A BEGINNERS GUIDE TO PROGRAMMING

Book 2

Development Environment Overview

Alex Maddern

29 January 2024

Table of Contents

Licence

Written by Alexander Maddern.

With many thanks to Daniel Moore for guidance with the Linux OS system, proof reading and corrections as well as the many ideas, examples and suggestions that have been provided.

This document is provident in the hope that it will be provide a useful overview of the concepts that are covered and that many concepts will only be covered in part or brief. The author does not except liability for the accuracy of the content provided within this document. The reader should seek to obtain documentation for the specific IDEs, compilers, interpreters as well as the programming languages and platforms used within this document. It is recommended to use a sandboxed virtual environment to test all of the examples that have been provided.

Revisions

Version	Date	Notes
Draft V 0.1	19/03/2022	Windows Dev installs
Draft V 0.2	19/04/2022	Linux base utility installs.
Draft V 0.3	20/04/2022	Add Linux C Dev. Clean up unneeded notes.
Draft V 0.4	25/04/2022	Add Linux FBC.
Draft V 0.5	26/04/2022	Add Python3 development – Thonny.
Draft V 0.6	26/04/2022	Add VBox “Shared Folders” Lubunu.
Draft V 0.7	27/04/2022	Add VBox “Shared Folders” Windows > 8.1. Addede Shared clipboard.
Draft V 0.7	28/04/2022	Corrections to formatting and layout.
Final V 0.8	01/05/2022	Final Draft – Limited Public Preview.
Final V 0.9	21/10/2022	Final Draft – Limited Public Preview. Correct Supplemental VirtualBox additions and shared folders. Correct C/C++ IDE setup in Windows and Ubuntu. Fix output PIE to No-PIE for Linux.
Final V 0.10	03/10/2022	Proof read and corrections.
Final V 0.11	13/10/2022	Correct Linux archive extraction method FB.
Final V 0.12	01/11/2022	Update Supplemental.
Final V 0.13	07/07/2023	Testing with Lubuntu 22.04 LTS
Final V 0.14	08/07/2023	Testing with FreeBASIC 1.10.0
Final V 0.15	29/01/2024	Minor corrections.

TODO:

[Fix sf_shared user account permissions issues sudo usermod -G vboxsf -a user2 (whoami?)]

[Add brief note about static vs shared project compile.]

[Add information on debuggers, executable analysis, and code analysis] Partial

[Add additional information on removing and importing files into an IDE project. See: Example 2 curses project.]

About development environments

Development environments are the collection of tools required in programming and coding. This can range from compilers and interpreters through code editors and IDEs as well as additional tools and libraries that are required to complete a software project.

The most fundamental component is focused around the programming language and will come in the form of either a compiler and machine language assembler or a language interpreter that transforms code to machine language at run time.

Most compilers and interpreters will also be distributed with a standard code library as part of the package. This will include library modules containing pre-written code containing solutions to the most common tasks and solutions to a programming problem. Many specialised libraries are also available to implement specific solutions.

Most coding environments can be used via a text editor and the command line environment to write and test coding solutions, although this can become intensive, tedious and inconvenient especially when working with a large project. To make the development task faster and more convenient we can employ the use of an “Integrated Development Environment” (IDE).

Many developers will use special scripting files from the command line for building the final executables of the project. The 2 most common are MakeFile and CMake. These are in some way quite similar to a shell script or batch file except they are orientated toward working with compiler chains and building projects from source. Many IDEs including the C/C++ IDEs I use in these guides will automatically generate a basic “MakeFile” based upon the configuration and switches set in the IDE. If you take the time to read through the Makefile or CMake build scripts you will begin to see the command line calls sent to the compiler. In large projects the MakeFiles can become quite complex.

An IDE is a collection of tools that make a programming development task easier to manage. At the most basic level an IDE will consist of a text editor with code syntax highlighting ability as well as a built in method to debug, test and compile our code. IDEs can range from the most simple as described above to extremely complex tool suites. There is no “best” or “worst” when it comes to an IDE and the choice of IDE will generally be made with regards to what skill levels you have, what is required to complete the programming solution with a minimum of effort and in many cases a matter of personal preference.

The development environments and IDEs that I have chosen to illustrate within this document don’t constitute the “best’ or “most recommended’. I have chosen what I believe is the most convenient method to install all of the development environments and IDEs with some predefined configurations with the aim of allowing a beginner to quickly jump into the world of coding with the minimum of frustration.

The development environments used in this guide

In my previous booklet *“A BEGINNERS GUIDE TO PROGRAMING - Book 1 - Programming and Coding Overview”* I offered an introduction to the fundamentals of programming and coding with illustrations using 3 programming languages; the C compiler language, the FreeBASIC compiler language and the Python 3 interpreter language. I also offered examples that will run in both the

Windows and the Ubuntu operating systems, so the following guide will outline the setup tasks for all 3 languages in both Windows and Ubuntu that I used in that guide.

I have offered a quick setup guide for each OS for those that are already confident in installing applications. You can then skip down to the relevant configs for each IDE if you wish.

Although there are many different methods available to install a development stack I am going to show the simplest method and the setup least likely to encounter conflicts. On Windows based machines there are often different installers with different install paths that will also result in different OS environment variables. Installing from 2 alternative sources for a development stack can easily lead to version and install path conflicts. This is not to say we cannot install different versions of a development stack, it just means that the level of difficulty and the risks of a dysfunctional programming environment increase quite rapidly.

For C/C++ development many will recommend Linux subsystem environments such as Cygwin and Msys2. Although it may appear convenient to have the Linux system available on a windows machine I do not recommend this as you will inevitably write code that is part Windows, part Linux. This hybrid code will need to be shipped with the Linux subsystem libraries to run on any other Windows machine. I feel it is better, especially as a beginner, to write code and build executable files based on the built in libraries that ship natively with all Windows ensuring your applications will compile and run on any Windows machine.

As a rule I install all development stacks for Windows systems into the root of OS system drive, usually **C:**.

I give each programming environment a specific and unique directory name so as to avoid any possible naming conflicts in the future. So be pedantic about using version numbers and CPU architecture in folder names. For example do not use generic folder names such as Python for the python stack, use Python3x64 for example. Avoid spaces in directory names, so don't use **C:\Python 3 x86; Use C:\Python_3_x86*.*** instead.

You can use the generic zip folder names if you wish; I just find it convenient to use a uniform naming convention.

It's best to avoid changing directory names after they are created as you may need to alter the system environment paths as well as the default paths of some applications using the stack, so take a moment to think about naming conventions for your development paths. You can install version updates to your development tools in the same directory once you are familiar with the language and how the development tools work, or alternatively you can install updates to a new directory. For example you may already have **C:\Dev_Python\Python3x64*.*** so you could update that folder, or create a new directory **C:\Dev_Python\Python39x64*.*** and point your IDE and other tools to that path.

As a beginner I would recommend just using the basic install methods below. Once you are confident you can delete the development environment and try alternative methods.

Windows 64 bit Development environments

IDE and environment overview

You do not have to install all of the IDEs and only choose what you require. If you would like to attempt all the code examples from "A Beginners Guide To Programming" then you will need to install all IDEs.

If you are planning on using a guest OS in a VirtualBox environment please be sure to read the section under "Supplemental".

C/C++

Embarcadero Dev-C++

Free

https://www.embarcadero.com/free-tools/dev-cpp

Download:

https://www.embarcadero.com/free-tools/dev-cpp/free-download

You will need to register with an email address and basic details to download.

Alternative Download

https://github.com/Embarcadero/Dev-Cpp/releases

"Embarcadero_Dev-Cpp_6.3_TDM-GCC_9.2_Portable.7z" or the most recent version.

Embarcadero Dev-C++ is a new and improved fork (sponsored by Embarcadero) of Bloodshed Dev-C++ and Orwell Dev-C++. It is a full-featured Integrated Development Environment (IDE) and code editor for the C/C++ programming language. It uses the MinGw port of GCC (GNU Compiler Collection) as its compiler. Embarcadero Dev-C++ can also be used in combination with Cygwin or any other GCC based compiler. Embarcadero Dev-C++ is built using the latest version of Embarcadero Delphi. Embarcadero Dev-C++ has a low memory footprint because it is a native Windows application and does not use Electron.

Main Features Include:

- TDM-GCC 9.2.0 32/64bit

  https://jmeubank.github.io/tdm-gcc/

  There is a new update for TDM-GCC 10.3.0. This is the GCC C/C++ MinGW libraries.

  I don't recommend attempting to upgrade until you have spent some time familiarising yourself with the development environment. I will offer an outline of how to do this at the end of the chapter.

- Support GCC-based compilers
- Integrated debugging (using GDB)
- GPROF profiling

- Project Manager
- Customizable syntax highlighting editor
- Class Browser
- Code Completion
- Code Insight
- Function listing
- AStyle code formatting support
- GPROF Profiling support
- Quickly create Windows, console, static libraries and DLLs
- Support of templates for creating your own project types
- Makefile creation
- Edit and compile Resource files
- Tool Manager
- Devpak IDE extensions
- Print support
- Find and replace facilities
- CVS support

Supported Operating System:

- Windows 7
- Windows 8.1
- Windows 10

FreeBASIC

FreeBASIC Compiler

Free

https://www.freebasic.net/

Downloads

https://sourceforge.net/projects/fbc/files/

I recommend checking for the most recent version "/FreeBASIC-1.xx.x/"

https://sourceforge.net/projects/fbc/files/FreeBASIC-1.09.0/Binaries-Windows/

Choose the file download the reflects the most recent version “FreeBASIC-1.xx.x-winxx.xxx”

FreeBASIC-1.09.0-win64.zip

FreeBASIC-1.09.0-win64.7z (Recommended)

FreeBASIC-1.09.0-win32.zip

FreeBASIC-1.09.0-win32.7z

FreeBASIC is a free/open source (GPL), BASIC compiler for Microsoft Windows, DOS and Linux.

FreeBASIC is a self-hosting compiler which makes use of the GNU binutils programming tools as backends and can produce console, graphical/GUI executables, dynamic and static libraries. FreeBASIC fully supports the use of C libraries and has partial C++ library support. This lets programmers use and create libraries for C and many other languages. It supports a C style preprocessor, capable of multiline macros, conditional compiling and file inclusion.

The FreeBASIC project is a set of cross-platform development tools, consisting of a compiler, GNU-based assembler, linker and archiver, and supporting runtime libraries, including a software-based graphics library. The compiler, fbc, currently supports building for i386-based architectures on the DOS, Linux, Windows and Xbox platforms. The project also contains thin bindings (header files) to some popular 3rd party libraries such as the C runtime library, Allegro, SDL, OpenGL, GTK+, the Windows API and many others, as well as example programs for many of these libraries.

FreeBASIC is a high-level programming language supporting procedural, object-orientated and meta-programming paradigms, with a syntax compatible with Microsoft QuickBASIC. In fact, the FreeBASIC project originally began as an attempt to create a code-compatible, free alternative to Microsoft QuickBASIC, but it has since grown into a powerful development tool. FreeBASIC can be seen to extend the capabilities of Microsoft QuickBASIC in a number of ways, supporting more data types, language constructs, programming styles, and modern platforms and APIs.

FreeBASIC has been rated close in speed with mainstream tools, such as GCC.

Note that there are a number of different install sets available for FreeBASIC and exist under a variety of names such as “FreeBASIC-1.09.0-winlibs-gcc-9.3.0”. I have only included the 64-bit version to keep it simple for the books and exercises.

FreeBASIC IDE

FBIde is an open source IDE for FreeBASIC compiler. It allows you to edit the source code and then compile it right from within the FBIde without any need to use the command line.

Features

- Compile & run with just 1 click

- Auto Format your code
- Automatic code indentation
- Configurable syntax highlighting
- Has translations in 18 languages
- Compile error reporting
- Code browser with search
- Source code export into HTML
- Integrated file browser
- Context sensitive FreeBASIC help
- Block comment & uncomment

Download

http://fbide.freebasic.net/download

Download the IDE Only without the FB compiler FBIde .zip

“FBIde0.4.6r4.zip”

Geany Editor

Free

https://geany.org/

Download

https://geany.org/download/releases/

Windows (64-bit1) “geany-1.38_setup.exe”

Geany is a powerful, stable and lightweight programmer's text editor that provides tons of useful features without bogging down your workflow. It runs on Linux, Windows and macOS, is translated into over 40 languages, and has built-in support for more than 50 programming languages.

Geany is the default Editor IDE used with FreeBASIC on linux.

Python 3

Python 3

Free

https://www.python.org/

Download

https://www.python.org/downloads/

https://www.python.org/downloads/release/python-3103/

“python-3.10.3-amd64.exe”

Python is a high-level, general-purpose programming language. Its design philosophy emphasizes code readability with the use of significant indentation. Its language constructs and object-oriented approach aim to help programmers write clear, logical code for small- and large-scale projects.

Python is dynamically-typed and garbage-collected. It supports multiple programming paradigms, including structured (particularly procedural), object-oriented and functional programming. It is often described as a "batteries included" language due to its comprehensive standard library.

Guido van Rossum began working on Python in the late 1980s as a successor to the ABC programming language and first released it in 1991 as Python 0.9.0. Python 2.0 was released in 2000 and introduced new features such as list comprehensions, cycle-detecting garbage collection, reference counting, and Unicode support. Python 3.0, released in 2008, was a major revision that is not completely backward-compatible with earlier versions. Python 2 was discontinued with version 2.7.18 in 2020. [en.wikipedia.org]

Thonny

Free

https://thonny.org/

Download

“thonny-3.3.13.exe” (Python 3.7 built in)

Thonny is an integrated development environment for Python that is designed for beginners. It supports different ways of stepping through the code, step-by-step expression evaluation, detailed visualization of the call stack and a mode for explaining the concepts of references and heap. [en.wikipedia.org]

Thonny comes with Python 3.7 built in, so just one simple installer is needed and you're ready to learn programming. (You can also use a separate Python installation, if necessary.) The initial user interface is stripped of all features that may distract beginners.

PyScripter

Free

https://sourceforge.net/projects/pyscripter/

Download

https://sourceforge.net/projects/pyscripter/files/PyScripter-v4.2/

"PyScripter-4.2.5-x64-Setup.exe"

PyScripter is a free and open-source Python Integrated Development Environment (IDE) created with the ambition to become competitive in functionality with commercial Windows-based IDEs available for other languages.

Features

https://sourceforge.net/p/pyscripter/wiki/Sidebar/

Additional useful coding apps.

on-screen ruler

https://sites.google.com/site/rulerhelp/welcome/

Alternative PicPick has a pixel ruler.

EPIM

NP++

AutoVer

7-Zip

SumatraPDF

AutoIt Window Info Tool

https://www.autoitscript.com/autoit3/docs/intro/au3spy.htm

Windows installation

You do not have to install all of the IDEs and only choose what you require. If you would like to attempt all the code examples from "A Beginners Guide To Programming" then you will need to install all of the IDEs.

If you are planning on using a guest OS in a VirtualBox environment then please be sure to read the section under "Supplemental".

Requirements (or most recent versions)

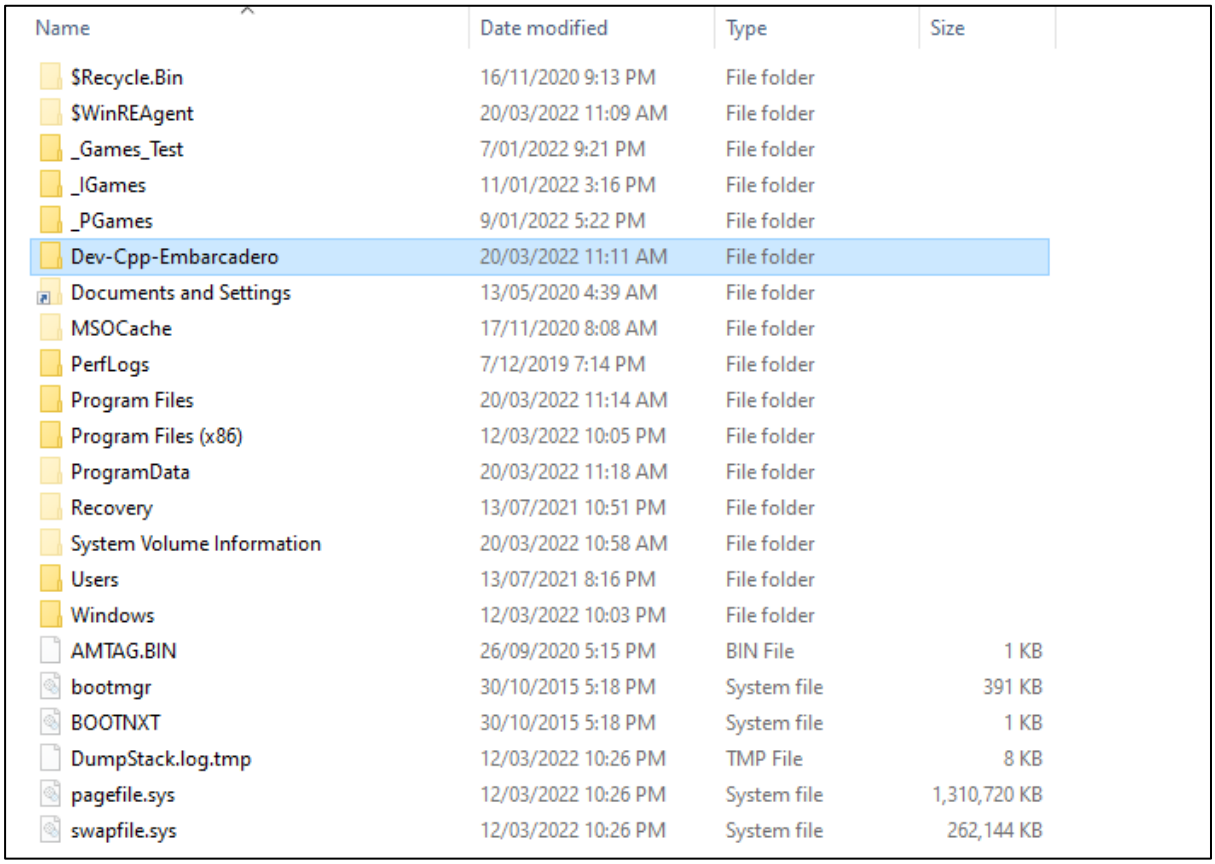
- **Embarcadero_Dev-Cpp_6.3_TDM-GCC_9.2_Portable.7z**
- **FreeBASIC-1.09.0-win64.7z**
- **FBIde0.4.6r4.zip**
- **geany-1.38_setup.exe**
- **python-3.10.3-amd64.exe** This is not mandatory if using Thonny as Thonny has the Python 3 development bundle built into the application install.
- **thonny-3.3.13.exe**

- **PyScripter-4.2.5-x64-Setup.exe** (optional alternative to Thonny)

Embarcadero Dev-C++

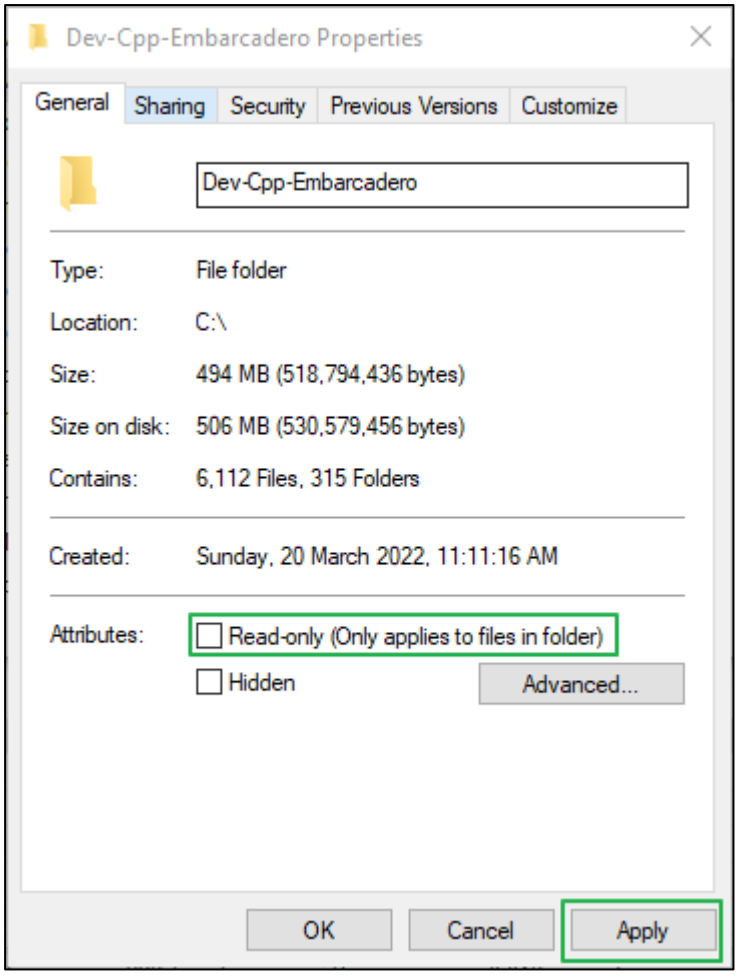
Unzip the "`Embarcadero_Dev-Cpp_6.3_TDM-GCC_9.2_Portable.7z`" archive to its top folder structure in your OS drive. "`C:\Embarcadero_Dev-Cpp_6.3_TDM-GCC_9.2_Portable\*.*`".

Rename "`C:\Embarcadero_Dev-Cpp_6.3_TDM-GCC_9.2_Portable\*.*`" to "`C:\ Dev-Cpp-Embarcadero\*.*`".



Name	Date modified	Type	Size
$Recycle.Bin	16/11/2020 9:13 PM	File folder	
$WinREAgent	20/03/2022 11:09 AM	File folder	
_Games_Test	7/01/2022 9:21 PM	File folder	
_IGames	11/01/2022 3:16 PM	File folder	
_PGames	9/01/2022 5:22 PM	File folder	
Dev-Cpp-Embarcadero	20/03/2022 11:11 AM	File folder	
Documents and Settings	13/05/2020 4:39 AM	File folder	
MSOCache	17/11/2020 8:08 AM	File folder	
PerfLogs	7/12/2019 7:14 PM	File folder	
Program Files	20/03/2022 11:14 AM	File folder	
Program Files (x86)	12/03/2022 10:05 PM	File folder	
ProgramData	20/03/2022 11:18 AM	File folder	
Recovery	13/07/2021 10:51 PM	File folder	
System Volume Information	20/03/2022 10:58 AM	File folder	
Users	13/07/2021 8:16 PM	File folder	
Windows	12/03/2022 10:03 PM	File folder	
AMTAG.BIN	26/09/2020 5:15 PM	BIN File	1 KB
bootmgr	30/10/2015 5:18 PM	System file	391 KB
BOOTNXT	30/10/2015 5:18 PM	System file	1 KB
DumpStack.log.tmp	12/03/2022 10:26 PM	TMP File	8 KB
pagefile.sys	12/03/2022 10:26 PM	System file	1,310,720 KB
swapfile.sys	12/03/2022 10:26 PM	System file	262,144 KB

Right click on the `\Dev-Cpp-Embarcadero` folder and unselect the Read Only Permissions and select apply to all folder and files. Windows will reset the permission for the entire folder in line with your current user account.

