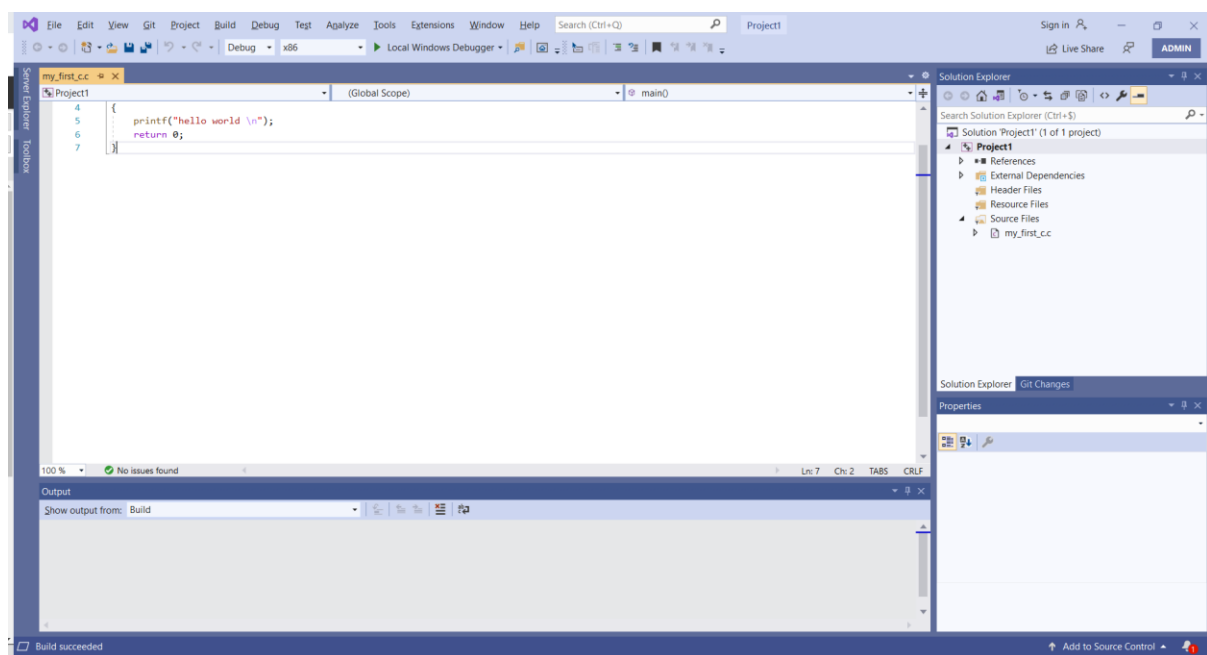
Right-click on the source files item and select to add a new item



Give the file name of the source file to be created. It has to have the extension .c (my_first_c.c)

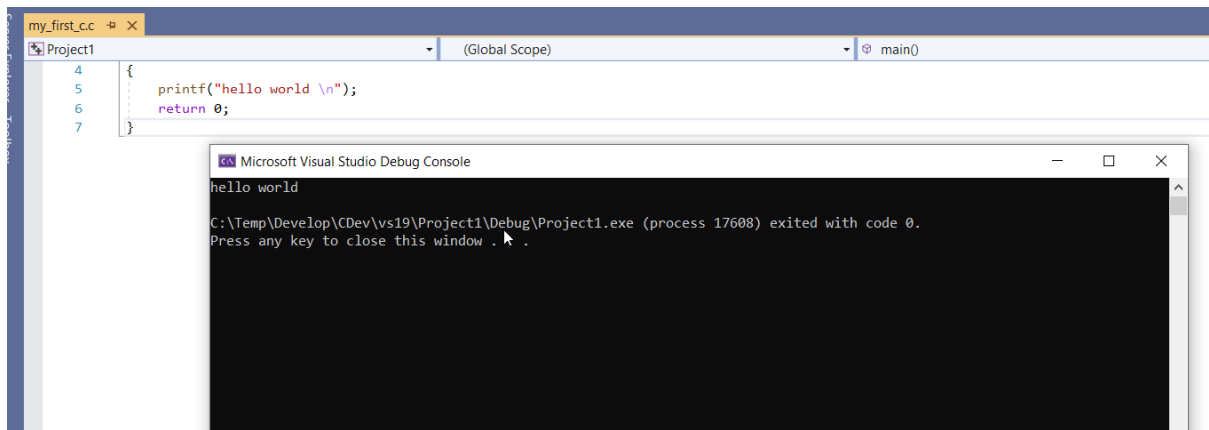


Enter the code in the newly created file.



Save the file and when the file is ready to compile , select Build solution from the Build menu.

After a successful build, the code can be executed. Select Start without debugging from the Debug menu.



It is useful to have an understanding of the files that Visual Studio creates and where these files are stored on disk.

If changes to the program are needed, return to the editor and modify the source file. To run the program again, compile it again and then run it.