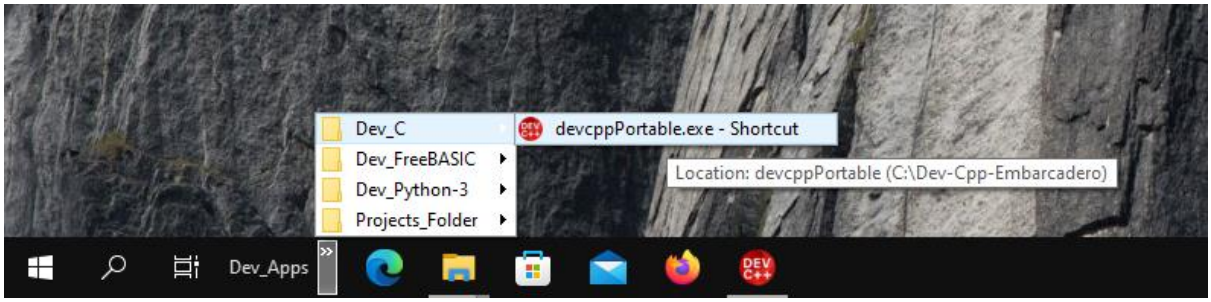


You now have the Dev-C++ IDE and C development tools "\TDM-GCC-64\" ready to go.

Name	Date modified	Type	Size
AStyle	31/01/2021 11:19 AM	File folder	
Fonts	31/01/2021 11:19 AM	File folder	
Help	31/01/2021 11:19 AM	File folder	
Icons	31/01/2021 11:19 AM	File folder	
Lang	31/01/2021 11:19 AM	File folder	
TDM-GCC-64	31/01/2021 11:19 AM	File folder	
Templates	31/01/2021 11:19 AM	File folder	
ConsolePauser.exe	12/10/2020 3:35 PM	Application	179 KB
COPYING.txt	17/05/2020 2:38 PM	Text Document	19 KB
devcpp.exe	31/01/2021 11:00 AM	Application	9,529 KB
devcpp.exe.manifest	17/05/2020 2:38 PM	MANIFEST File	1 KB
devcpp.ico	10/10/2020 7:23 AM	Icon	308 KB
devcpp.map	31/01/2021 11:00 AM	MAP File	6,523 KB
devcppPortable.exe	12/10/2020 3:21 PM	Application	478 KB
NEWS.txt	31/01/2021 10:48 AM	Text Document	63 KB
Packman.exe	16/10/2020 11:46 PM	Application	6,705 KB
Packman.map	16/10/2020 11:46 PM	MAP File	4,493 KB

Select the ”`devcppPortable.exe`” and create a shortcut to the desktop or to a quick launch menu.

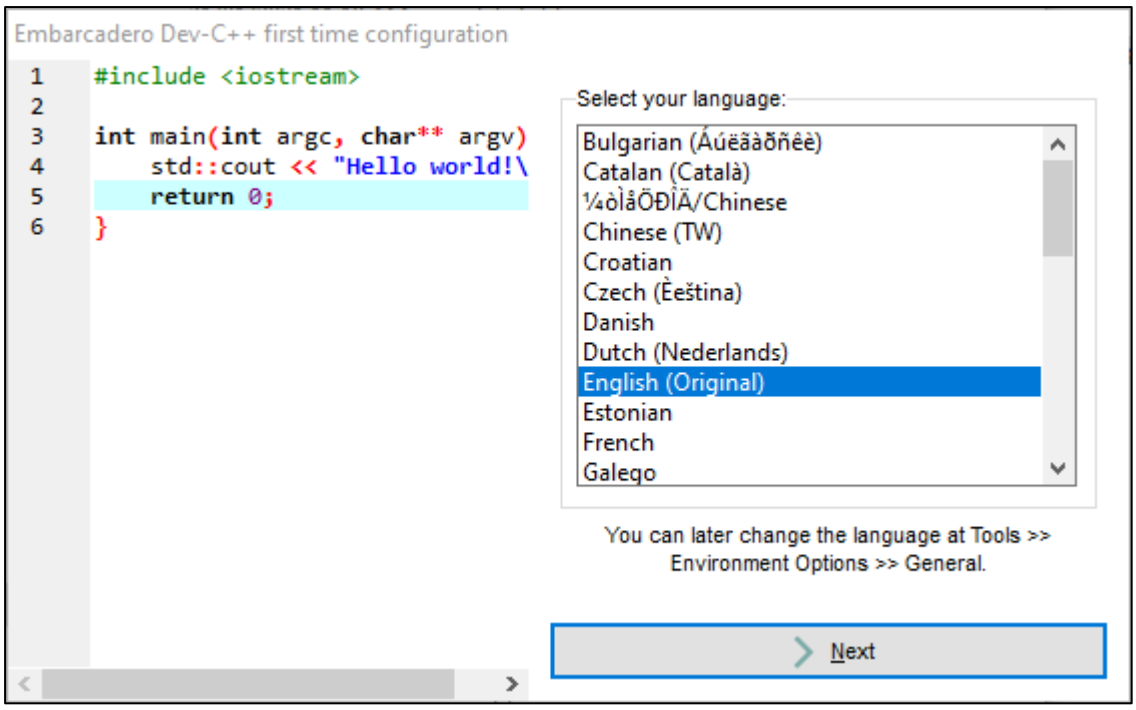
I usually use a custom quick launch menu next to the start icon for ease of organizing.

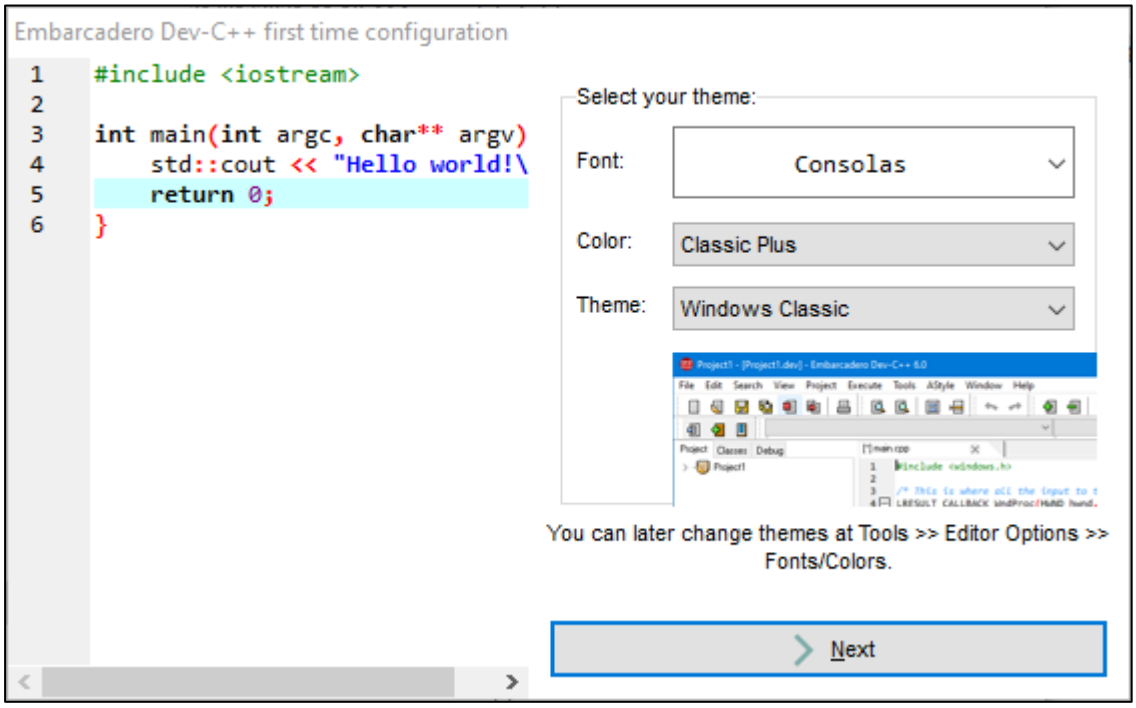


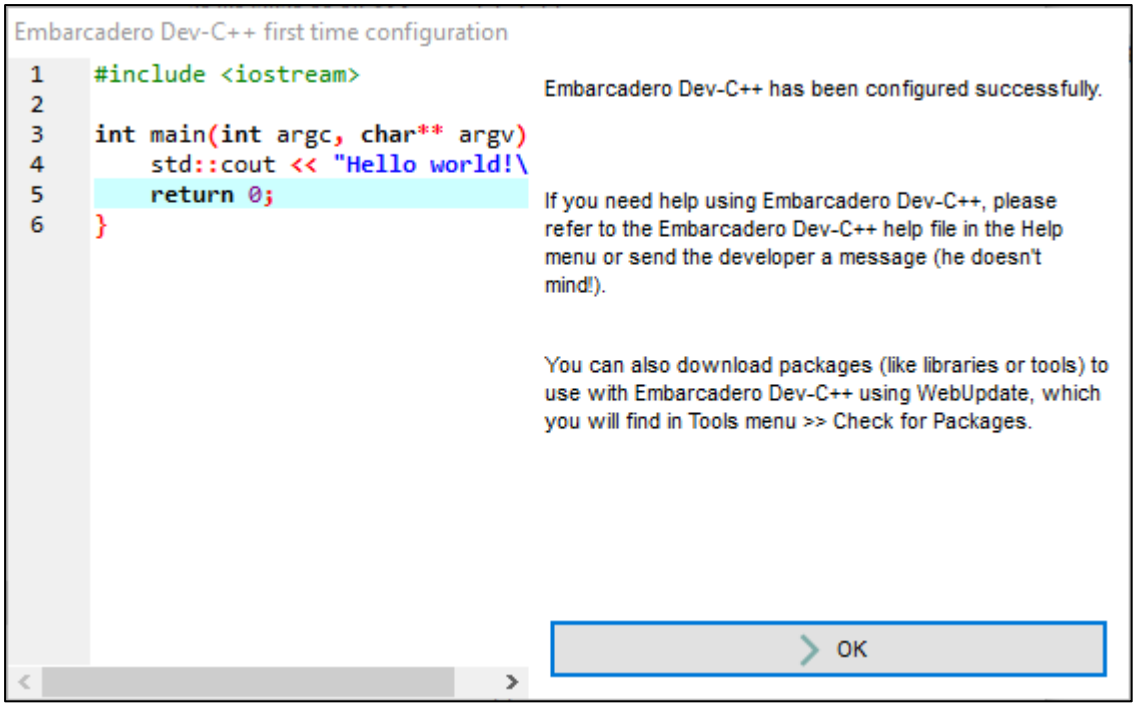
Only use the “`devcppPortable.exe`” and **not** the devcpp.exe.
devcppPortable.exe writes its config files to a `\config\` directory in the application directory.
devcpp.exe is for the installer version and writes the config files to the windows registry.

Open devcppPortable.exe and set up the first run configuration.

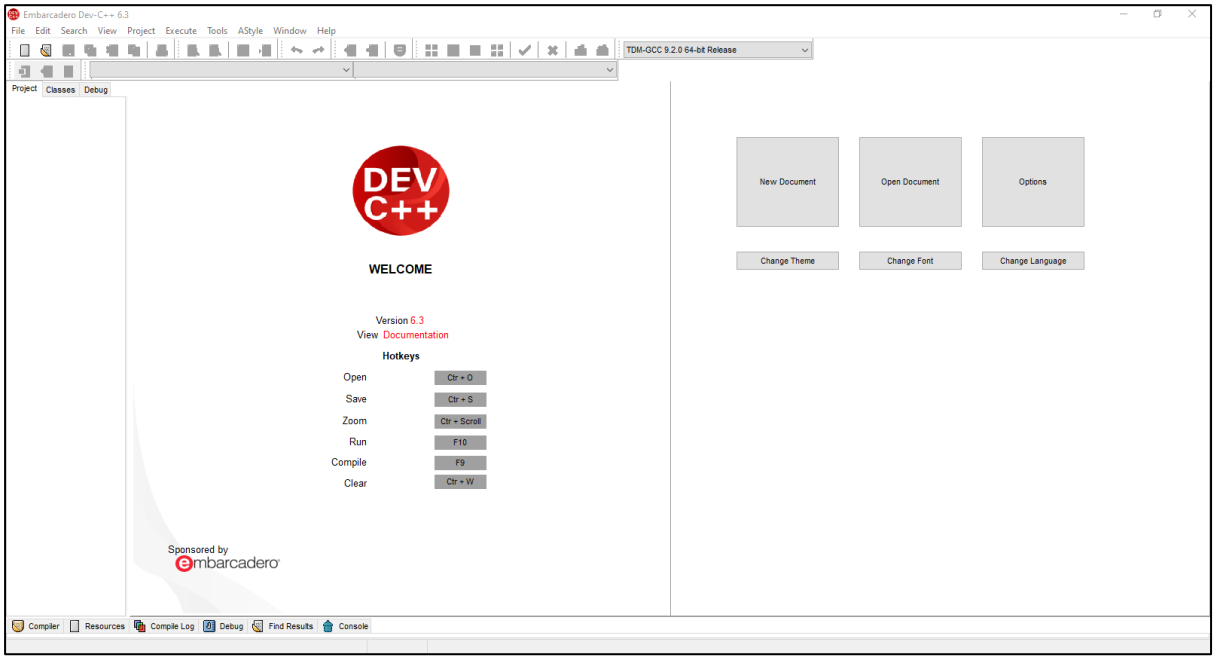
Use the default settings as you can alter them later when you become more familiar and experienced with the IDE.


Embarcadero Dev-C++ first time configuration
#include <iostream>
int main(int argc, char** argv)
std::cout << "Hello world!\
return 0;
}
Select your language:
Bulgarian (Áúëãàðñêè)
Catalan (Català)
¼òÌåÖÐÎÃ/Chinese
Chinese (TW)
Croatian
Czech (Èeština)
Danish
Dutch (Nederlands)
English (Original)
Estonian
French
Galego
You can later change the language at Tools >> Environment Options >> General.
Next


Embarcadero Dev-C++ first time configuration
#include <iostream>
int main(int argc, char** argv)
std::cout << "Hello world!\
return 0;
}
Select your theme:
Font:
Consolas
Color:
Classic Plus
Theme:
Windows Classic
You can later change themes at Tools >> Editor Options >> Fonts/Colors.
Next

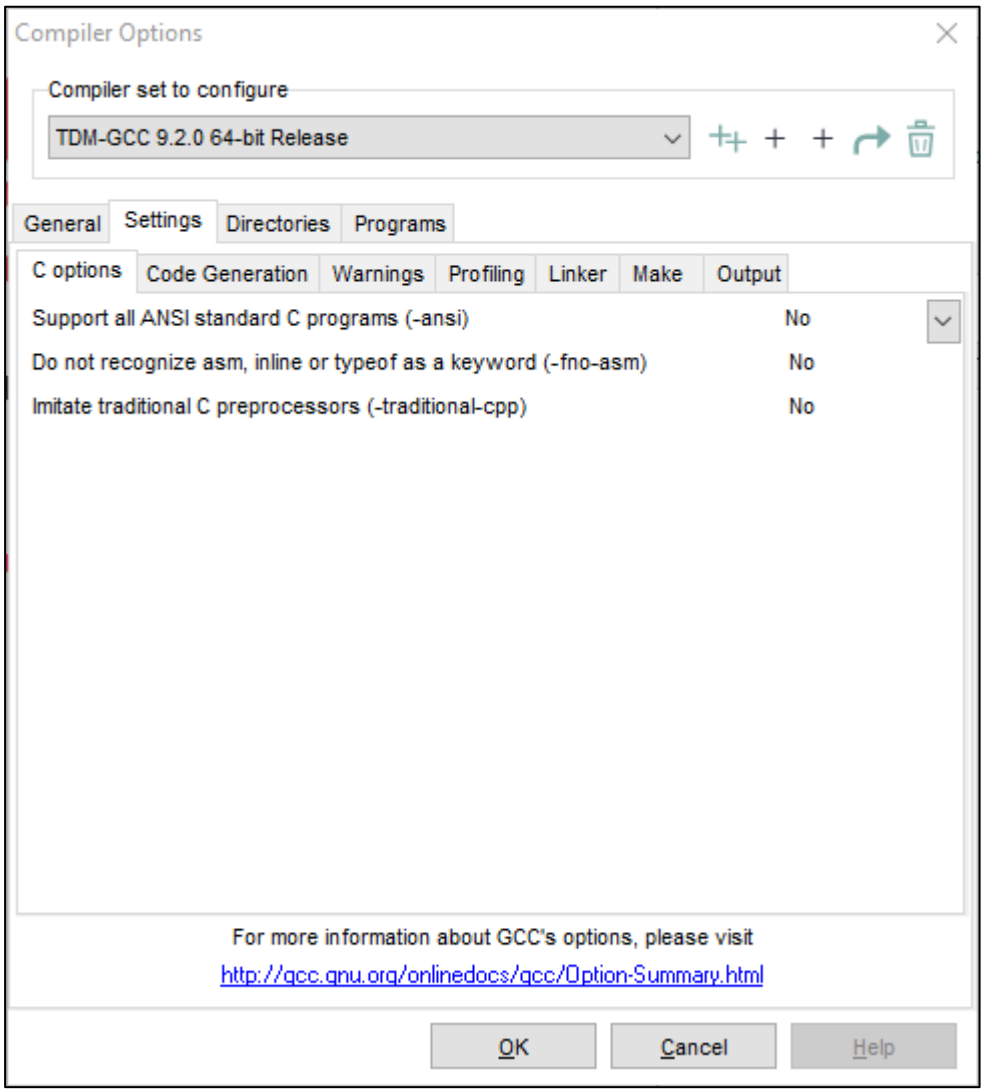


You will now have a default work area open in the IDE. The IDE will open to the default work area each time it is started.



Next, Become familiar with the Tools drop down on the menu bar. This is where we set the default custom settings for all development projects.
If you ever mess up the settings and can't get back to the defaults delete the `\config\` directory and the 2 config files "codeinsertion.ini" and "devcpp.ini" and relaunch the devcppPortable.exe

From the menu bar select “Tools” -> “Compiler Options…”. You will now see a window like the following.



Before I go any further it is important to note the differences between Dev-C++ and Code::Blocks. In Dev-C++ each “New” project will inherit a copy of this “Global” set of compiler options. This means there are 2 sets of compiler options, 1 global and 1 local to the project. Setting the global options makes it a little easier to just automatically “Inherit” the options into a new project, although it is still good practice to check though all of the settings to ensure they are correct for each project. Having the global settings in place also allows you to run individual source files outside of the project in a “Scratch Pad”. Any source that is not imported to the project will fall back to the global compiler settings.

In Dev-C++ the project only uses the compiler setting in the project and there is no risk of conflict with the global settings.

Dev-C++ uses a different system in this regard compared to Code::Blocks. As well as not allowing for code to be compiled outside of a projects, Code::Blocks also appends the Local project compiler setting to the end of the global compiler setting creating a set of duplicate flags set to the compiler.

In other words Code::Blocks uses the Global setting plus the Local project settings. For Code::Blocks it is safer to leave the global setting blank and only set the project compiler options to avoid conflicts. All IDEs have their differences and quirks.

For all of our C exercises in this book I will be static linking to any libraries. The exercises only use the standard library so these will be compiled into the final executable.
In Book 3 - Libraries Overview I will go into more detail about using static and shared libraries.

The drop down menu at the top selects the target executable type.

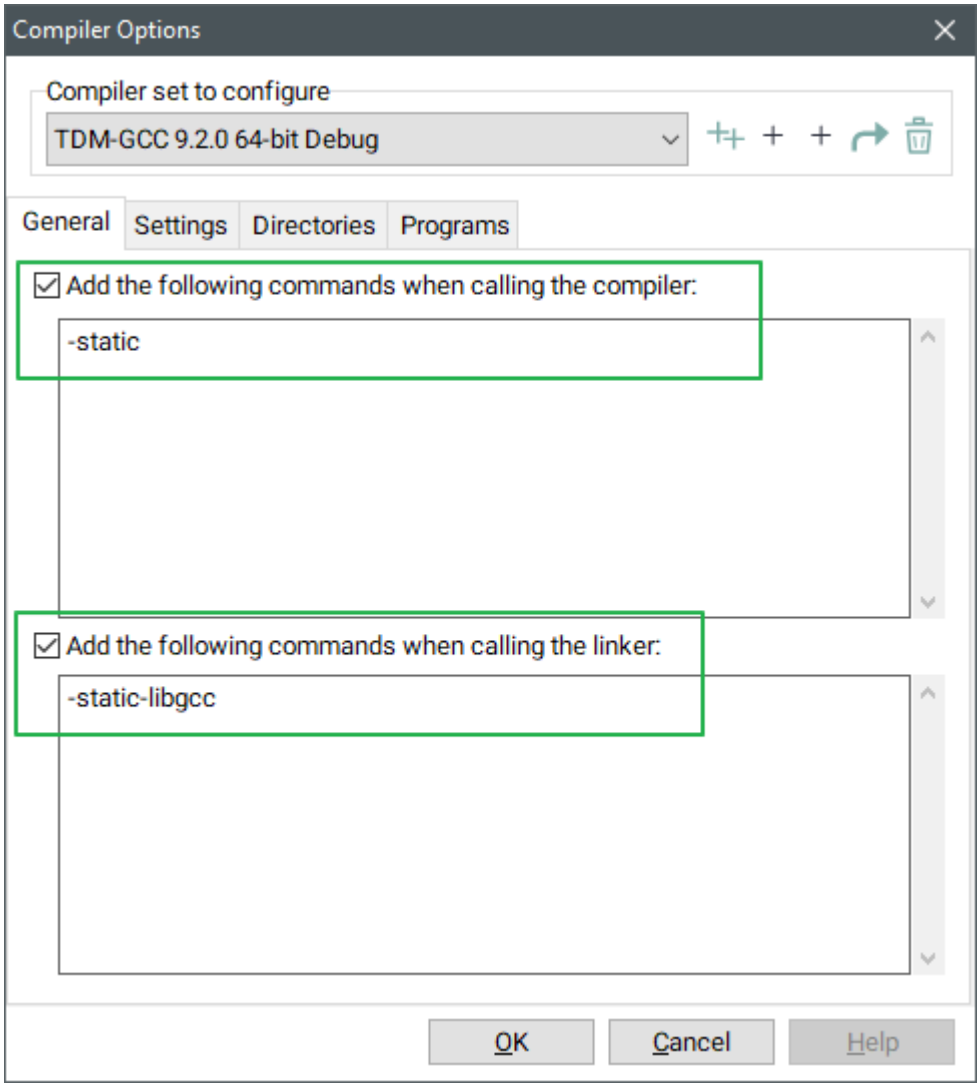
- Release is the final exportable exe with size and speed optimisations.
- Debug includes special switches to enable error and warning checks.
- Profiling allows us to test code performance and optimisations.

We will spend most of our development time in the Debug profile.

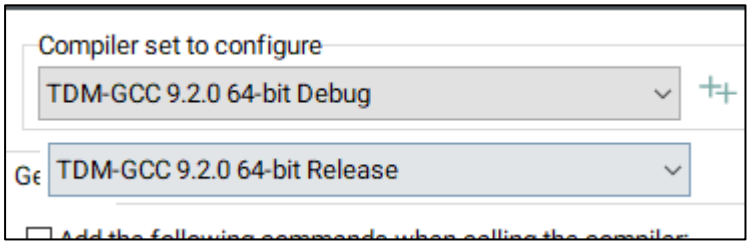
First select the **TDM-GCC 9.2.0 64-bit Release** profile from the dropdown. Most of the default settings should be OK, but we will check anyway.

The lower 2 text areas allow us to add custom compiler and linker options if we need to. This will be reflected in what is known as a “Make File”. A make file is essentially a build script that is similar to a batch file. Compiler languages traditionally work from a command line environment and we would have to create the makefile (build script) manually from the GCC documentation. One of the benefits of an IDE is that they make the most common settings available via a GUI config panel.

Select the General TAB and ensure that the default **[/] Add the following when calling the linker**. Is checked and the script line **“-static-libgcc”** is in the text area. This means that the libgcc runtimes are stored within the final exe rather than linking to a dll runtime library and is the default for C exception handling.



You will need to check/set the above settings for all compiler profiles!

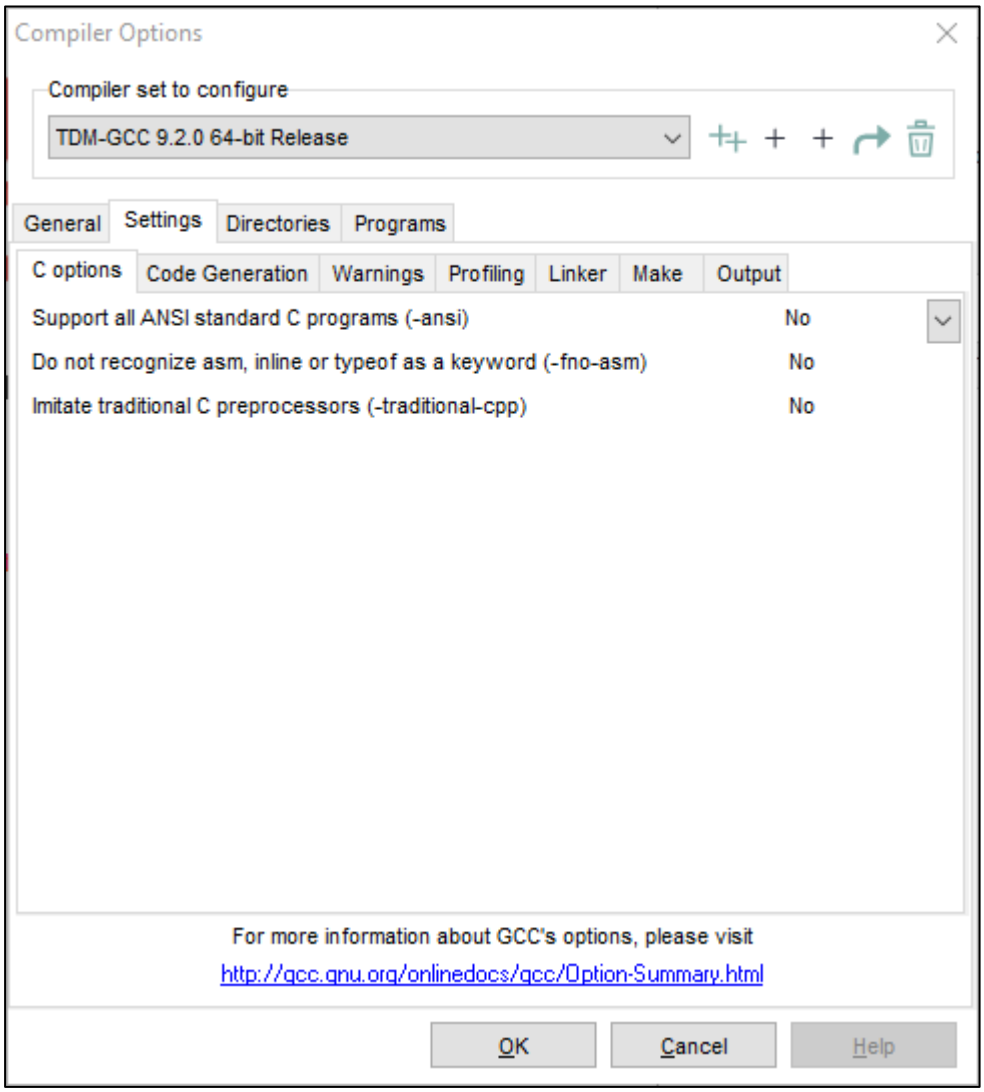


[-static-libgcc] will compile the standard C libraries into the final executable so that you don't have to transport the Runtime libraries with your project. In most instances the Shared objects may already be installed on the target system. If you choose to link "Dynamically" you will need to export your final executable projects with the appropriate MinGW DLLs. They can be found in the .\mingw64\bin directory. libgcc_s_sjlj-1.dll and libwinpthread-1.dll are the most common runtime DLLs.

[-static] Is much the same as above and will force most additional libraries to be included in the final executable. This will fail on systems that only have a shared object dll.so available and no library

archive .a. Again, this just means that you may need to install those shared libraries on the target computer.
The -static flag is entered as part of the compiler switches under Project setting at a stage further on. If you don't use static you will need your application will link to the shared libraries for you project. You will need to have a copy of any DLLs in the system path or in the directory alongside of your project executable to run. You will also need to make any dependent DLLs available when you export your application to another computer.

Select the settings TAB.



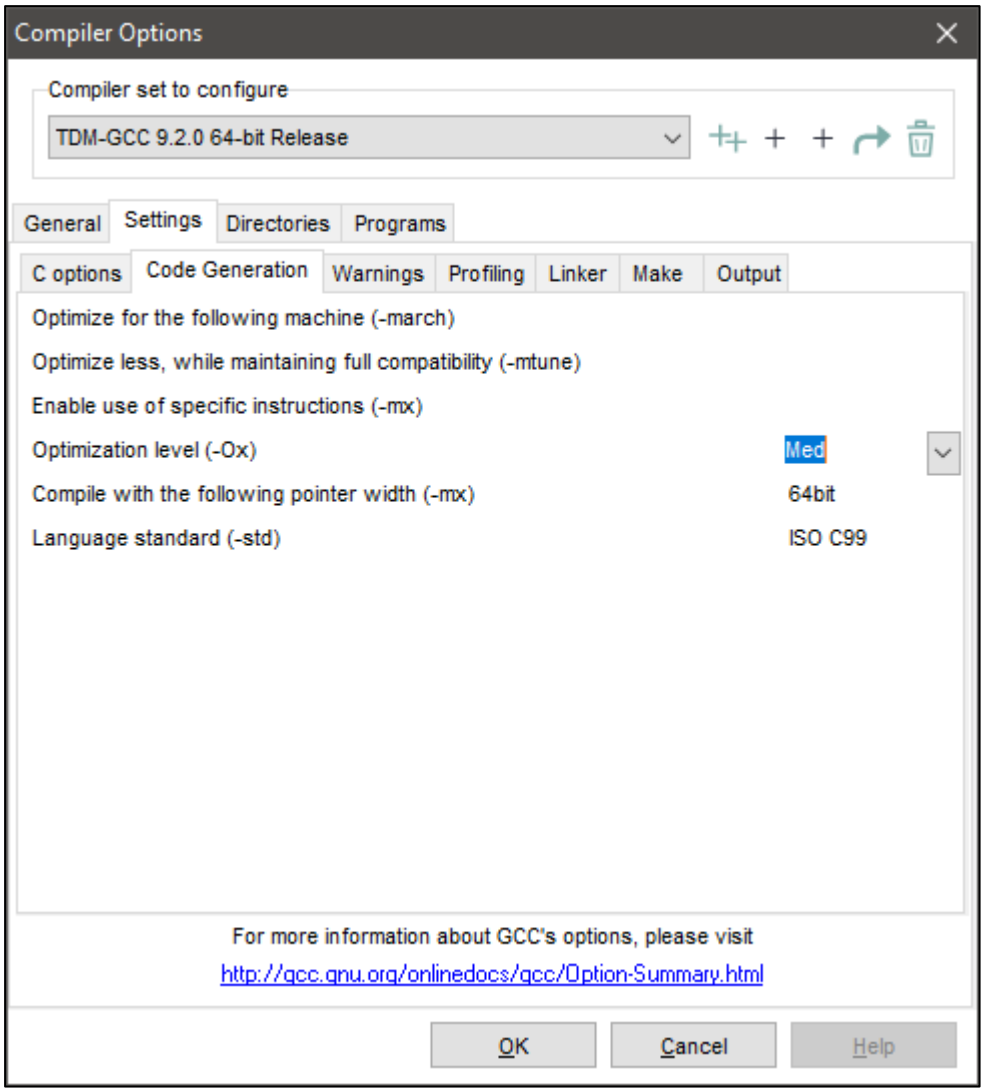
And next for "Code Generation". I am going to target 64 bit systems so I will set the default settings for this.

On the right select the Optimization level (-Ox) to Med. This adds some performance improvements to the release executable. We can choose options from slow and most stable to fast and less stable. A Med selection is usually safe.

In the next option Compile with the following pointer width select 64bit to output executables for 64 bit Windows target OS.

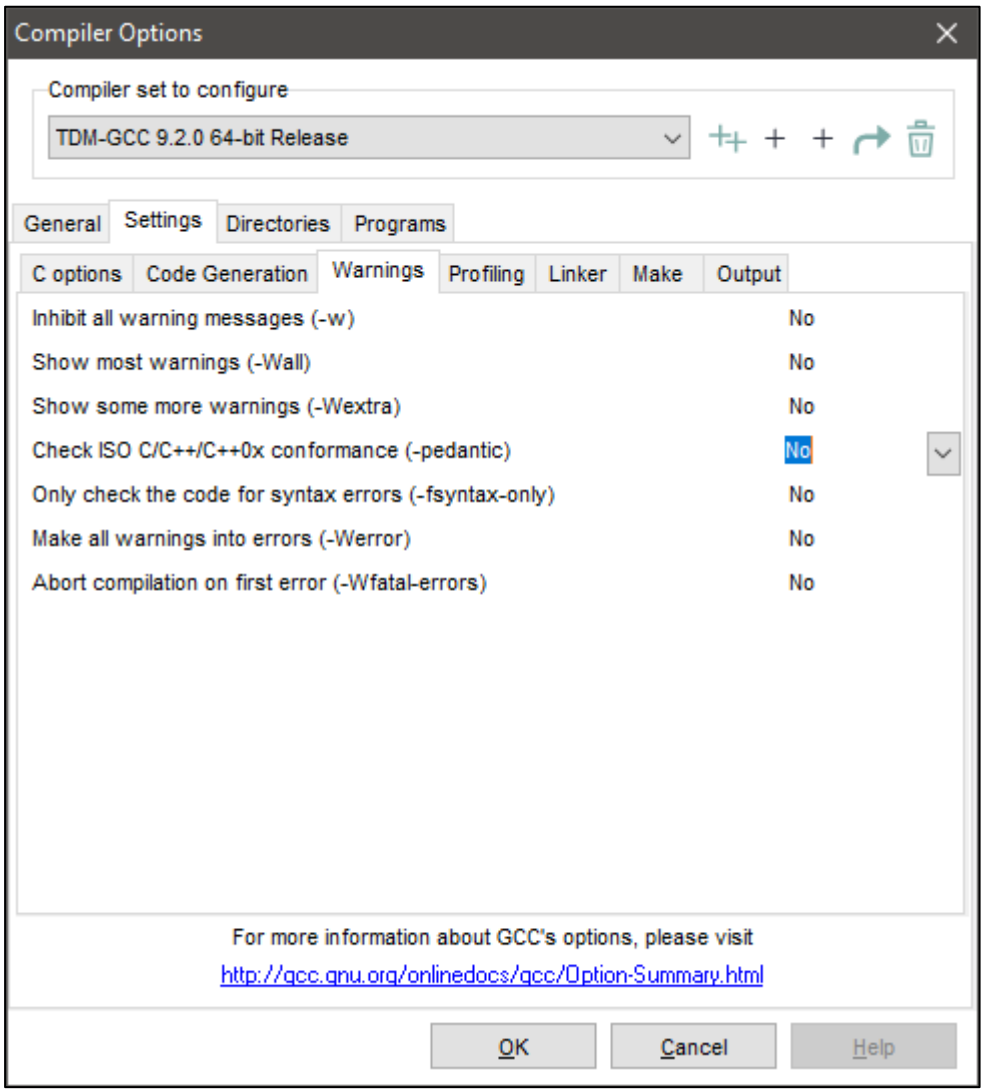
And set Language standard (-std) to ISO C99

1999 ISO C [-std=C99] is the highest standard available that is compatible with the MSVCRT.DLL. All source code that you write on Windows with GCC and MinGW will follow this standard, so I am setting it the same so that your code can be easily cross compiled between Windows and Linux. Windows 10 onwards can use the (UCRT Universal C Runtime). This is a different compiler and library in Windows, and I am not yet certain of the Linux compatibilities.

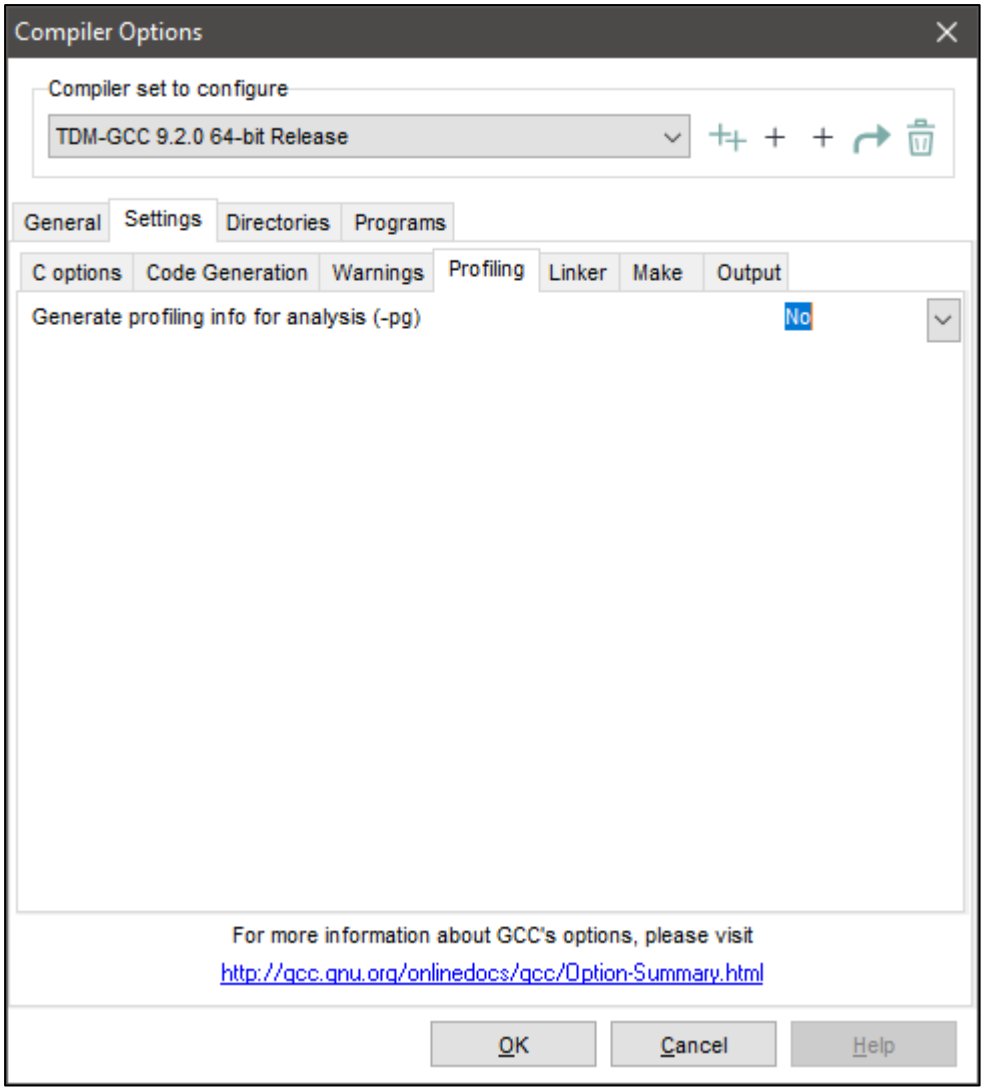


Optimize settings allow for faster and smaller executables but depends upon the target machine. It is a balance between speed and runtime stability.

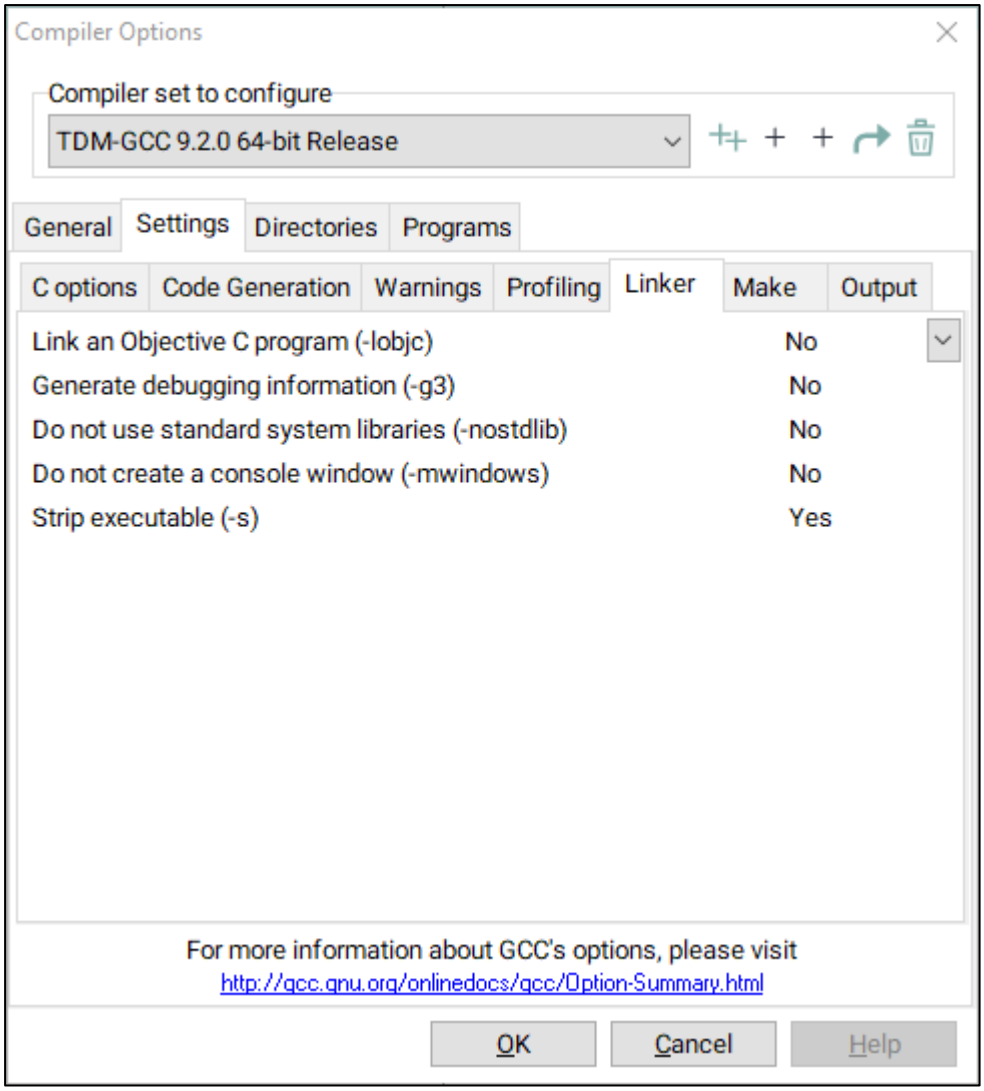
In the Warnings TAB all selections should be set to No as we use the Debug profile for debugging. The release version should be already well tested and bug free by this stage.



The same with all other TABS the default will be with profiling and debug setting turned off.