WSL (Windows Subsystem for Linux)

Source: (accessed 2020/12/15)

<https://www.cs.odu.edu/~zeil/FAQs/Public/win10Bash/>

<https://nickjanetakis.com/blog/using-wsl-and-mobaxterm-to-create-a-linux-dev-environment-on-windows>

<https://walkingrandomly.com/?p=6011>

Microsoft added to Windows 10 (64 bit) the ability to run Ubuntu Linux in parallel with Windows: *Windows Subsystem for Linux (WSL)*, this is a very useful way to work with Linux-based software development tools from a Windows 10 machine.

You can install the compiler and the IDE directly in Windows, but some things just seem to me to run better in Linux.

- This provides a Linux OS running alongside Windows. Both share the same hard drive (and can access each other's files), and the clipboard supports copy-and-paste between the two quite naturally.
- It is a command line implementation. This provides a server-style installation of Linux. There is no full Linux desktop. The main entry point to Linux is a text-only bash shell for entering Linux commands.
- What it could be
 - Combined with an X server running under Windows, you can launch and run GUI programs from Linux.
 - Walk through the process of setting up a programming environment consisting of:
 - The basic Linux on Windows system.
 - An X server (to display Linux-based GUI programs on the Windows-managed display screen).
 - C compiler

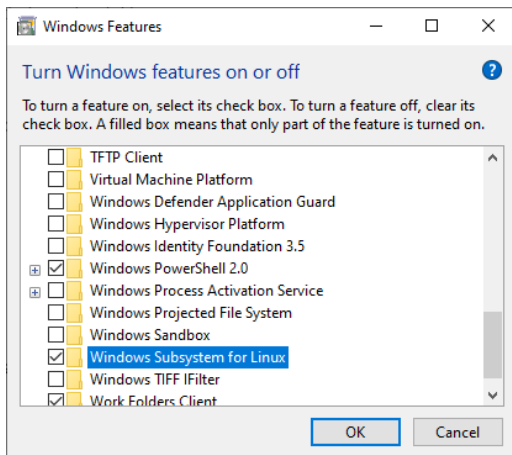
Prerequisites:

- WSL

Get WSL

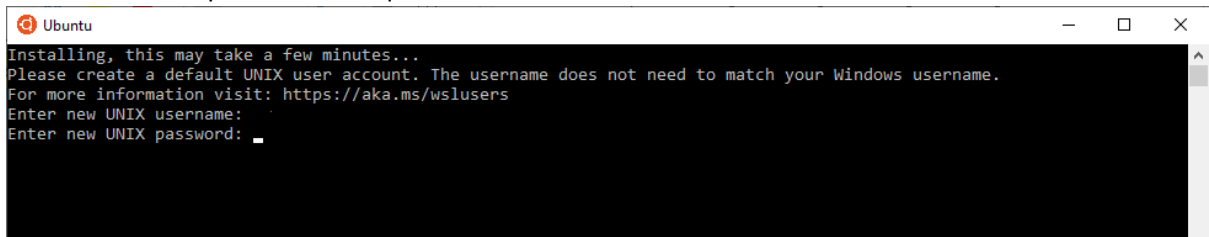
Stepping stones:

- Make sure you have Windows 10, 64-bit
- Turn on Developer Mode: Open Settings -> Update and Security -> For Developers and select the *Developer mode*. Close the Settings window.
- Enable Linux for Windows: From the taskbar, search for *Turn Windows features on or off*. A new window will pop up (Windows Features). Select *Windows Powershell 2.0* (if not already selected) and select *Windows Subsystem for Linux*. Click OK.

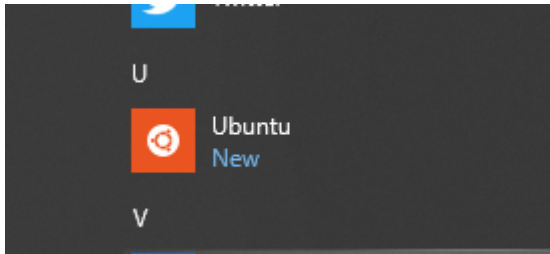


The files will be retrieved, and the system needs a reboot.

- After restart, open the Microsoft Store from the Start menu, and search for *Linux* in the store. To install a Linux distribution, click it, and then click the *Get* or *Install* button to install it like any other Store application. In the Store environment, on the current page go to the top and start the application (it can take a while). This tutorial uses Ubuntu.
- A username and password is required



- In the list of installed programs Ubuntu will appear



- Try some simple Linux commands such as `ls`, `cd`, and `pwd`. You'll find that your Windows lettered disc drives are available under `~/mnt`. For example, your C: drive is available under `/mnt`. Close your bash session for now by giving the command `exit`.

X (running graphical programs on WSL)

Source:

- <https://virtualizationreview.com/articles/2017/02/08/graphical-programs-on-windows-subsystem-on-linux.aspx>
- <http://www.alvinsim.com/my-experience-with-wsl/>

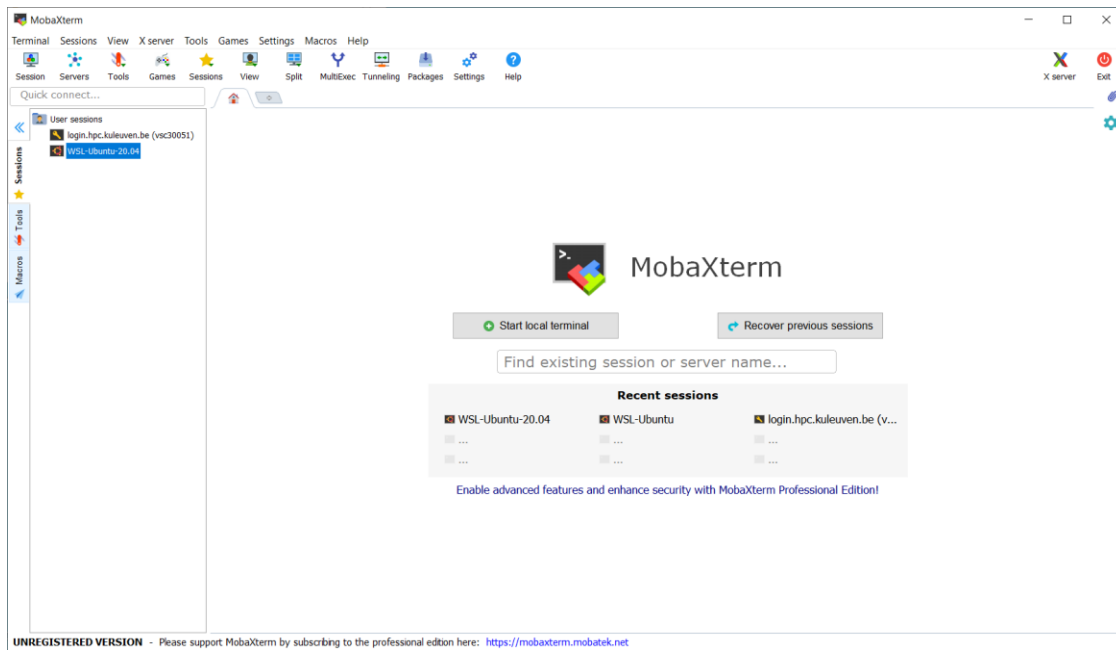
Microsoft doesn't support graphical programs on WSL. Bash on WSL is intended for running *command-line* programs that developers might need, but it's possible to run graphical Linux desktop programs on Windows using the Bash shell. To be more precise, you'll be able to display graphical

programs running in WSL on a Windows 10 desktop by using an X server which runs on the Windows side.

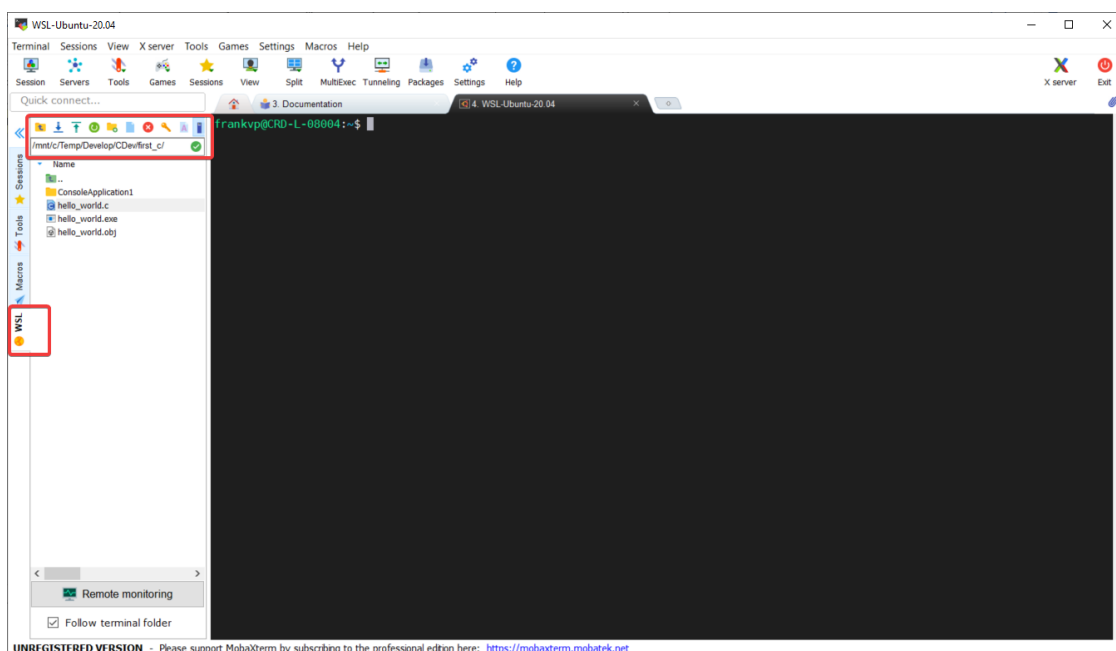
To use WSL with graphical programs, an X server will need to be installed on the Windows 10 system.