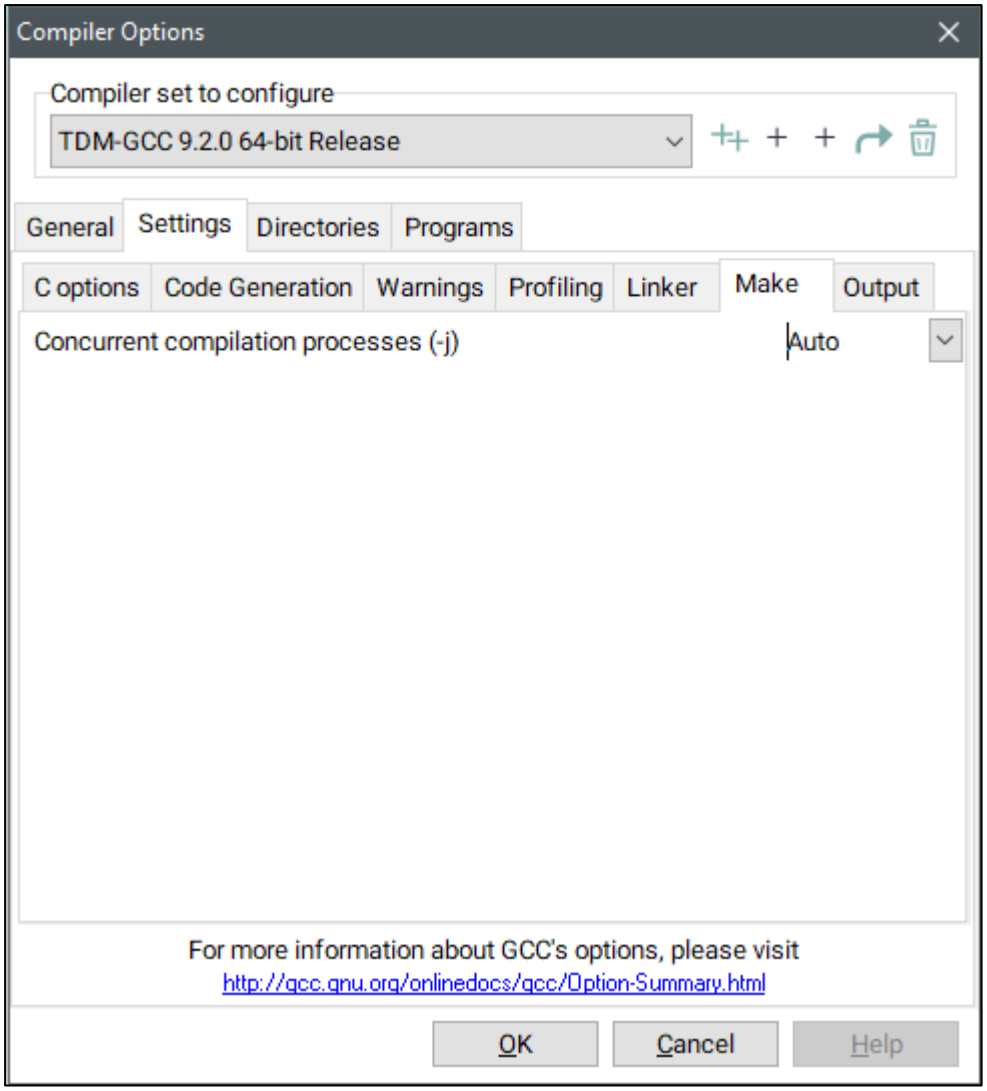
[-pg] This generates additional code that allows us to use applications such as “gprof” to Profile our application at runtime. This allows us to test many performance aspects or our final application as well as the performance of individual routines. Performance profiling is an advanced technique that won’t be covered in detail in the “Beginners Guide To Programming” series.



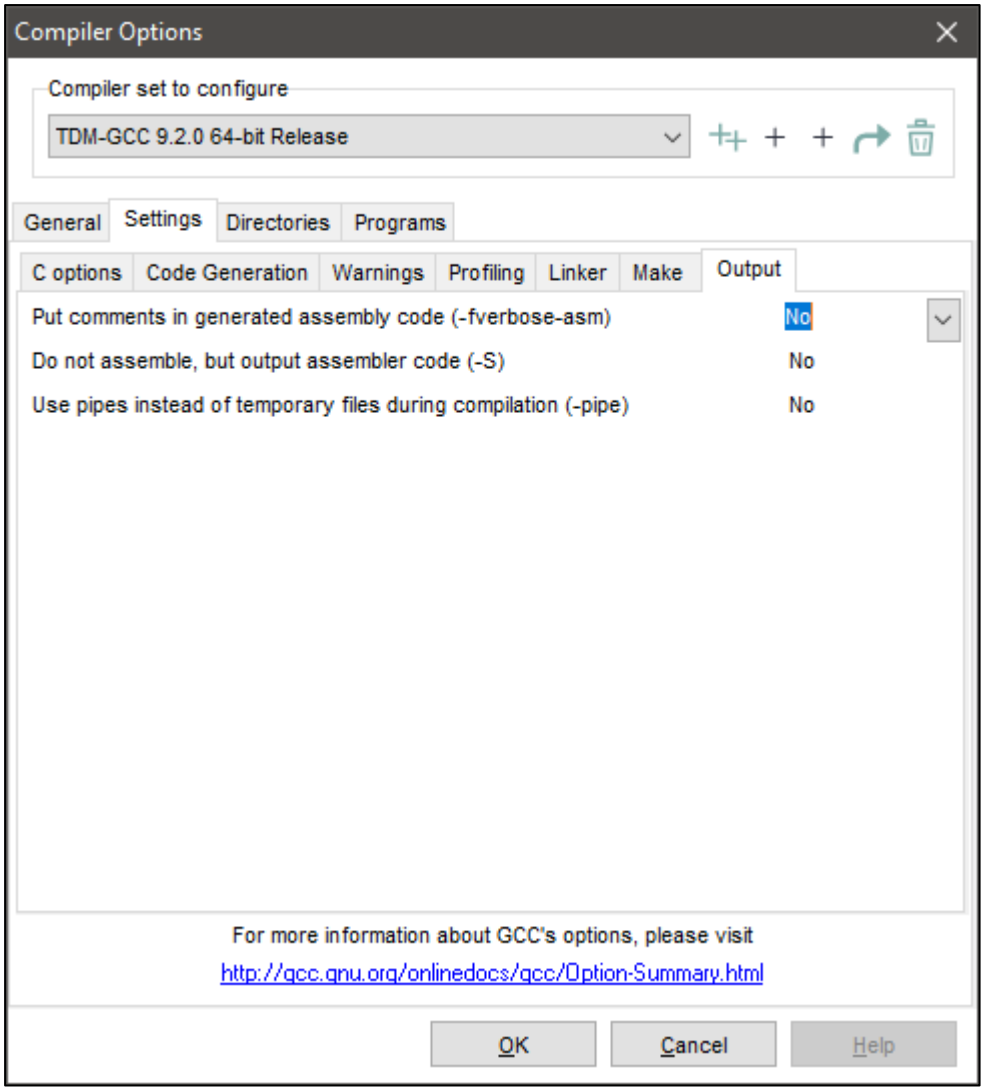
[-mwindows] will suppress the console window from opening when you create GUI or Graphical windows.

[-s] Will strip all comments and unimportant lines from the final executable to make it smaller.

The following can speed up build times by using multithreading if you have a large project. You can leave it blank for single threading, or set it to Auto and GNU Make will make use of multitasking if it is available. None of the example applications in this books will be complex enough to require this.



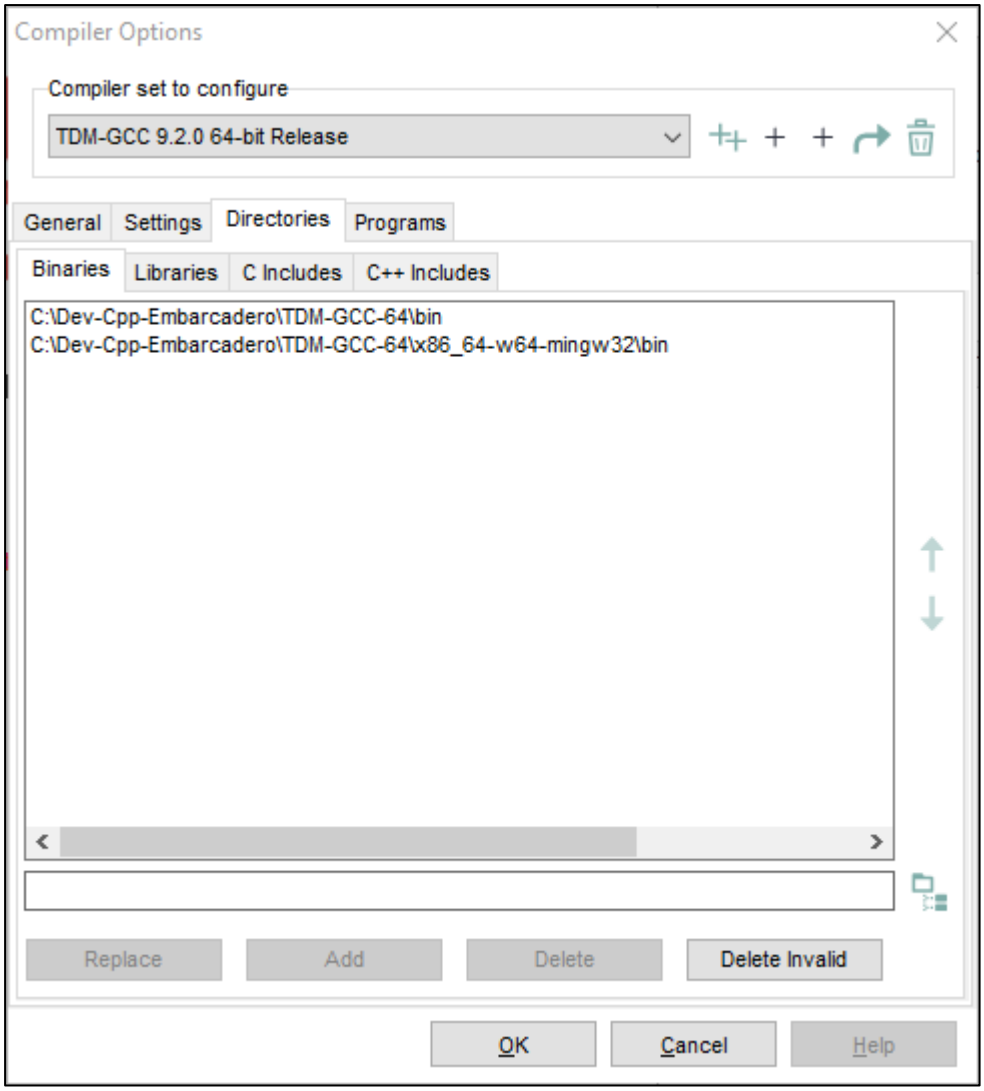
The following are advanced options for working with Assembly language. Note that the flag “-s” and “-S” are not the same as Unix and C languages are case sensitive.

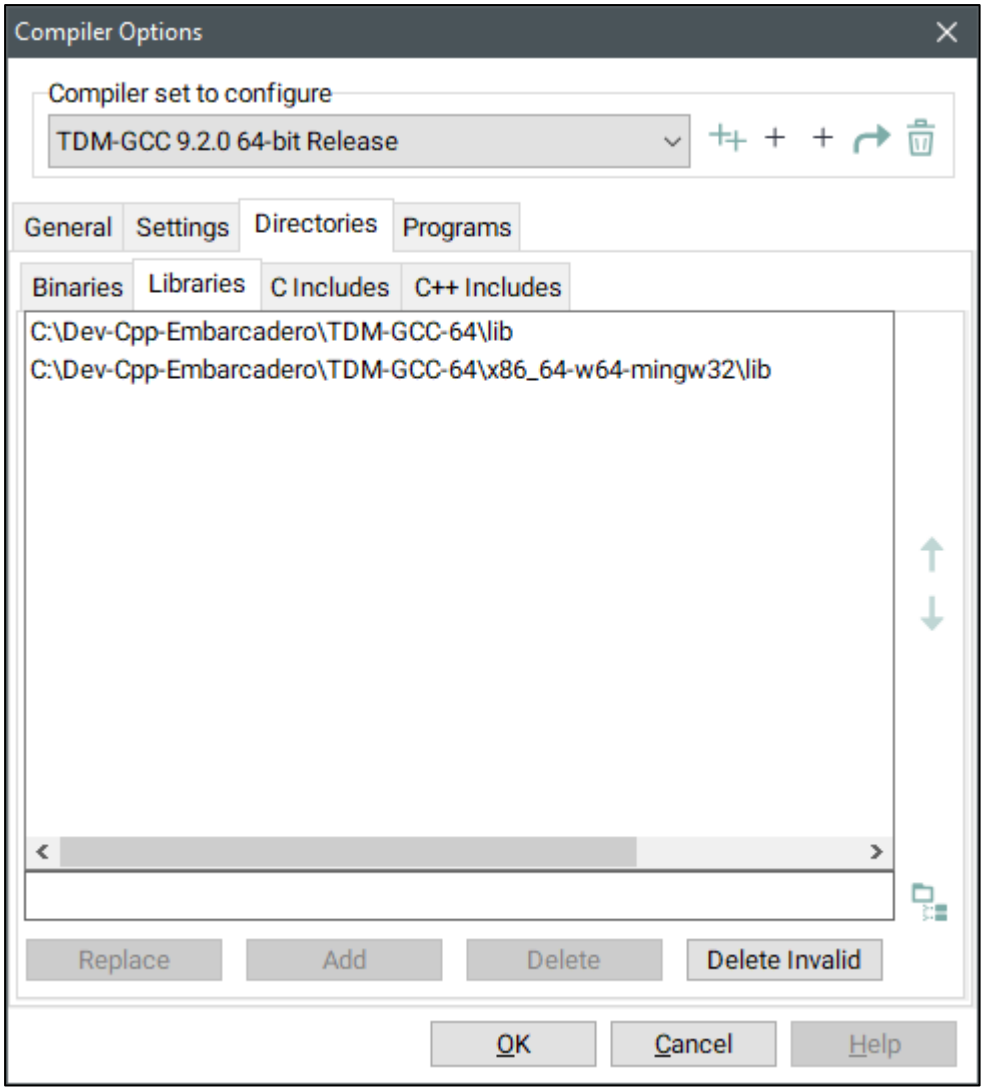


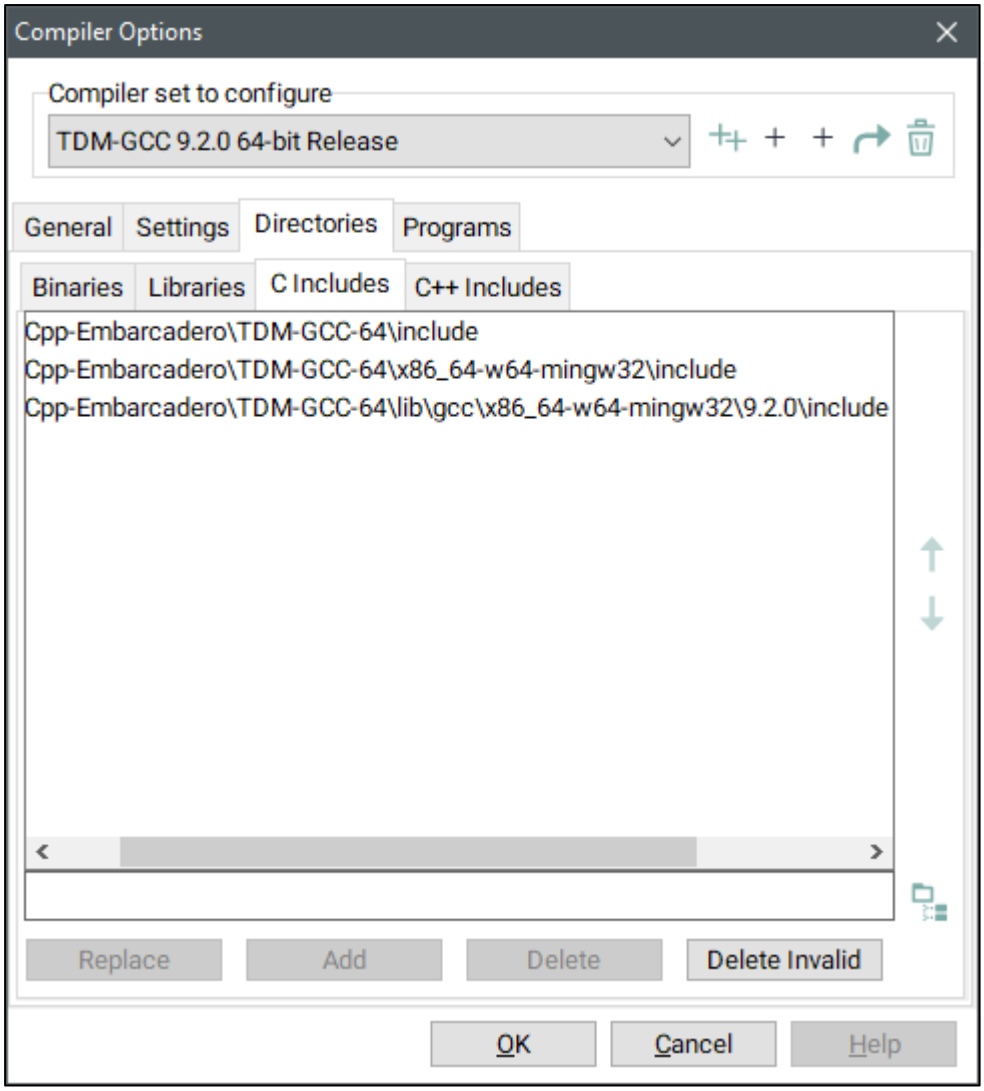
Directories TAB.

The default settings do not need to be altered. This is the equivalent to the “Path Environment” where the application finds all required files. Occasionally it may be required to add additional paths if for instance we add a 3rd party library to a folder structure outside of the defaults.

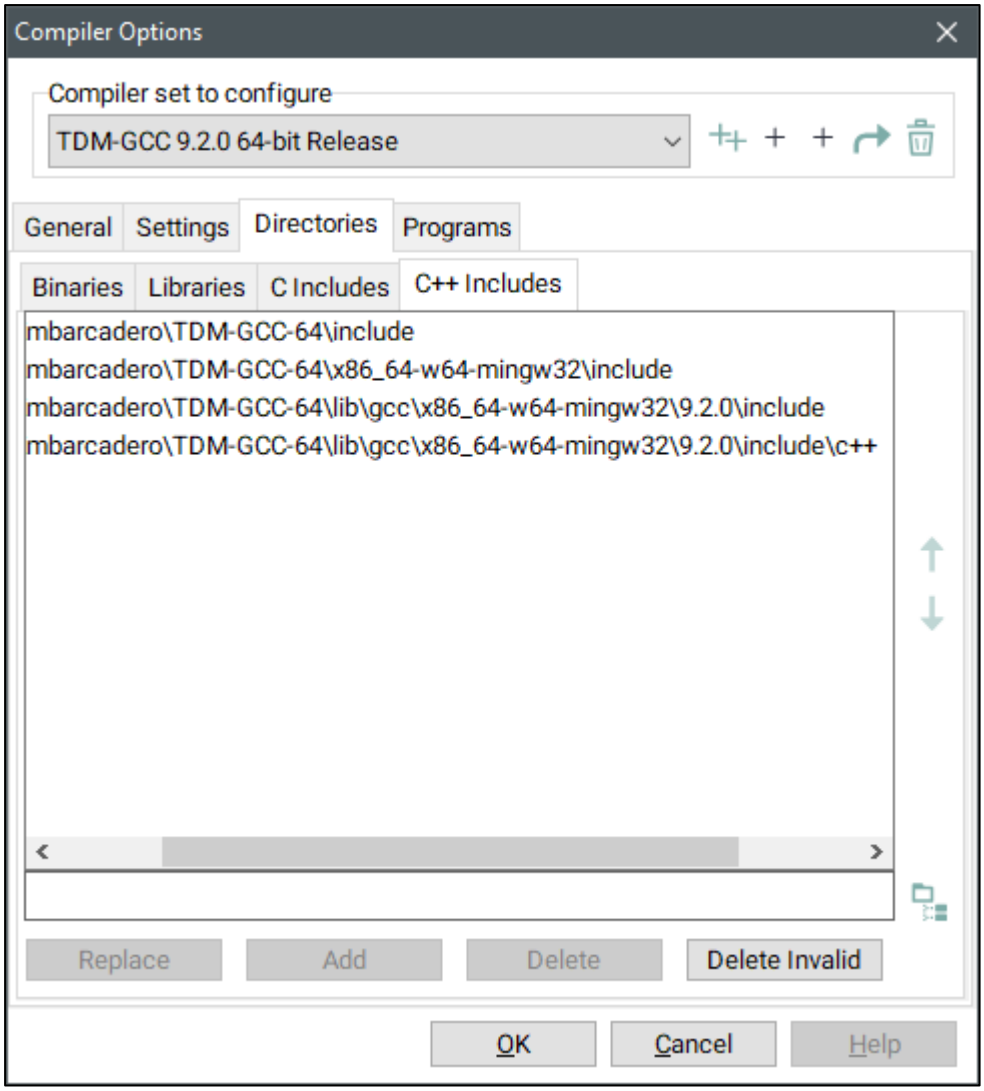
C and C++ use a different compiler and different libraries and this is why both are shown.


Compiler Options
Compiler set to configure
TDM-GCC 9.2.0 64-bit Release
General
Settings
Directories
Programs
Binaries
Libraries
C Includes
C++ Includes
C:\Dev-Cpp-Embarcadero\TDM-GCC-64\bin
C:\Dev-Cpp-Embarcadero\TDM-GCC-64\x86_64-w64-mingw32\bin
Replace
Add
Delete
Delete Invalid
OK
Cancel
Help


Compiler Options
Compiler set to configure
TDM-GCC 9.2.0 64-bit Release
General
Settings
Directories
Programs
Binaries
Libraries
C Includes
C++ Includes
C:\Dev-Cpp-Embarcadero\TDM-GCC-64\lib
C:\Dev-Cpp-Embarcadero\TDM-GCC-64\x86_64-w64-mingw32\lib
Replace
Add
Delete
Delete Invalid
OK
Cancel
Help

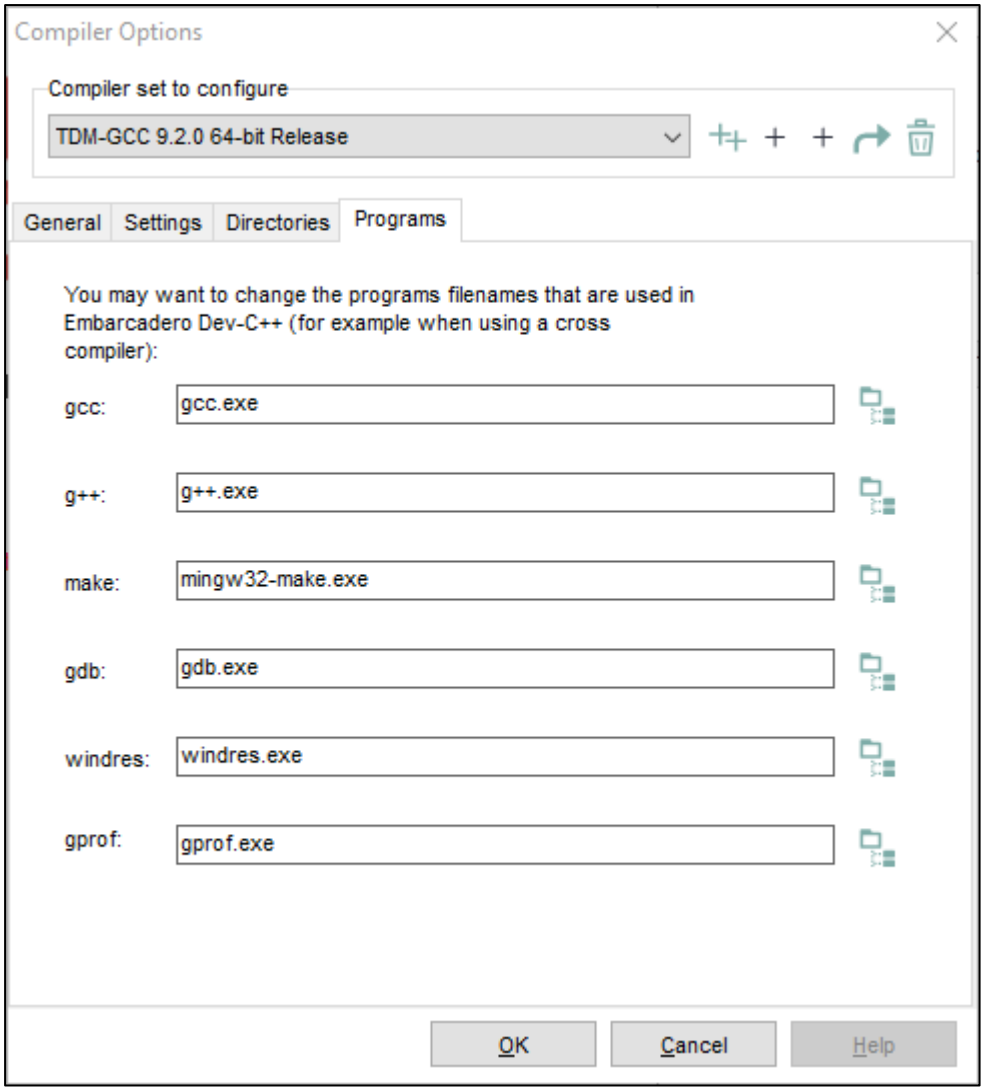


For C++ projects. Not used in this series of books.