MobaXterm

An easy and quick way to get started is provided by MobaXterm (<https://mobaxterm.mobatek.net/>). A portable version can be downloaded and when you start the software, you can select for opening a WSL session, no hassle.

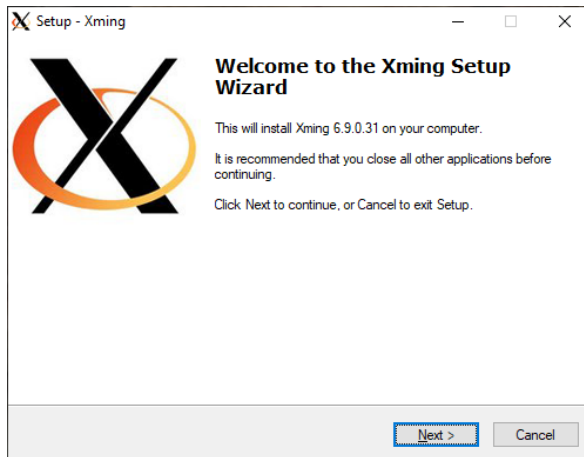


In the recent version of MobaXterm (20.6), a WSL tab is available in the desktop, enabling you to walk through the files in the WSL system, double clicking the file starts the MobaXterm editor, making life easy.

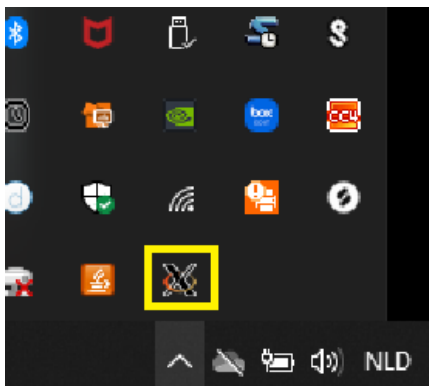


Xming

Another way to run X windows is Xming. Download the software from <https://sourceforge.net/projects/xming/> and follow the installation wizard.



The default settings are used; after launching it, Xming appears in the system tray, running in the background and waiting for a graphical WSL program.



Testing the graphical environment:

- Start Ubuntu if not running yet
- Export the display
`export DISPLAY=:0`
- Test with a graphical command: `xclock`

The `DISPLAY=` part of the earlier command is the way to tell an *X_client* program (in that case, `xclock`) where to find an X server to handle drawing things and on which of many possible screens we want to draw (from among many that server might be managing). In this case, the `DISPLAY` value is pretty simple, because we are not connecting to a remote machine and we're using the default screen.



The `DISPLAY` needs to be entered every time Ubuntu is started. So let's set that as the default for our Linux applications; add it to the `bashrc` file (at the end)

```
export DISPLAY=:0
```

Installing the compiler

Have Ubuntu running and from the command prompt, install the GNU compiler tools and the GDB debugger by typing:

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

Verify that the install succeeded by locating g++ and gdb. If the filenames are not returned from the whereis command, try running the update command again.

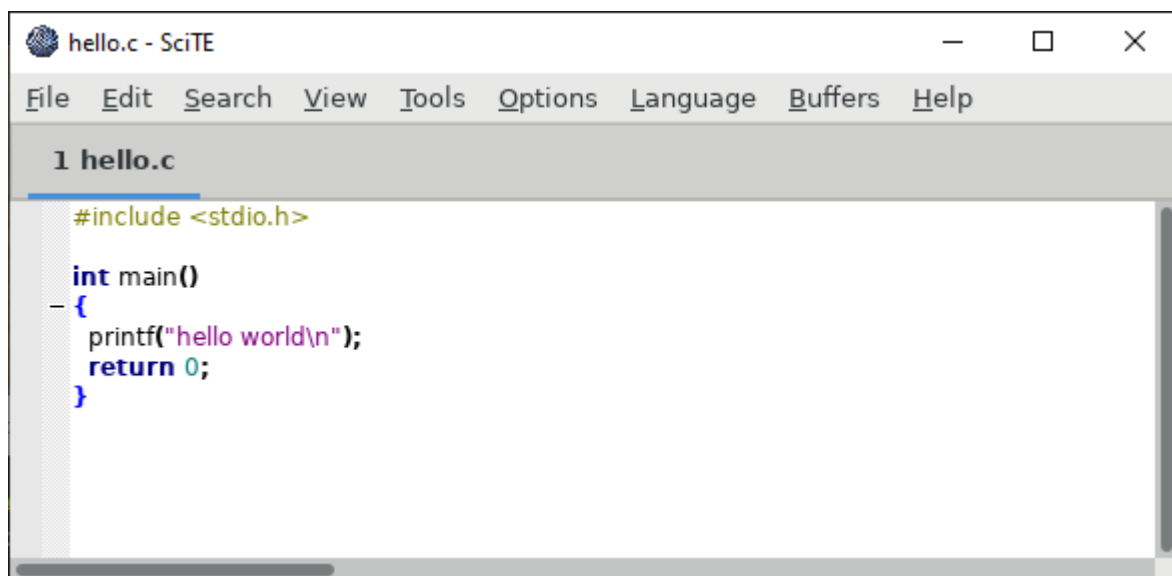
```
whereis g++
```

```
whereis gcc
```

```
whereis gdb
```

When the compiler is found, C code can be compiled.

Use an editor to edit a test program, this example uses the scite editor.



Save the file, with the extension .c (hello.c)

Compile it with `gcc hello.c`

A file a.out will be created, run this file `./a.out`

Optional packages

Editor: Your Linux installation already comes with `vim` (a Unix editor with a long history, a popular following, but a very steep learning curve) and `nano`, a basic, non-GUI editor that's great when you only need to change a few lines of text here and there. If you want something else, consider `scite`, a simple, intuitive GUI-based editor but offering syntax highlighting for many programming languages. Or if you are using MobaXterm, the built-in editor can be used.

```
sudo apt-get install scite
```

Libraries: In many cases you'll need to install some libraries that are needed by the code. Examples are LAPACK, BLAS, and the GSL libraries.

The GNU Scientific Library (GSL) is a collection of routines for numerical computing, open and free, easy to use, and well documented. Full documentation is available here:

<https://www.gnu.org/software/gsl/doc/html/index.html>

To do a full install:

```
sudo apt-get install libgsl-dev
```

Windows and WSL (Visual Studio Code approach)

Source:

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

<https://code.visualstudio.com/docs/languages/cpp>

Remark : Visual Studio Code is an editor while Visual Studio is an IDE.

This part details the configuration of Visual Studio Code to use the GCC C++ compiler (g++) and GDB debugger on Ubuntu in the Windows Subsystem for Linux (WSL). GCC stands for GNU Compiler Collection; GDB is the GNU debugger. WSL is a Linux environment within Windows that runs directly on the machine hardware, not in a virtual machine.

This setup can be considered a best-of-both-worlds setup, blurring the lines between Windows and Linux.

Prerequisites:

- Windows Subsystem for Linux (WSL) and then use the links on that same page to install your Linux distribution of choice. This tutorial uses Ubuntu.
- Visual Studio Code (VS Code).
- Remote - WSL extension.

Install WSL

See previous section

Install VS Code

Visual Studio Code is a lightweight but powerful source code editor which runs on your desktop and is available for Windows, macOS and Linux. It comes with built-in support for JavaScript, TypeScript and Node.js and has a rich ecosystem of extensions for other languages (such as C, C++, C#, Java, Python, etc.). VS Code is a cross-platform IDE that uses a tasks.json file to describe how to compile (and perform other tasks) your project.

The software can be downloaded from <https://code.visualstudio.com/download>

Follow the install wizard to get Visual Studio Code installed.