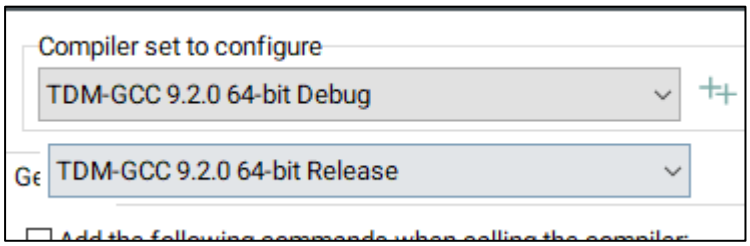


Programs TAB

These are the default compiler executables and should not need to be changed from the defaults.



Next, select the TDM-GCC 9.2.0 64-bit Debug profile from the dropdown.

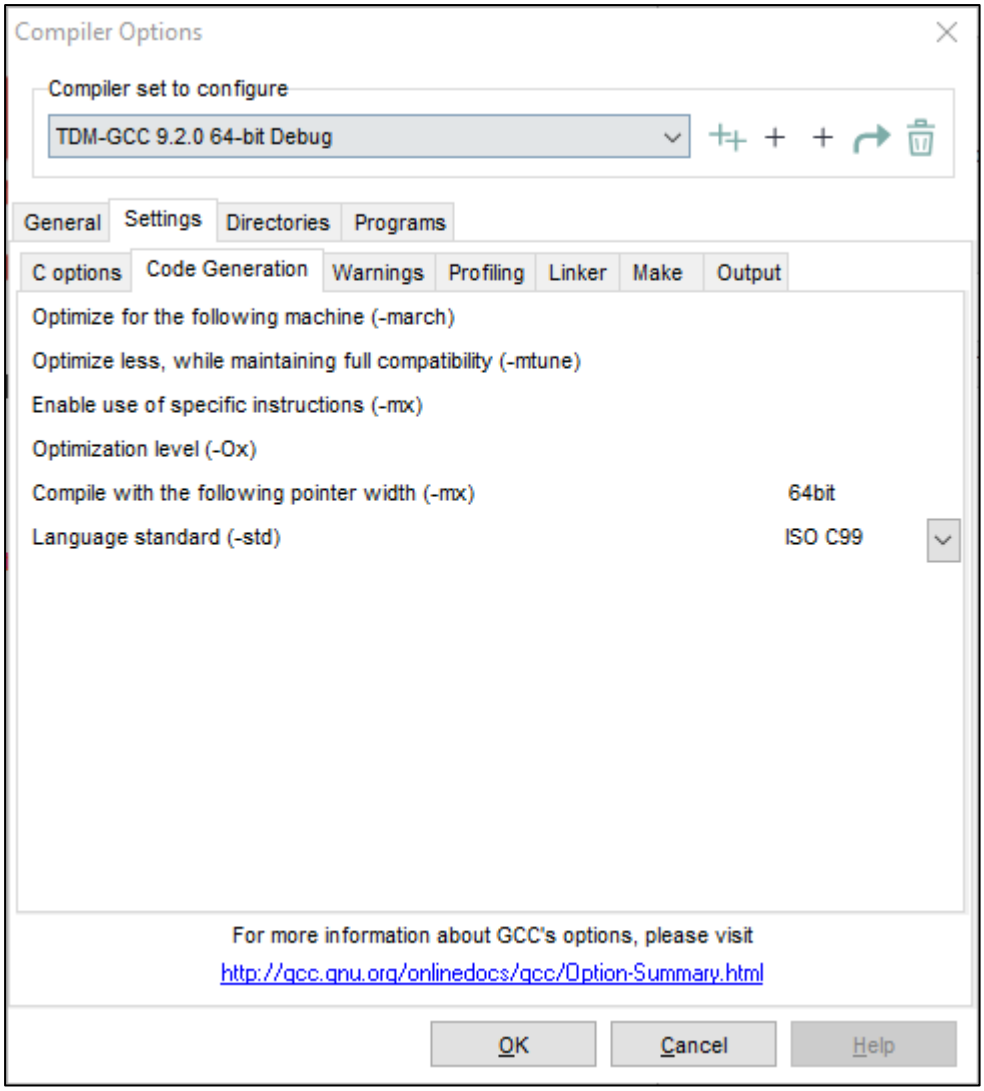


We will check and make some of the same adjustments as for the Release profile.

Under Settings – Code Generation set the following 2 options.

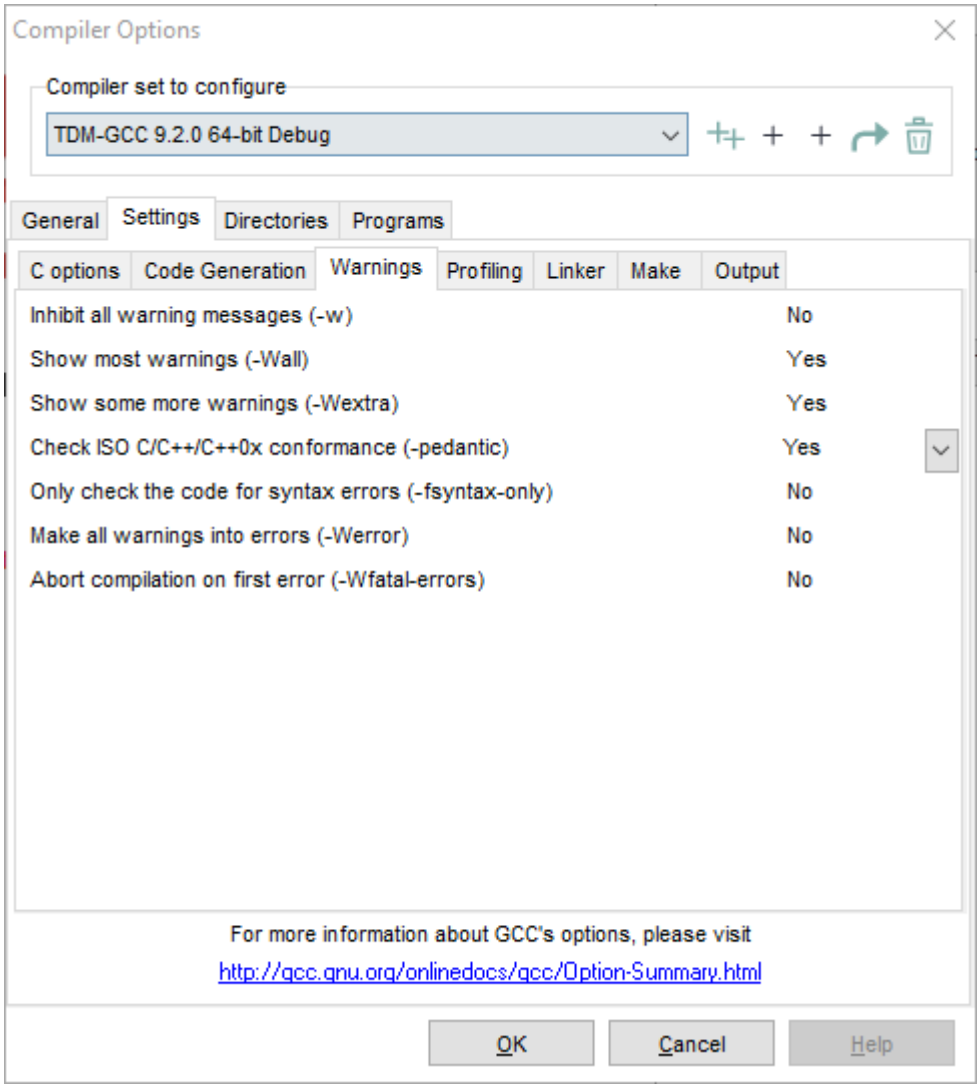
Compile with the following pointer width (-mx) to 64bit

Language standard (-std) to ISO C99



The following 3 settings will give very pedantic warnings about coding syntax mistakes as well as holding the coder to strict coding standards. Not all warnings will mean that your code is broken or not able to run, but are just warnings that sometime undefined behaviour or runtime faults can occur.

Under the Warnings TAB set all of the options according to the following image.



All other default settings will be fine.

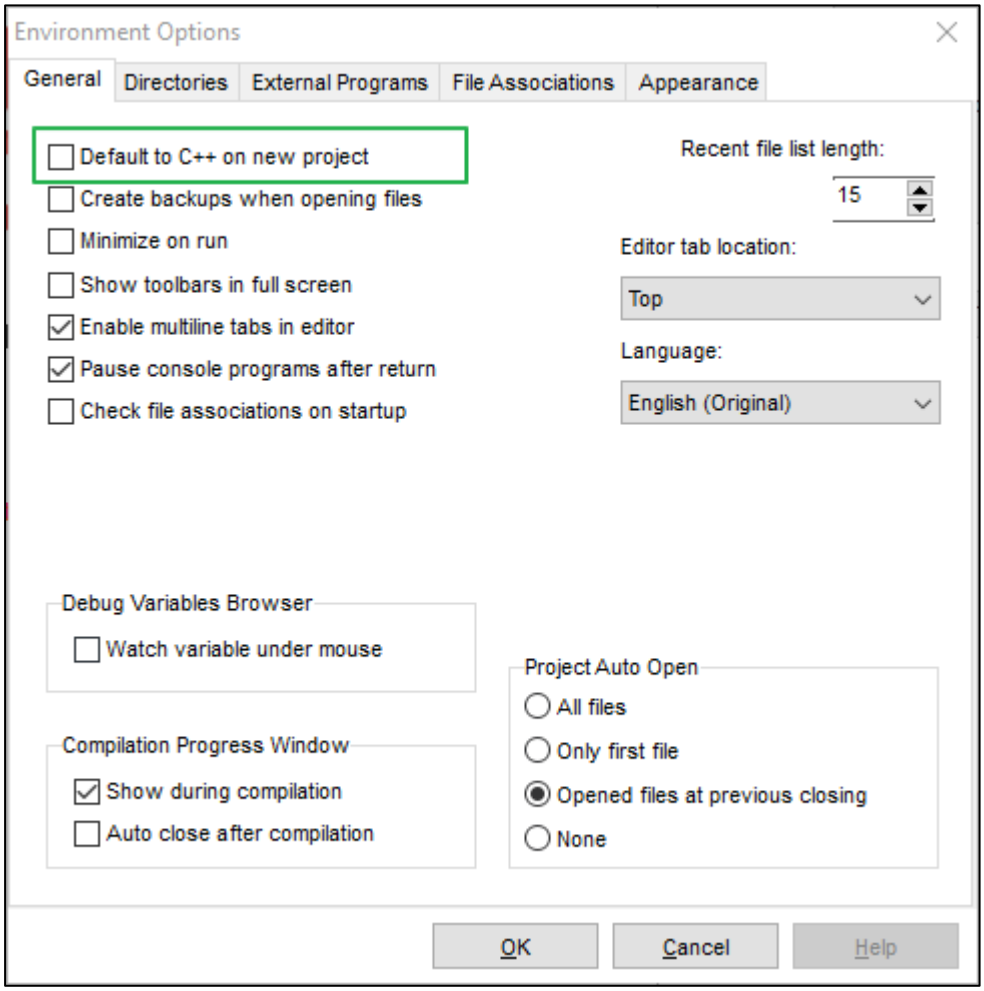
If you wish to check the directories paths, they should be the same as for the Release profile

This is the “Global” compiler options. We don’t need to do the above stage when we create a project, as they are also set for individual projects when we create them. These default settings will reflect in the project setting when creating a new project so we don’t need to repeat these settings for every new project.

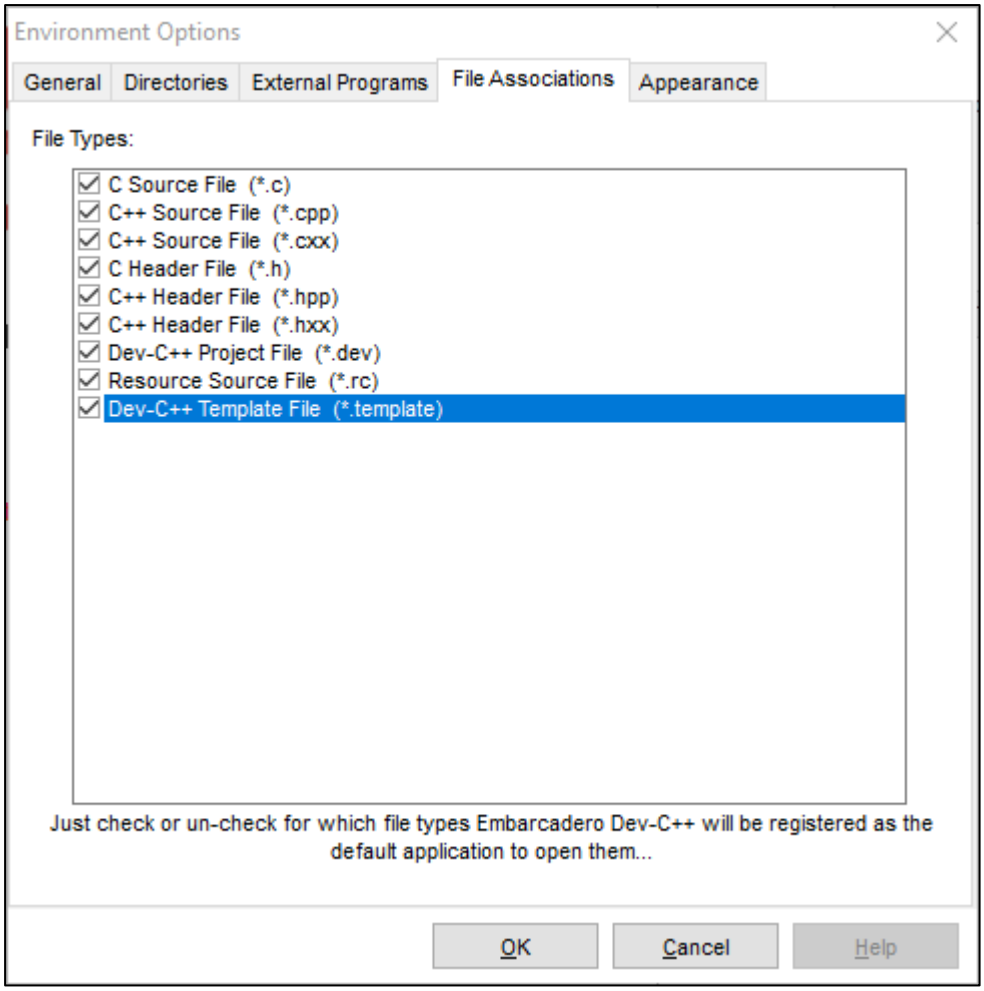
Close the by pressing [OK]. This will save the currently entered settings.

Next open Tools -> Environment Options…

Have a look around the options without changing them. This is where we set up the general look and feel (Theme) of the IDE. Change the following settings as shown in the images.



Select all file associations. Unfortunately this only works in the full install with the relevant registry settings enabled so you can skip this step in portable mode.



Close any options windows.

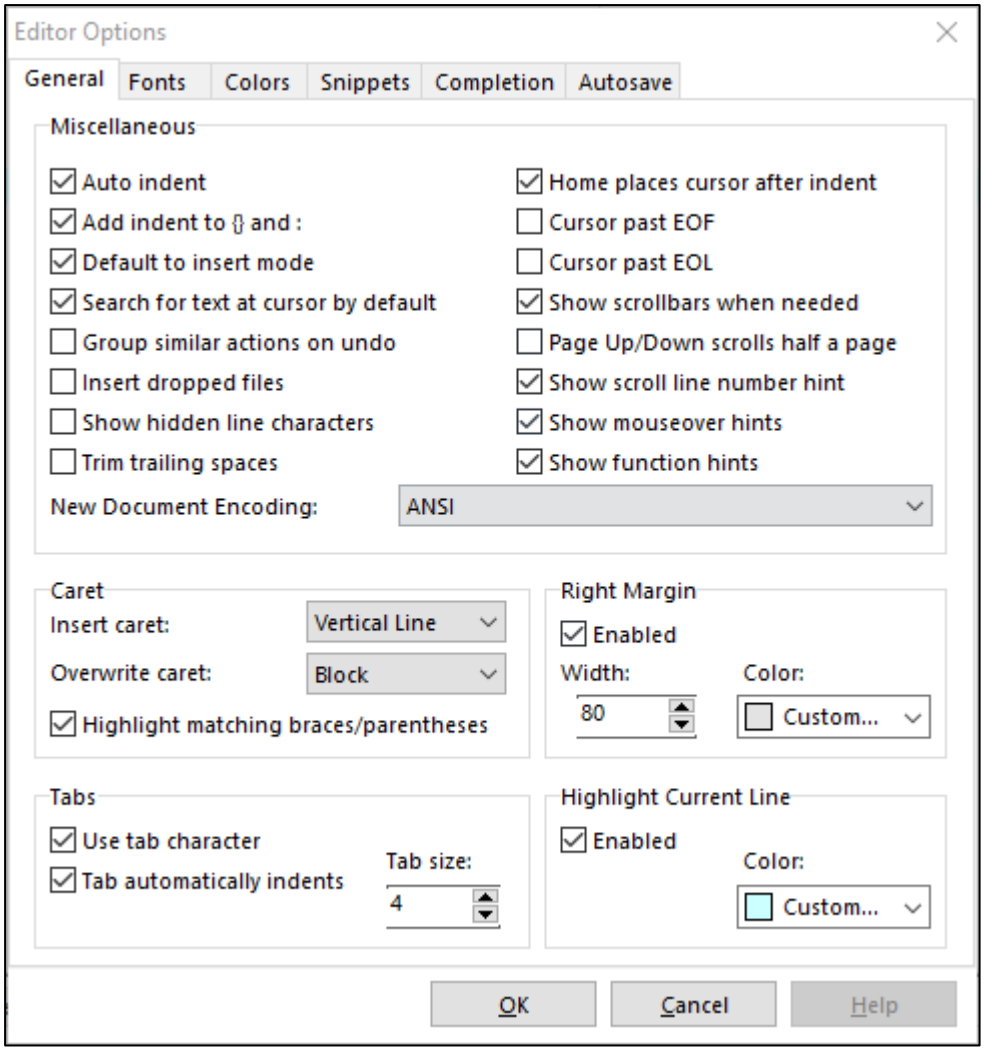
Next, open Tools -> Editor Options…

This is where we set the behaviour of the source code editor. You may find that you will quickly develop your own personal preferences here, but the following setting will get you started.

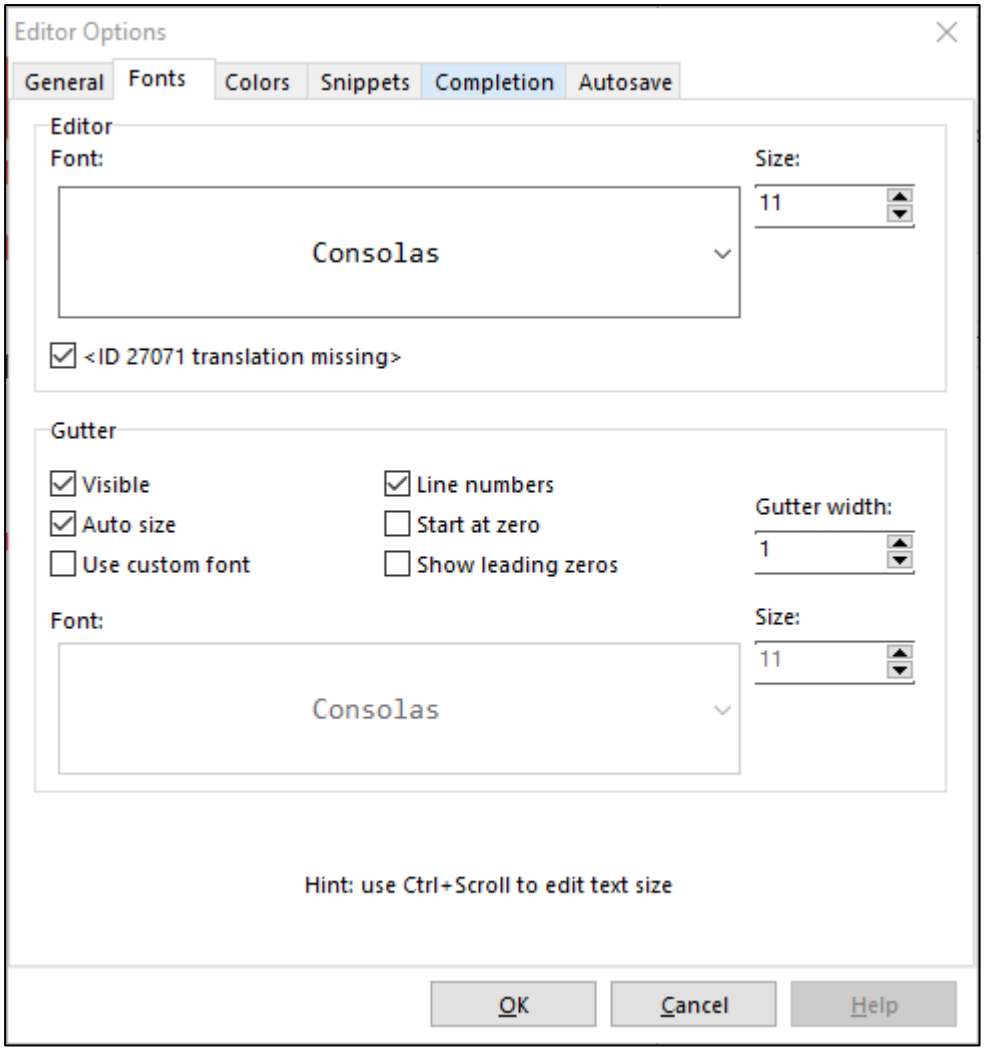
Set the editor options to match the following images.