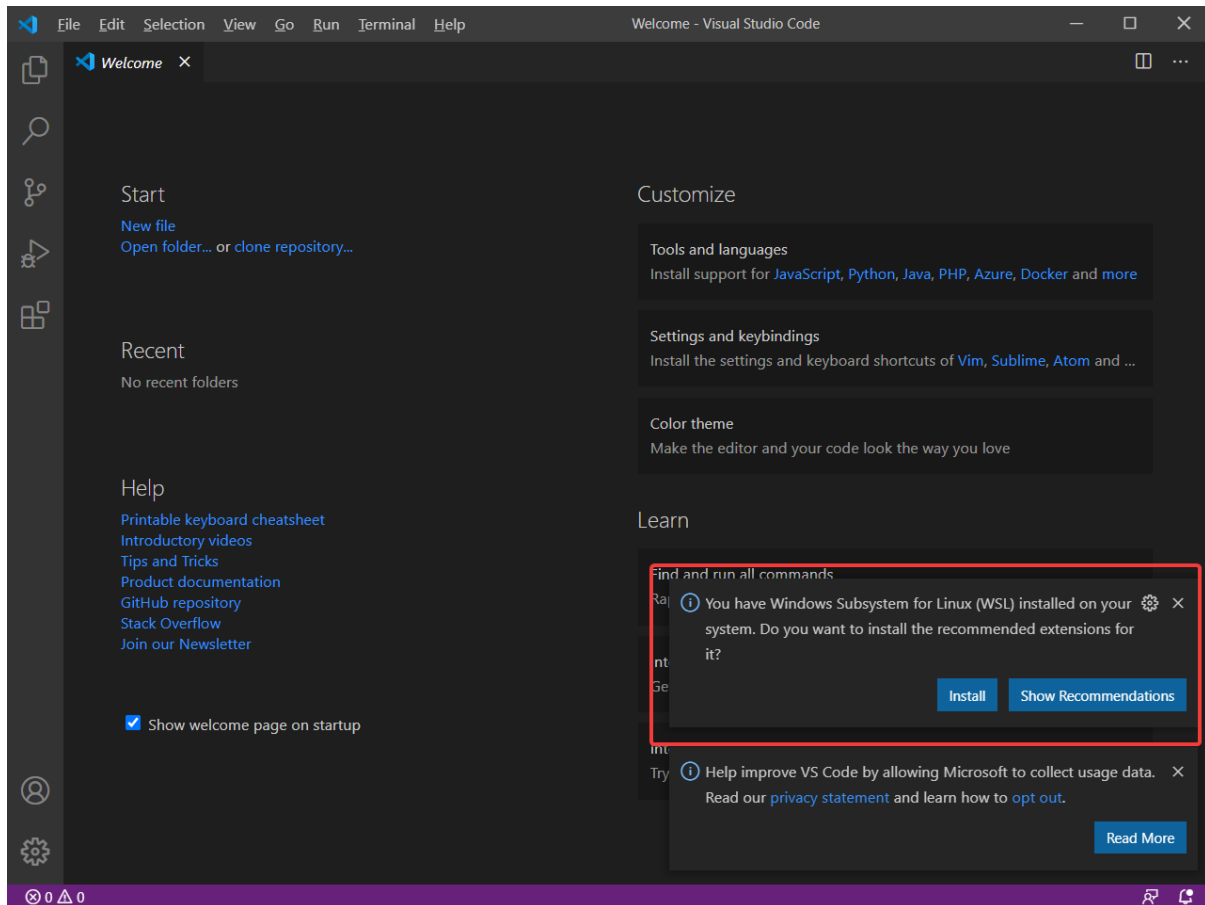
Remote – WSL extension

The Remote - WSL extension extension lets you use the Windows Subsystem for Linux (WSL) as your full-time development environment right from VS Code. This new, optimized support lets you:

- Use Windows to develop in a Linux based environment, using Linux specific toolchains and utilities.
- Edit files located in WSL or the mounted Windows filesystem (e.g. /mnt/c).
- Run and debug your Linux based applications on Windows, in VS Code.

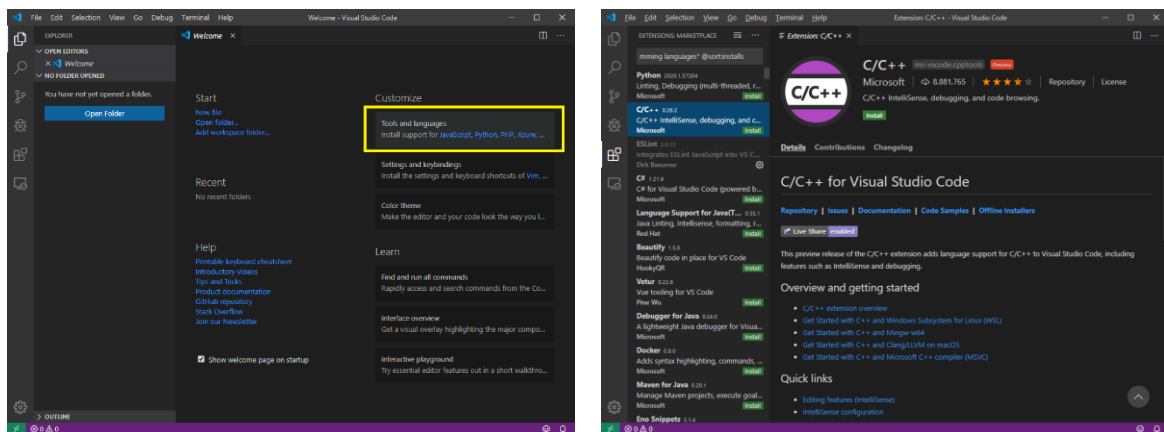
Remote - WSL runs commands and extensions directly in WSL so you don't have to worry about pathing issues, binary compatibility, or other cross-OS challenges. You're able to use VS Code in WSL just as you would from Windows.

If WSL is already installed, the VS Code proposes to install 'Remote-WSL' extension at startup. This is the magic trick to connect to the Ubuntu file system.



After installation, install a few extensions for C/C++ programming as well. After launching VS Code, click on the *Tools and Languages* box. The left part of the screen will show the support available for different languages. Select the extensions needed and click *install* to have them installed.

For C, install the C/C++ for Visual Code and C++ Intellisense extensions.



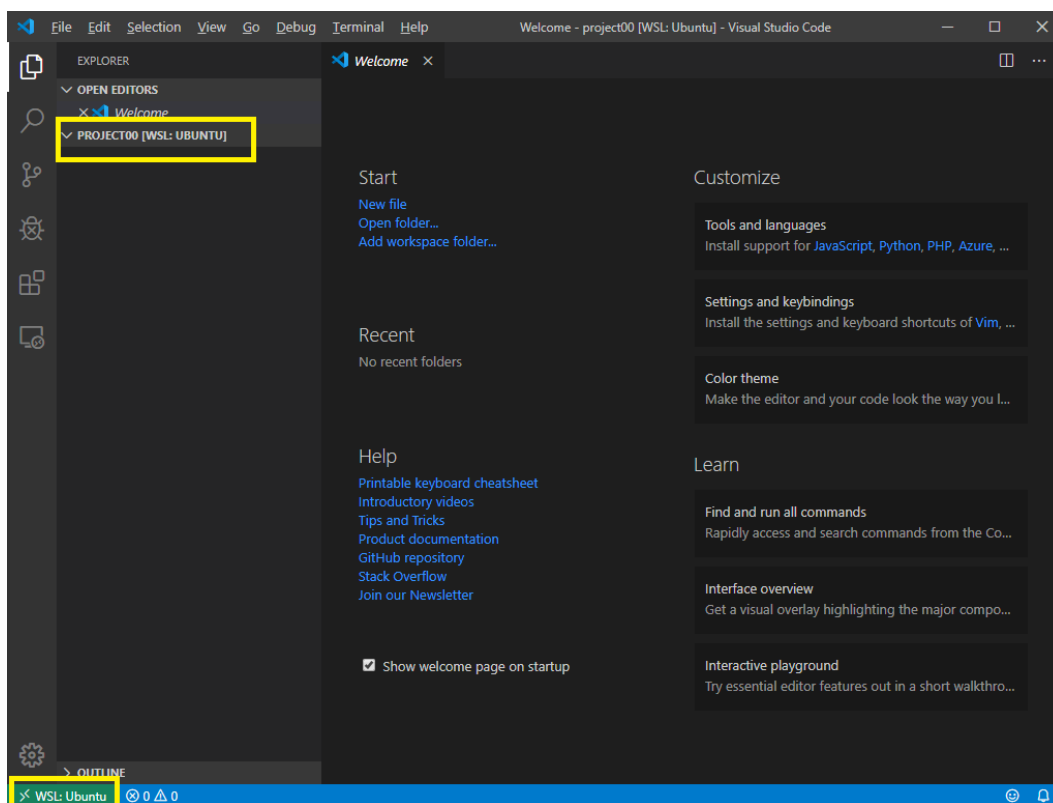
Test the installation

In WSL, create a folder (project00) and move into that folder

```
mkdir project00
```

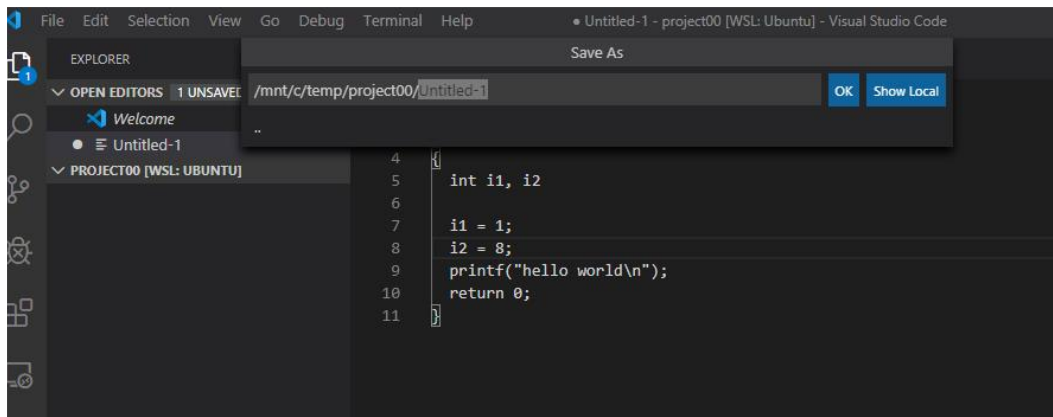
```
cd project00
```

Start VSCode in the folder: `code .` (. is important!)

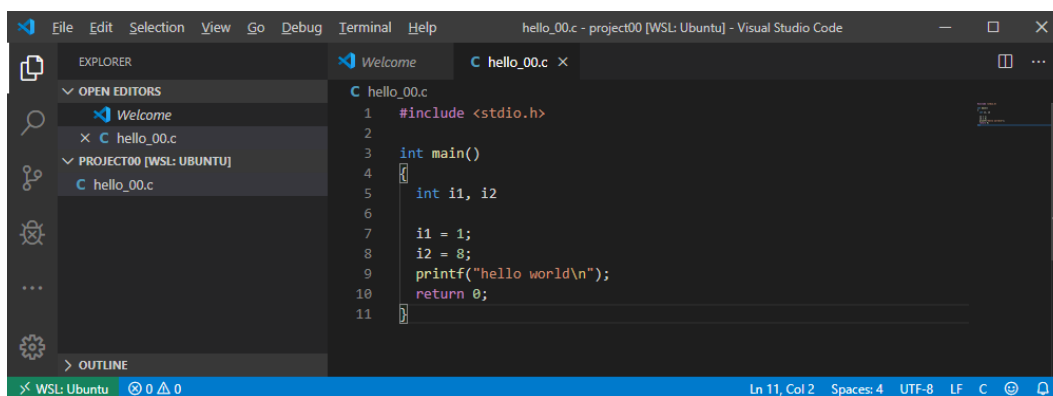


Look in the lower left corner, the WSL extension must be active, and the explorer shows the folder project00

Create a new file and enter the c code and save the file.