The default English characters for UTF-8 are identical to the ANSI set.

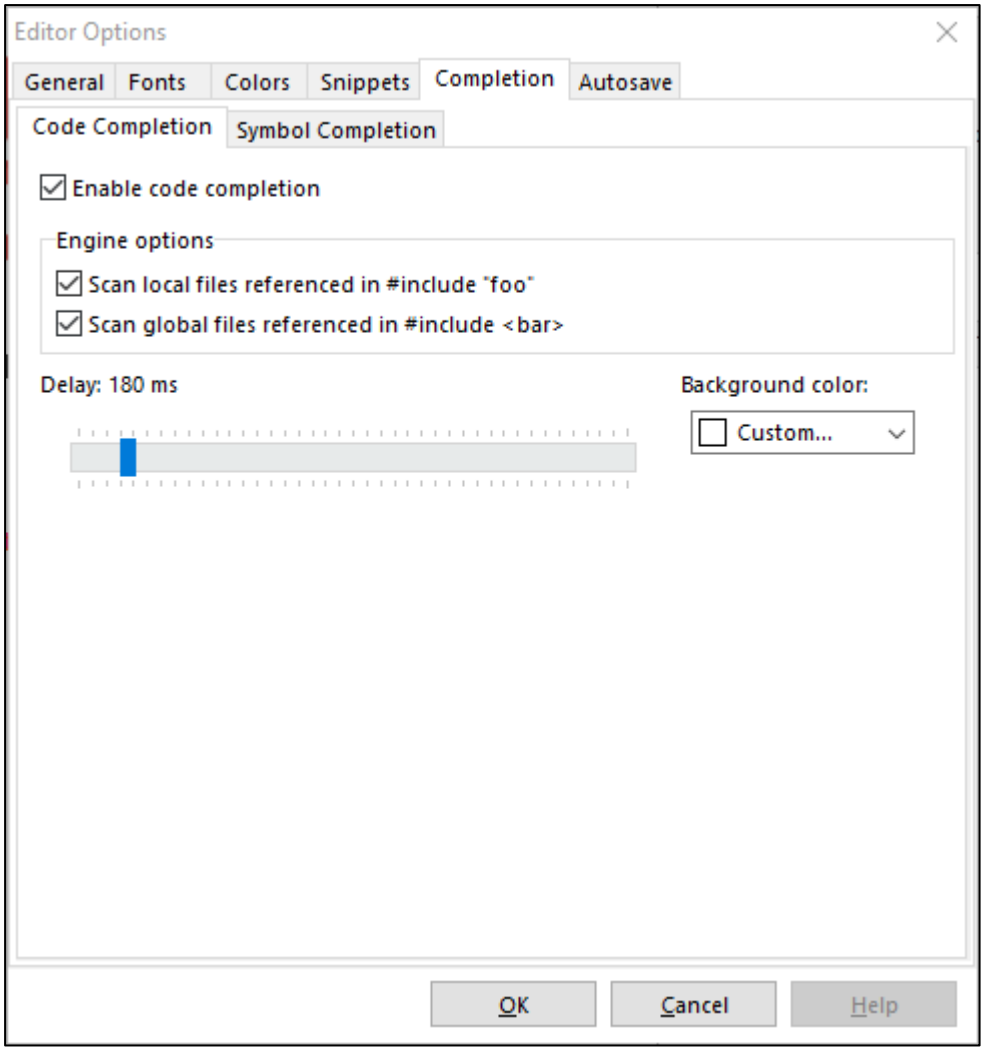
Historically console windows were 80 characters wide. This still remains a basic standard for writing code to keep code readability standards.



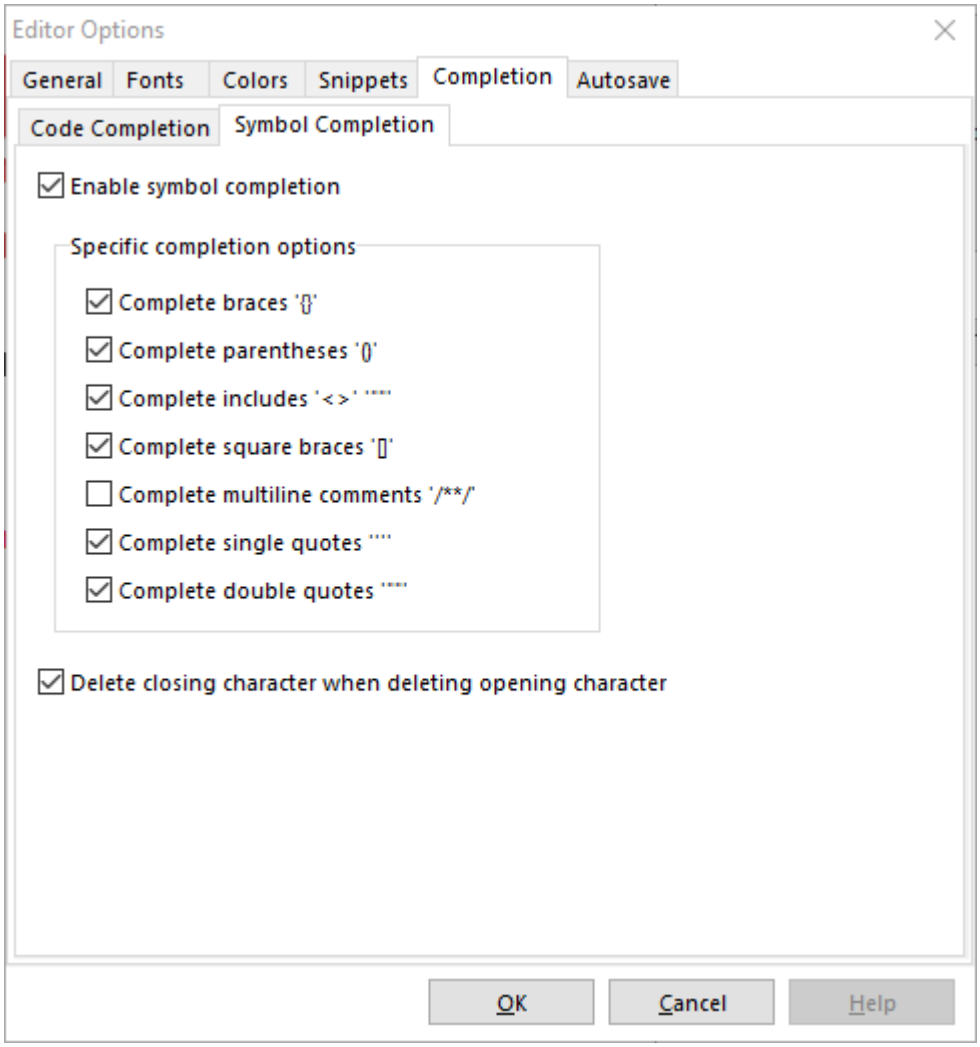
I often change the font size from 10 to 11 or 12 to make it easier on my eyes.



Under the TAB Completion, take note of the current settings. Sometimes auto completion can be helpful and sometimes it can be annoying and distracting. If at any point you find the auto complete options are annoying come back here and switch off **[ ]Enable code completion** or **[ ]Enable symbol completion**.



The following allows for auto completion of coding symbols. All opening symbols must have a closing symbol, so this is a bit of a safety mechanism to help you always close different symbols. Personally I find them a little annoying and turn them off, but sometimes I forget to close them. The compiler will give you a warning if you forget.



Auto save TAB will allow you to automatically save your source code as well as create time stamped revisions. You can experiment with this in the future if you wish.

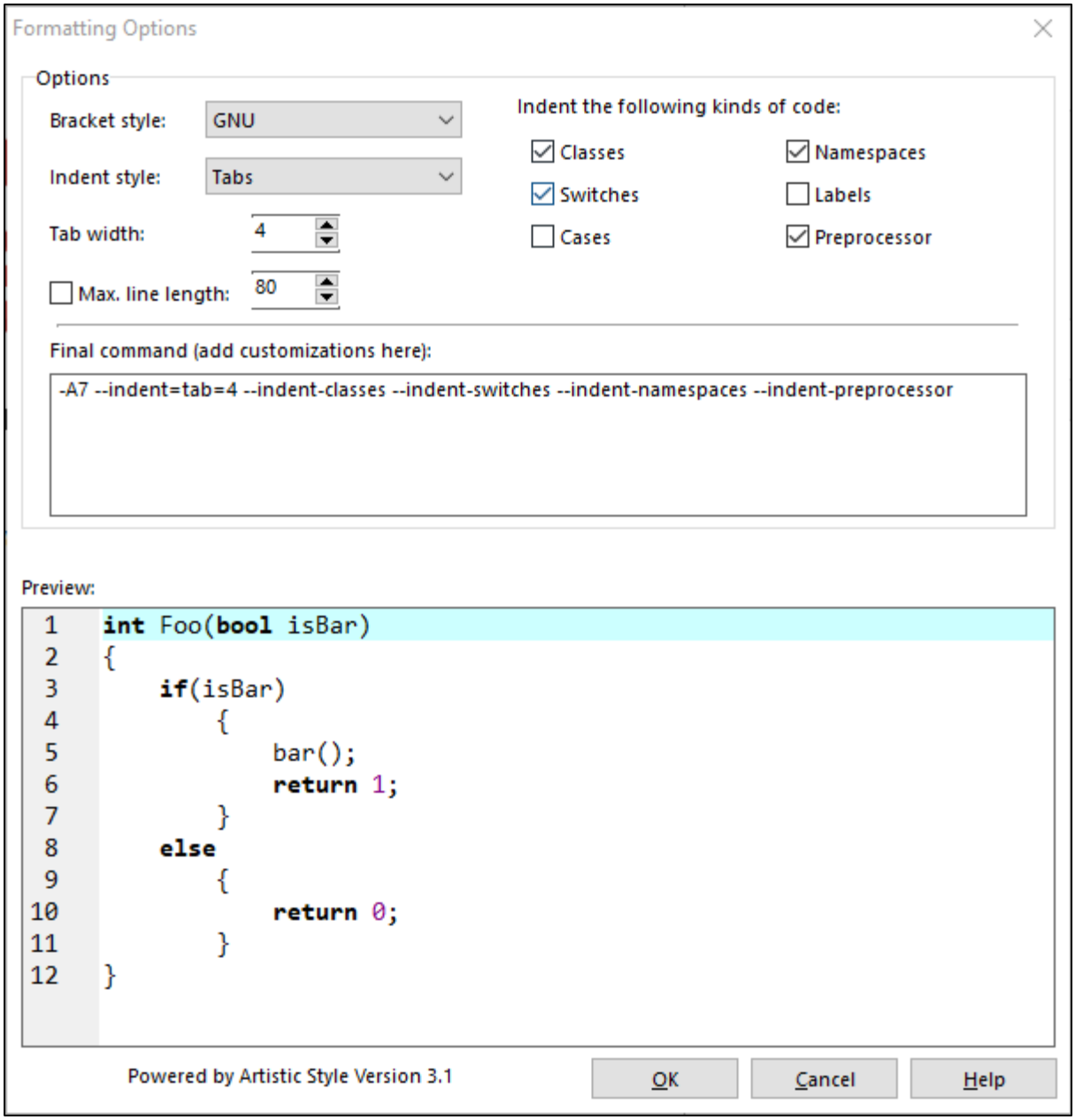
I don't use this function as I use Beanland AutoVer 2.2.1 to monitor all and create revisions of my source and document projects.

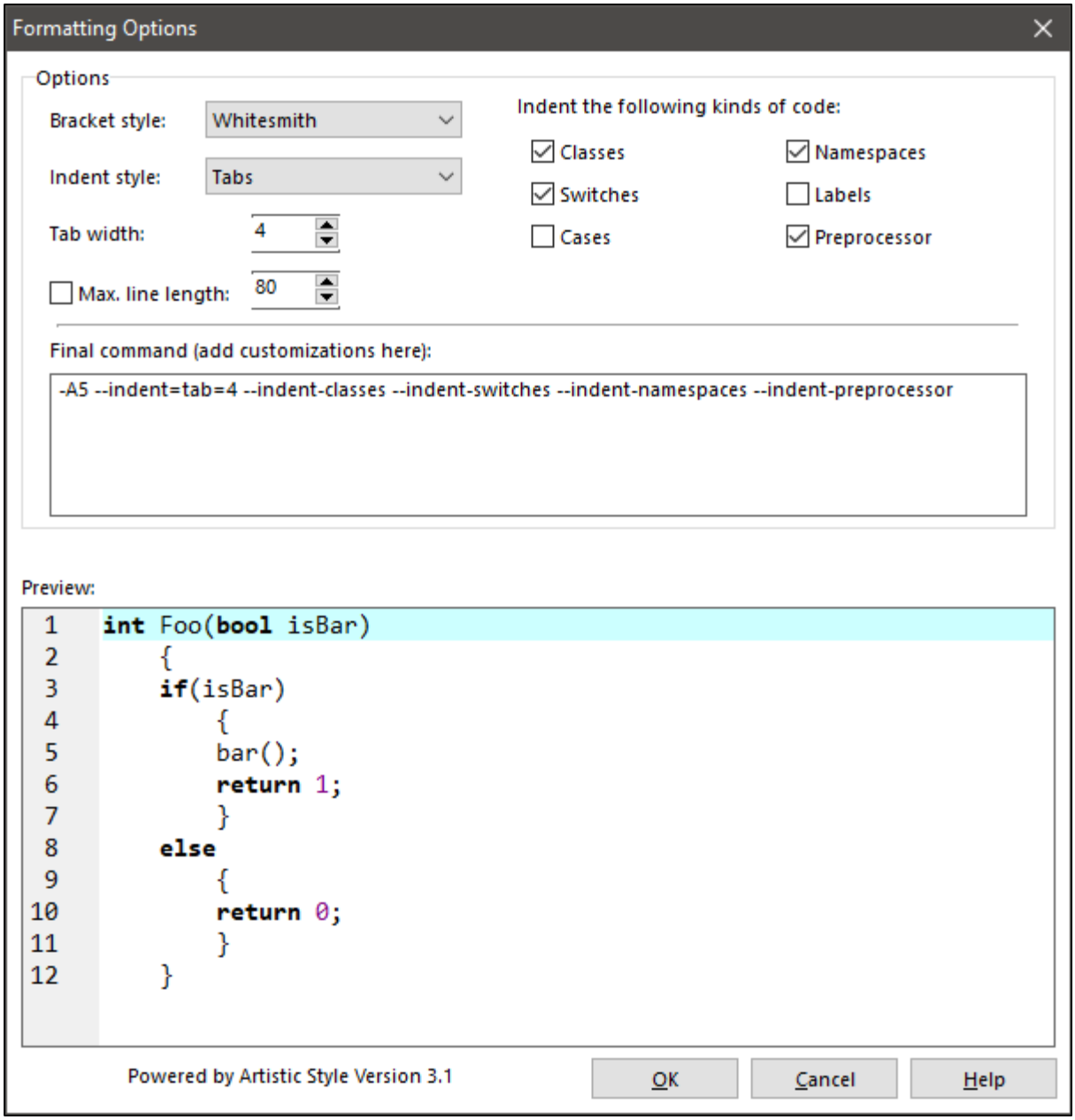
The next setting to look at is AStyle.

AStyle will automatically styles your source code and indentation according to the inbuilt rules that you select.

If you Open AStyle -> Formatting options select the GNU Bracket or Whitesmith Style from the drop down menu. GNU is the easiest to follow when beginning.

I have used Whitesmith in all of the examples in my other guides as it is more compact with regards to line width. Whitesmith will use an indentation style very similar to that which is used in Python.

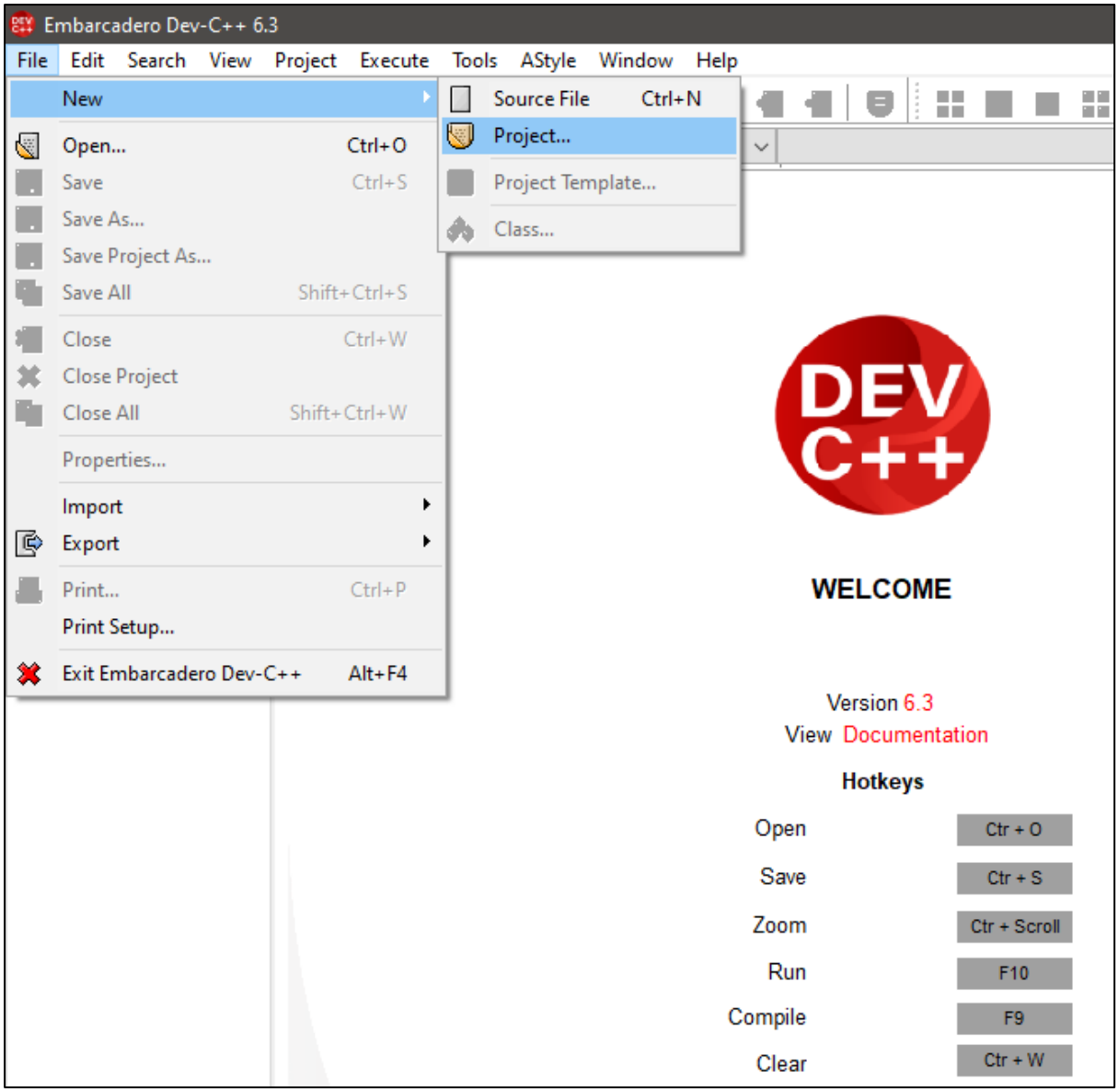

Formatting Options
Options
Bracket style:
GNU
Indent style:
Tabs
Tab width:
4
Max. line length:
80
Indent the following kinds of code:
Classes
Namespaces
Switches
Labels
Cases
Preprocessor
Final command (add customizations here):
-A7 --indent=tab=4 --indent-classes --indent-switches --indent-namespaces --indent-preprocessor
Preview:
1 int Foo(bool isBar)
2 {
3 if(isBar)
4 {
5 bar();
6 return 1;
7 }
8 else
9 {
10 return 0;
11 }
12 }
Powered by Artistic Style Version 3.1
OK
Cancel
Help