Save it as hello_00.c

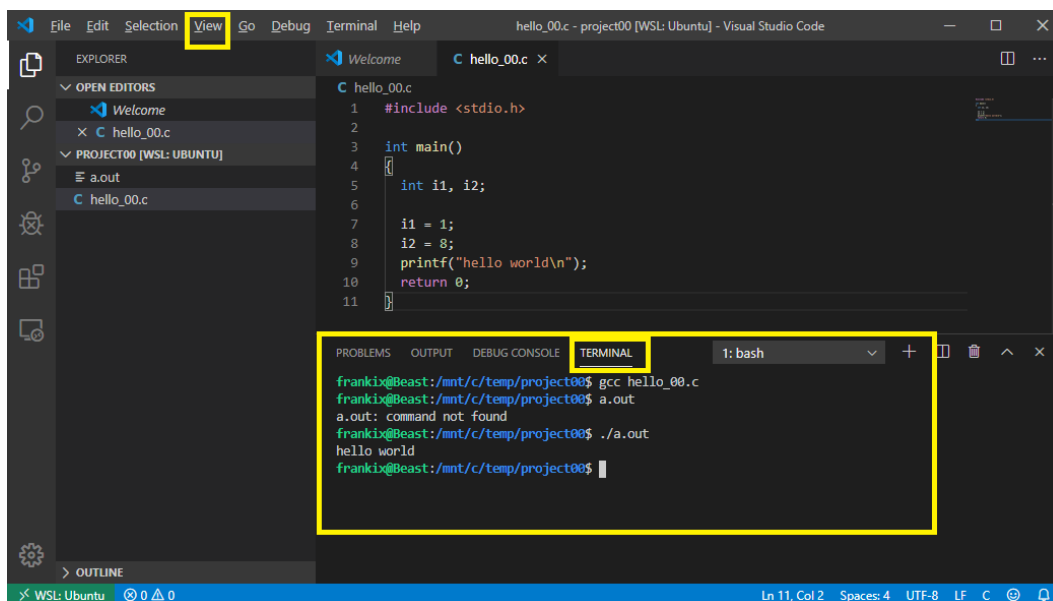


Open in the View menu a terminal window, this will open a terminal in WSL.

Compile the code with `gcc hello_00.c`

`a.out` will be created

run it with `./a.out`



Using the VS Code approach / tools (TODO – aanpassen voor WSL)

Source: <https://dev.to/mikkel250/setting-up-vscode-for-c-in-linux-po8>

Once the C/C++ extension is installed, you'll need to create a workspace to set up and run compile tasks. There are several options:

- Create a top level parent folder to hold all your C projects and then use subfolders for each project (good if you're doing learning exercises).
- Another option would be to create a test folder to get comfortable with this setup, and then create your real project folders and set them up as their own workspaces, which will result in a more organized final setup.

Once the folder is created, go to File>Add Workspace to Workspace. Then save the workspace from the File menu, or, once you close the editor, you'll be prompted to save it. Once you're comfortable with setting up the compiler, you can also create a separate workspace for each project folder instead of using the top level parent, if you prefer to organize it that way.

Once done, use Ctrl + Shift + P to open the command palette, type in "c/c++" and choose 'edit configuration UI' from the list. In the window that opens, scroll down and click on "Compiler Path" dropdown. Mine automatically detected the installed compilers and listed them in the dropdown, but if yours doesn't for some reason, you can always find the path and add it manually. Every linux system will generally ship with a C compiler, so you can search for yours via google, or in the terminal try the which command and the name of common compilers: gcc, clang, llvm. Then type the path into the "Compiler Path" field. Mine were in /usr/bin/clang and /usr/bin/gcc. Most likely yours are too.

If for some reason the user input field doesn't show up, you may have to click the "Add configuration" button in the section above. Just give it a name (like Linux default) and click OK and then the compiler path dropdown should (hopefully) appear.

Then scroll down to the "Intellisense mode" section and make sure that either `${default}` is selected, or that the correct mode is selected to match the compiler and PC architecture (e.g. if using the clang compiler, set the Intellisense mode to clangx64 if you're using a 64 bit PC). If default is selected, it should pick the right one.

And that's it! Real easy. The settings should now be saved in a `c_cpp_properties.json` file.

Now, if you want to compile files, just use Ctrl + Shift + B or choose 'Run Build Task' from the Terminal menu at the top. This should run the compiler.