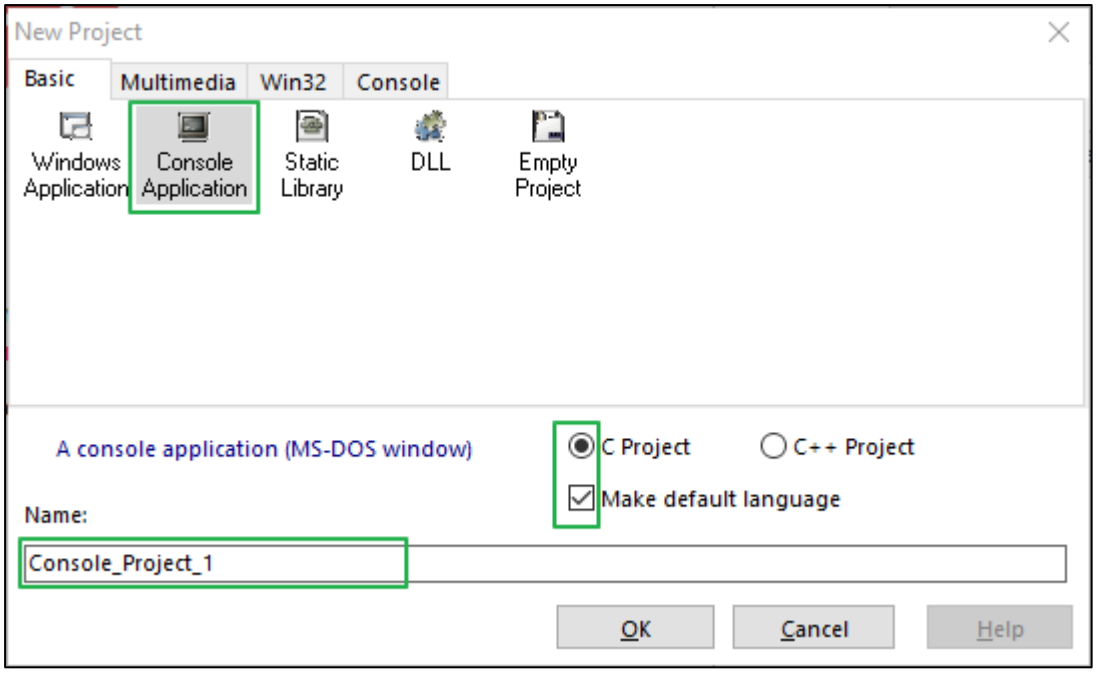


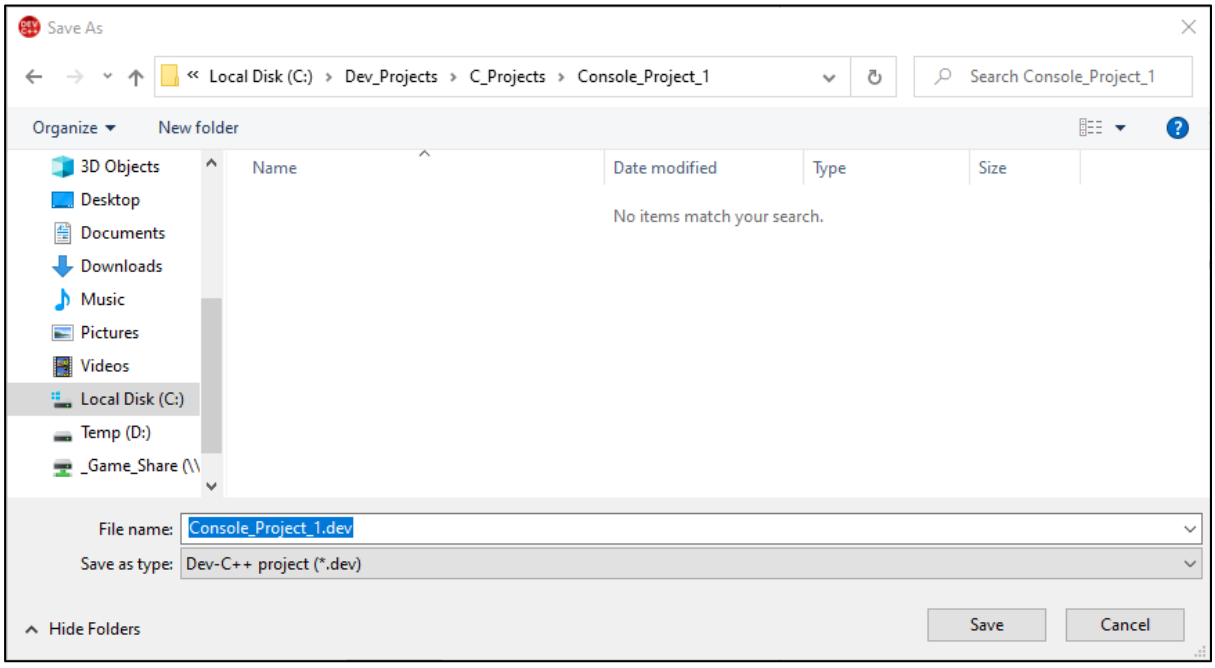
That is all of the essential Global Options set and checked. You can create a source file and compile it from here if you wish. Unlike Code::Blocks, Dev-C++ allows you to compile individual code files outside of a project.

In most cases it is beneficial to create a project. A project keeps track of all individual settings similar to the global settings but allows us to create settings that are specific to the project. In most cases we will just create a project using the defaults taken from the global settings.

Creating a Project.

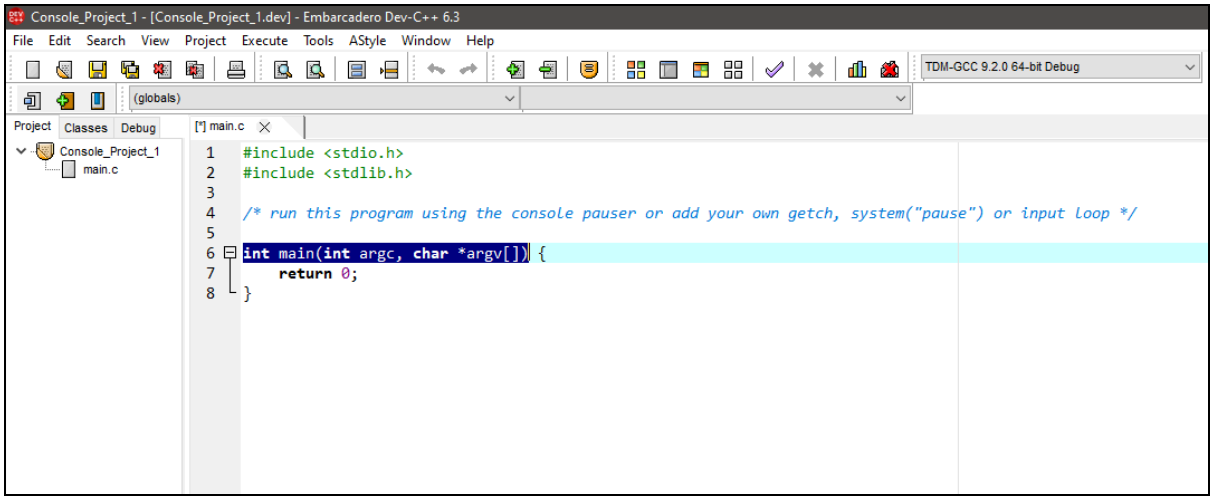

Embarcadero Dev-C++ 6.3
File Edit Search View Project Execute Tools AStyle Window Help
New
Open... Ctrl+O
Save Ctrl+S
Save As...
Save Project As...
Save All Shift+Ctrl+S
Close Ctrl+W
Close Project
Close All Shift+Ctrl+W
Properties...
Import
Export
Print... Ctrl+P
Print Setup...
Exit Embarcadero Dev-C++ Alt+F4
Source File Ctrl+N
Project...
Project Template...
Class...
DEV C++
WELCOME
Version 6.3
View Documentation
Hotkeys
Open Ctr + O
Save Ctr + S
Zoom Ctr + Scroll
Run F10
Compile F9
Clear Ctr + W





Select a path to your projects folder. It is best to create separate folders for the language as well as each individual project.

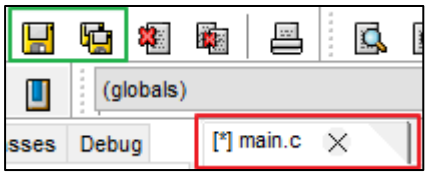
You will now have a console project with a source file called **main.c**



In C language **main.c** is the "Main" application and any other source file added to the project will be a child process of main. **main.c** is always called first by the IDE and compiler. You can only have one source file named **main.c** in a project. This also includes the formal entry point in your source files in which you can only have one "**int main(int argc, char *argv[]){}**" which will exist in **main.c**.

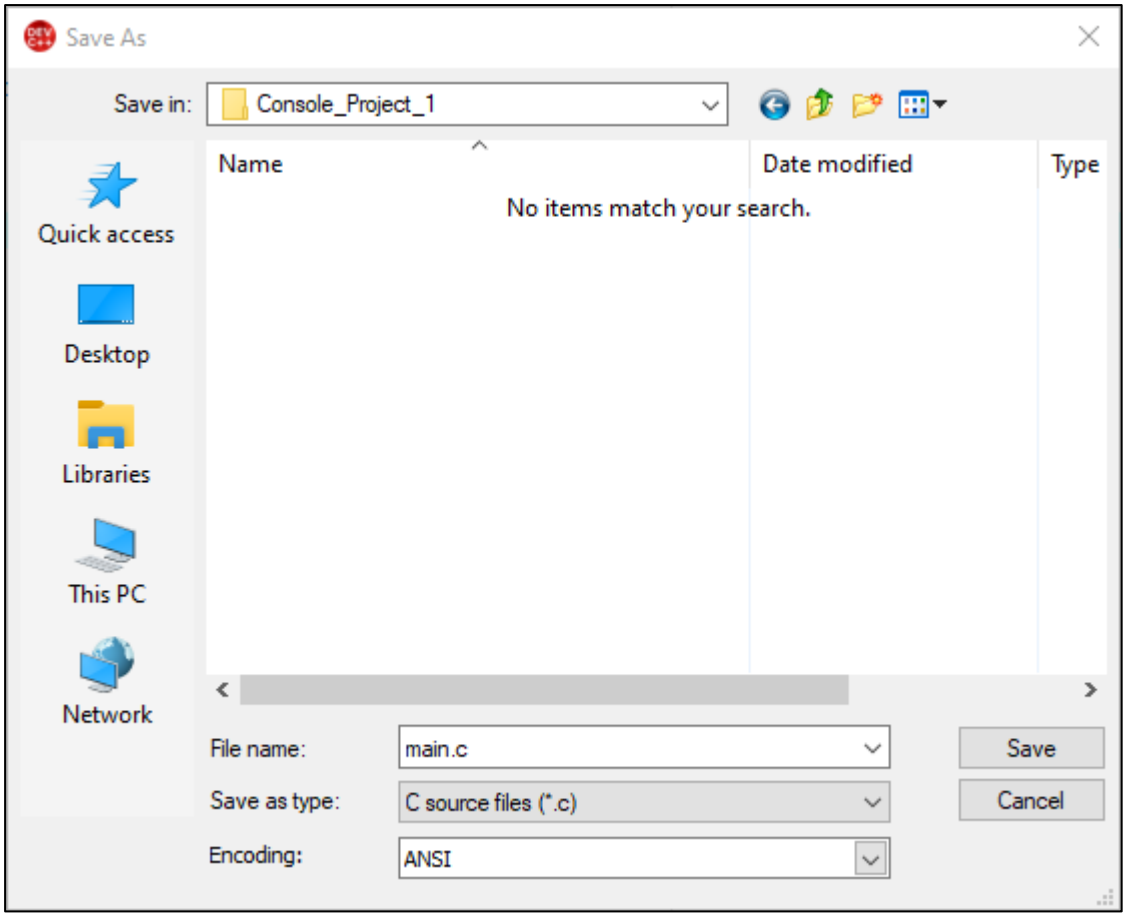
It is possible to compile a single source file using the global compiler setting if it is not included in the projects file list. This can be handy when testing small snippets of code that are not formally part of the project, or testing short pieces of code when learning. I have already set up the global compiler settings for this in our initial setup.

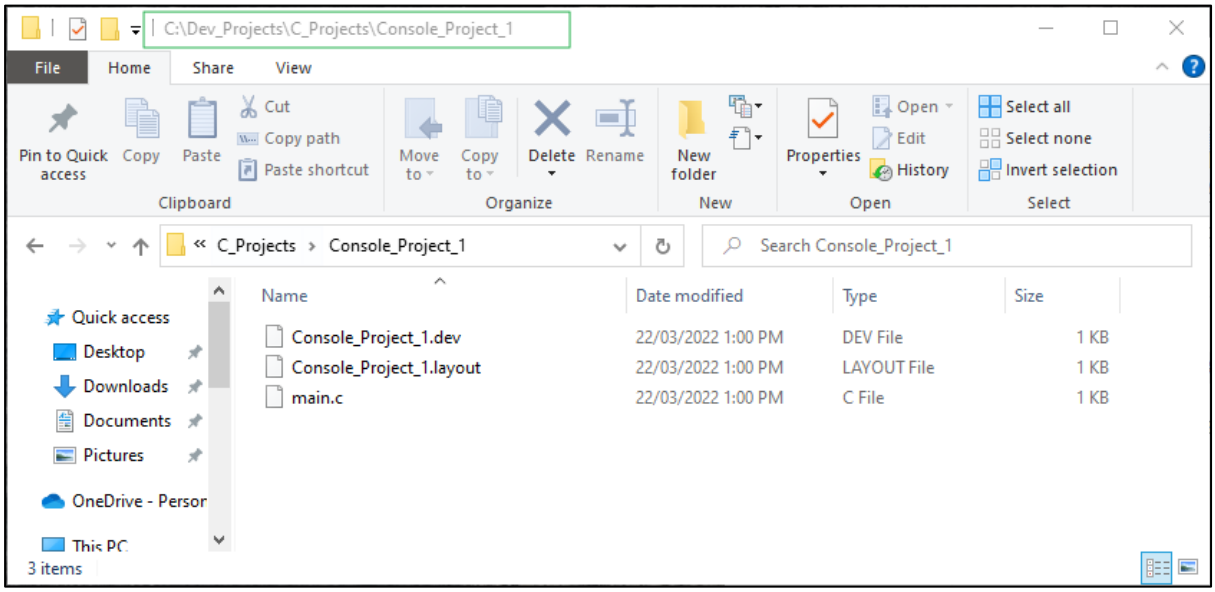
Use the save file or Save as to save the created source document.



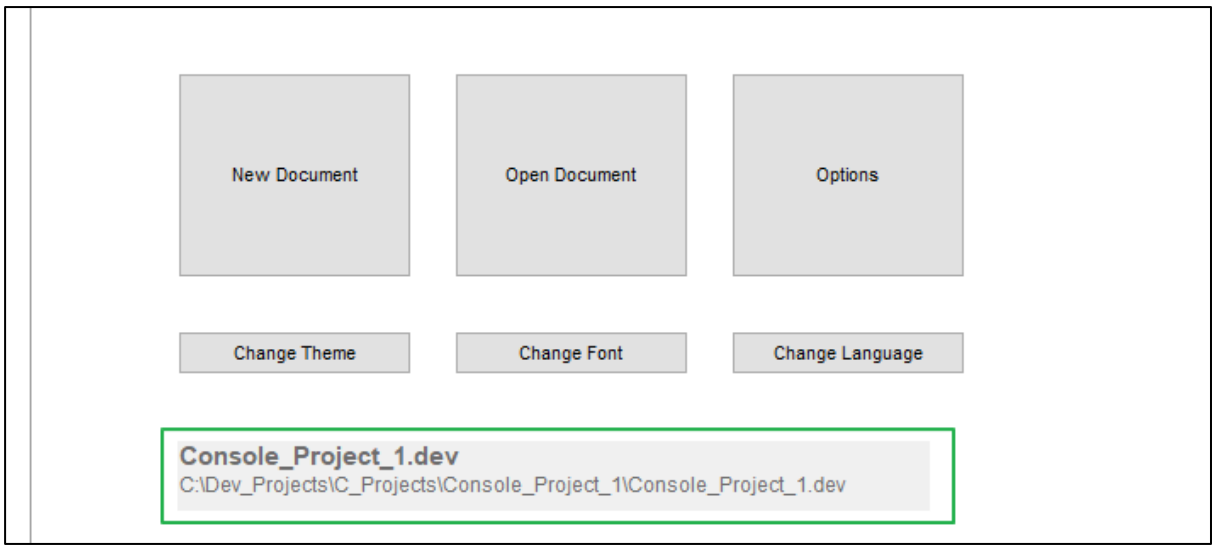
Note the [*] in the source TAB informing you that changes have not yet been saved.

Save the **main.c** to the same folder as the IDE project save.





The next time you start the Dev-C++ IDE you will have the option to open your previous projects.



Alternatively you can use the File Open menu and navigate to the project folder and open Console_Project_1.dev

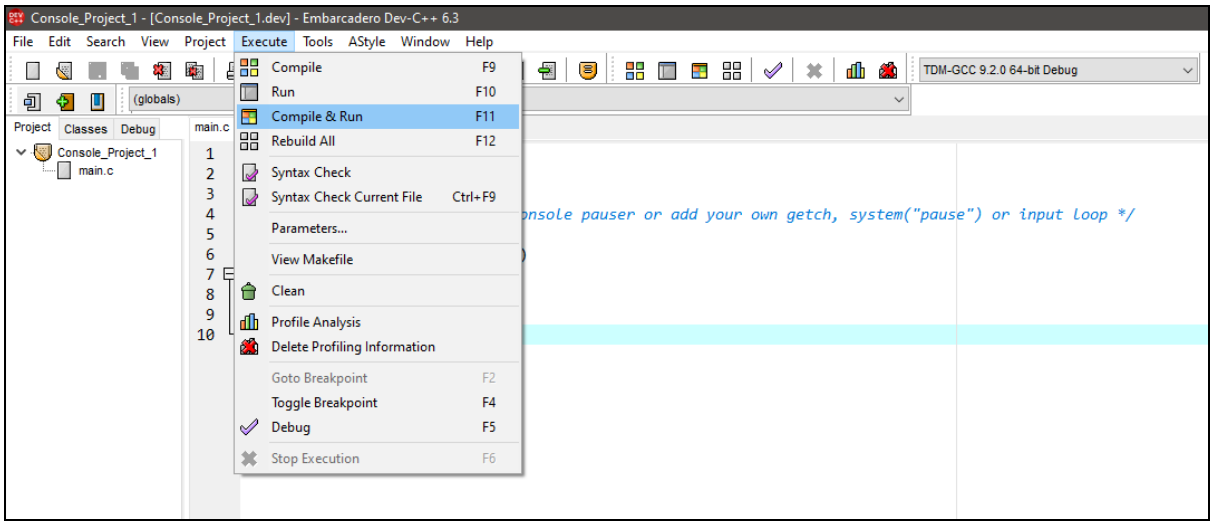
Set Project Options

With your first project open and **main.c** displayed in the editor add the line **`printf("Hello world!");`** and from the menu select AStyle – Format Current File, then save file.

The following is using whitesmith indentation style.

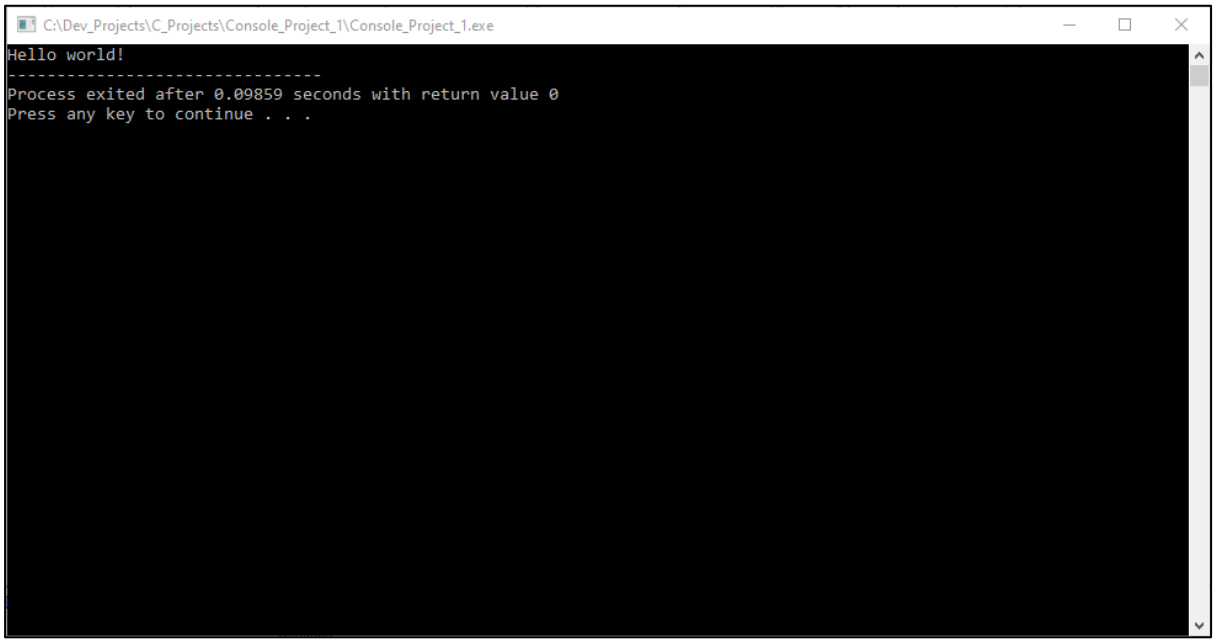
```
int main(int argc, char *argv[])
    {
    printf("Hello world!");
    return 0;
    }
```

Next Compile & Run the application.



You should see the console output showing the Make file script progress in the Compile Log as well as any Errors or Warnings in the Compiler TAB.

Next a console window will appear displaying Hello world!



Press [Enter] to close the console window and return to the IDE.

Don't be concerned about the "[Warning] unused parameter 'argc' [-Wunused-parameter]" as it just means that the variable 'argc' has not yet been used in the source project. 'argc and argv[] are there to accept console command line arguments sent to your application when it is first run.

You have now compiled your first application.

Next select from the Menu bar Project -> Project Options.

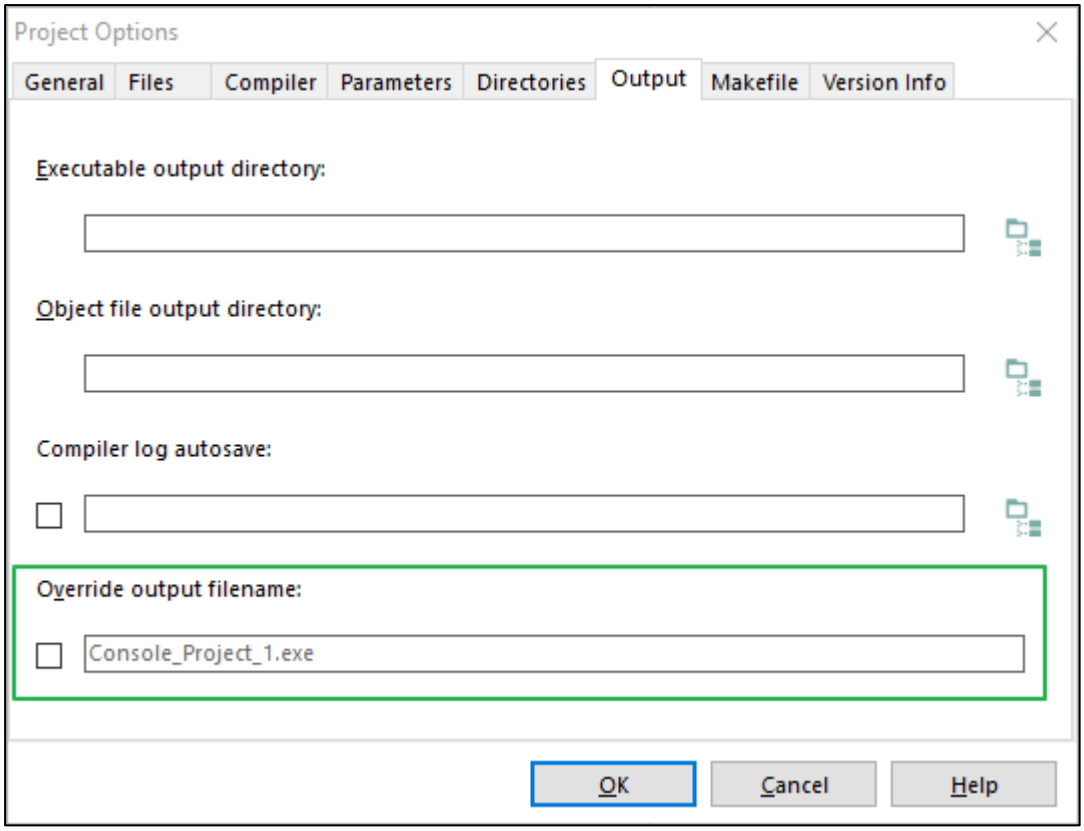
Have a browse through the TABs and you should see many of the same global settings already migrated to your project settings. Don't alter any of the settings as they are already set correctly for learning projects. You can alter the settings for individual projects as you become more confident in your programming ability.

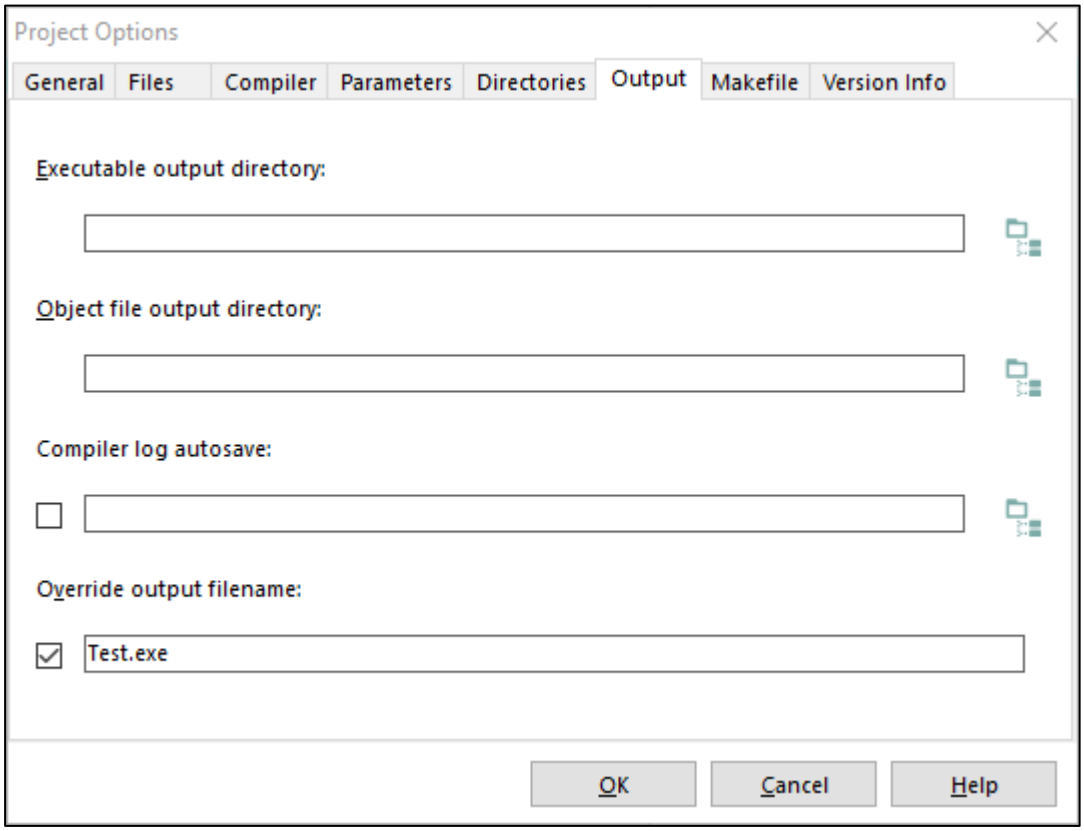
Project Options offers another 2 TABS that are not in the global setting; Output and Version Info.

Under Output TAB you can change the name of the final executable to something more desirable if you wish. The output exe will default to the project name.

If you select override Output filename

[/] Place your own exe name here. You can then provide your own exe name.





I had some issue changing the exe name… Select the check box [/] then close the Project Options window with [OK] then reopen the window and you should be able to alter the exe name.

From here you can add your own code to the **main.c** and test it. Or try some of the examples from "A Beginners Guide To Programming". You can also back up your main.c in a subfolder with a unique name main1.c and create a new main.c file. You can also use any name for your main source file as long as only one file in your project contains "int main(int argc, char *argv[]){}".

Helloworld.c

```
// Helloworld.c

#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
    {
    printf("Hello world!");
    printf("Press the [Enter] key to continue...");
    getchar();  // Pause the program until a key is pressed
    return 0;
    }
```

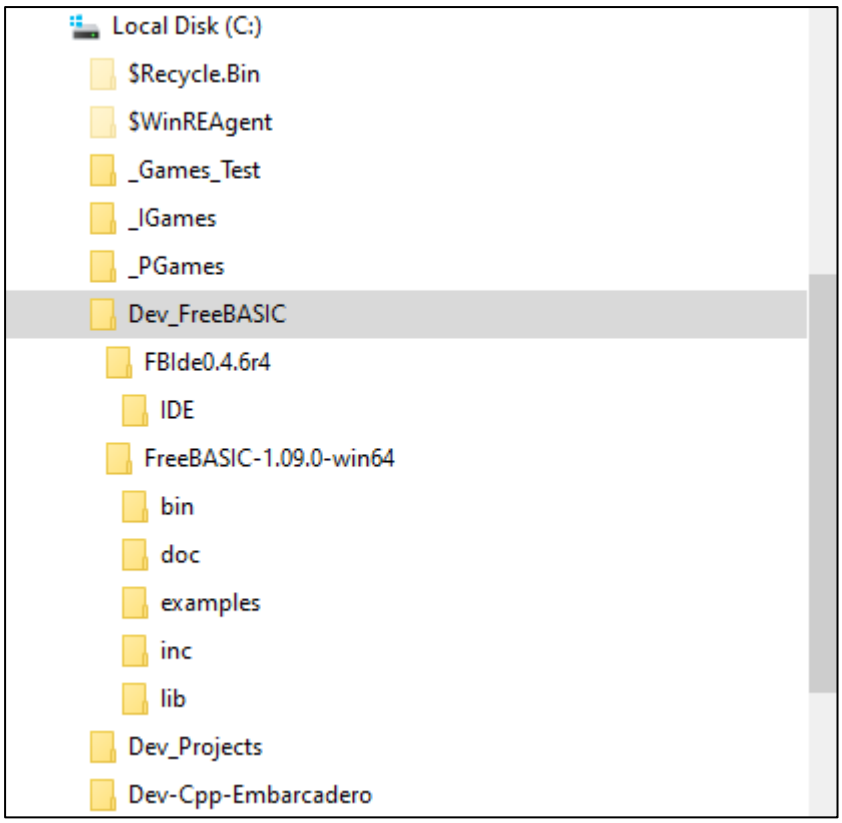
The use of "main.c" is the common convention in C languages for naming the Main source file in large projects.

Dev-C++ also allows you to create a source file of any name and compile it in place with the file name for the exe as long as it is not included in the project files. For example a source file "example1.c" compiled outside of a project will output "example1.exe" in the same directory.

FreeBASIC Compiler plus FBIde

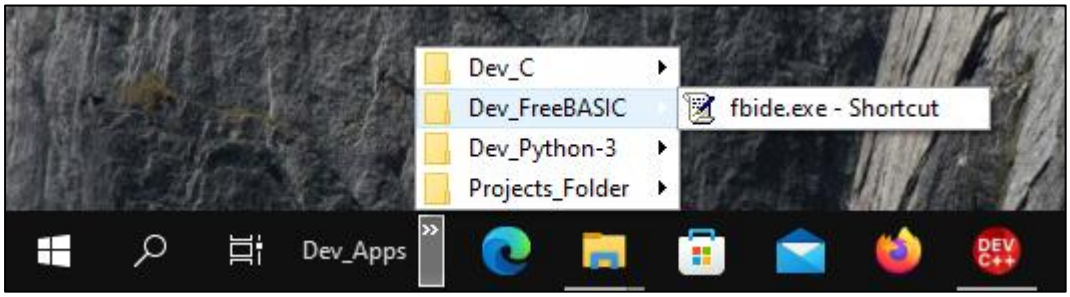
Create a new folder in your OS root drive named "\Dev_FreeBASIC". Unzip the FreeBASIC Compiler package "FreeBASIC-1.09.0-win64.7z" and copy the internal folder "\FreeBASIC-1.09.0-win64" into "Dev_FreeBASIC". You should now have a directory path that looks like "C:\Dev_FreeBASIC\FreeBASIC-1.09.0-win64\fbc.exe"

Unpack the FreeBASIC IDE (FBIde) package "FBIde0.4.6r4.zip" to the "\Dev_FreeBASIC" folder. You should now have a directory path that looks like this "C:\Dev_FreeBASIC\FBIde0.4.6r4\fbide.exe".

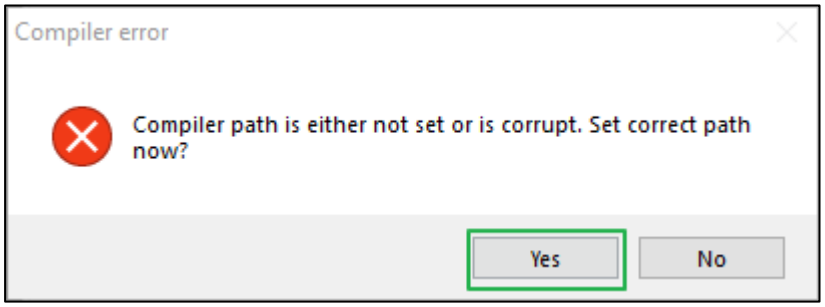


Create a shortcut to fbide.exe on your desktop or place it into a quick launch menu.

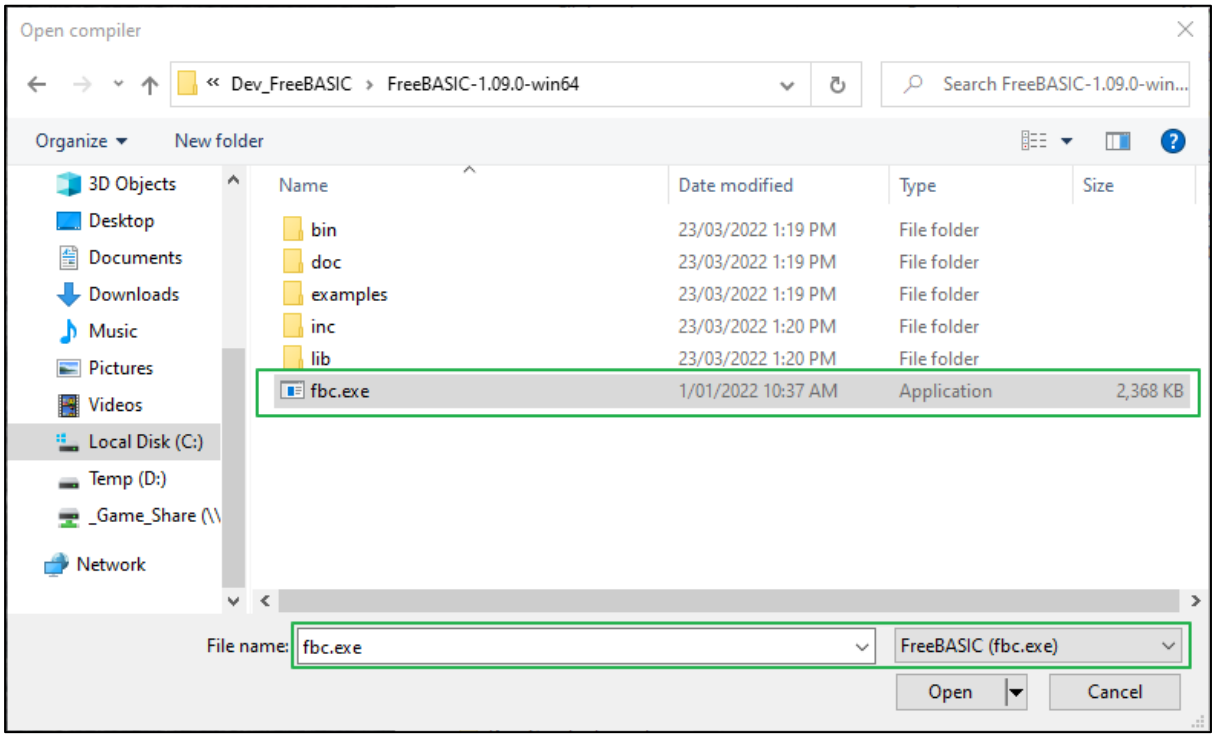
Navigate to the IDE sub folder "`C:\Dev_FreeBASIC\FBIde0.4.6r4\IDE`" and create a shortcut on the desktop to "FB-manual-0.23.0.chm". You can copy this to your quick launch menu alongside the FBIde shortcut if you wish. "FB-manual-0.23.0.chm" is the FreeBASIC programming guide that you will require as a programming reference.



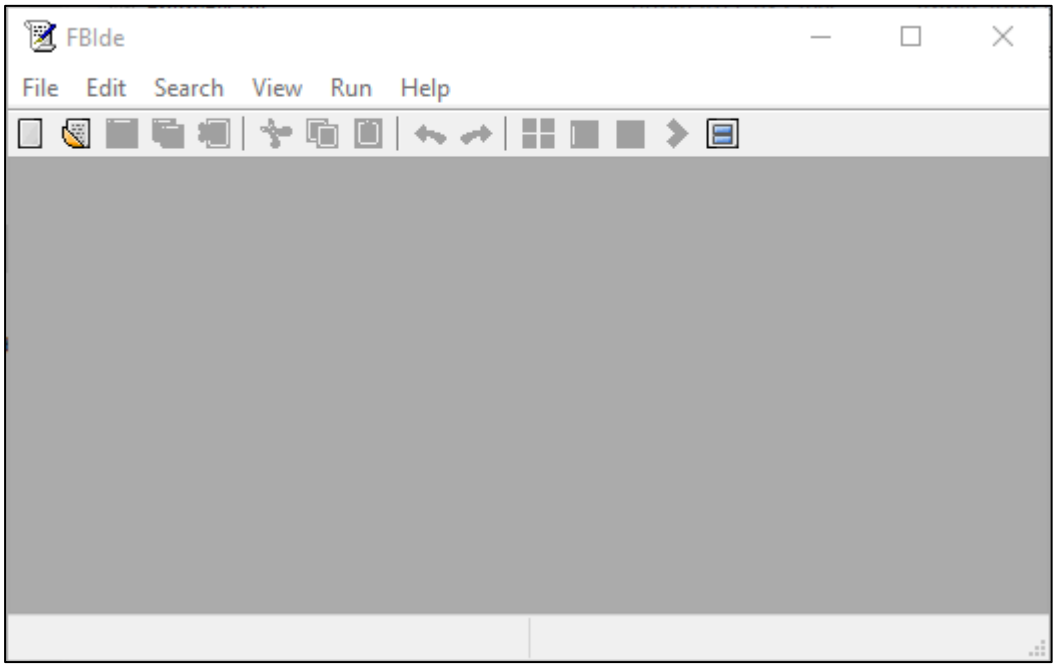
Open the FBIde application. The first dialog you will encounter is a "Compiler error" and an option to set the compiler path "fbc.exe". Select [Yes].



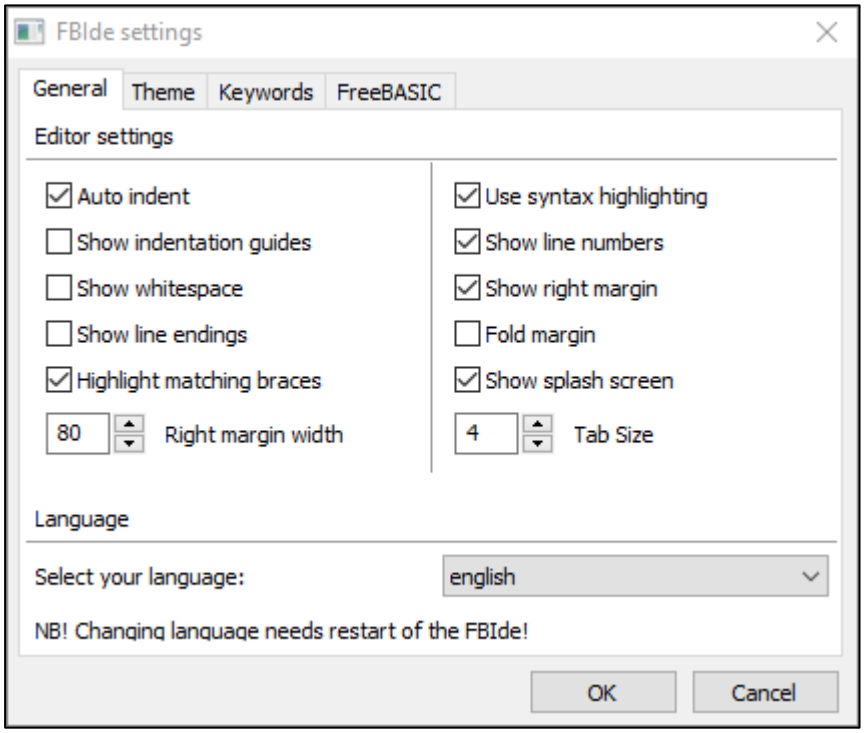
Navigate to "`C:\ Dev_FreeBASIC\FreeBASIC-1.09.0-win64\fbc.exe`" and select "fbc.exe", and then select [Open].

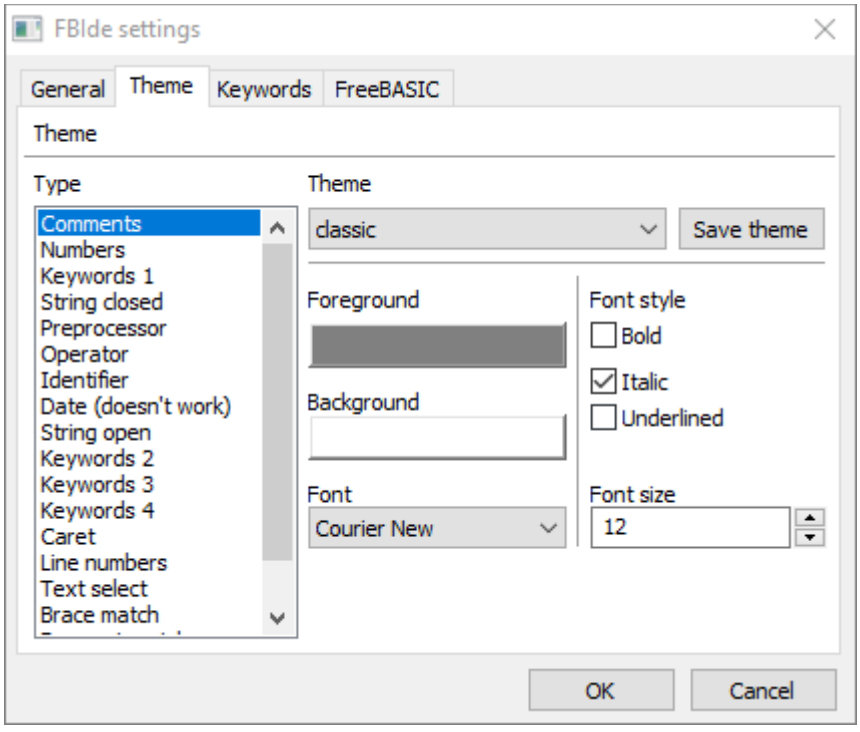


Your compiler path is now correctly set for the FBIde and you will now have the FreeBASIC IDE application open and ready to use.



In the FBIde select View – Settings. The default setting should be fine to work with straight away, but it is always a good idea to become familiar with settings for personalising the IDE as you become more confident.

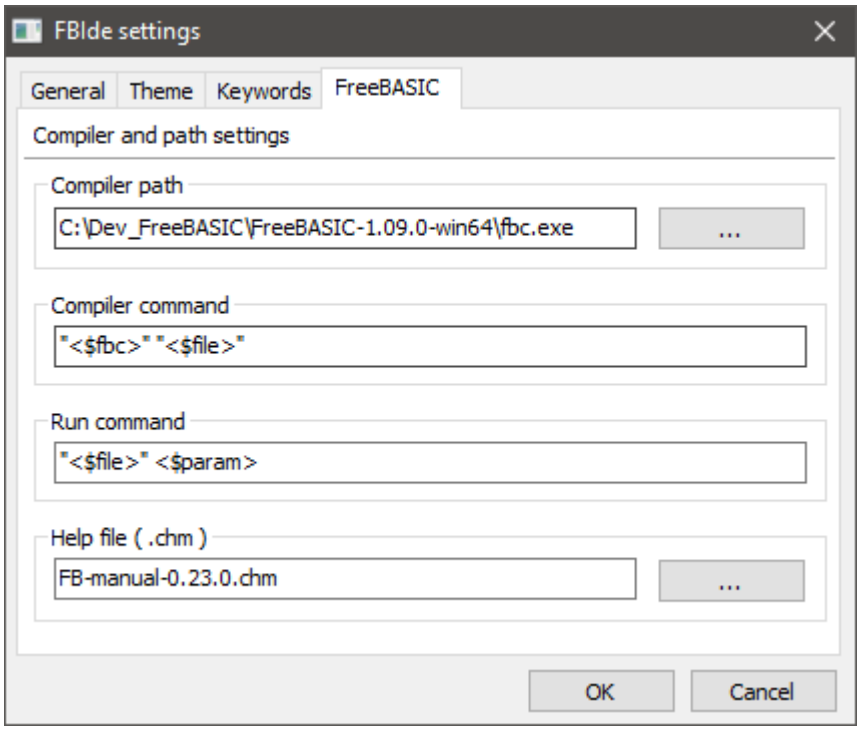




The font size relates to the font size used in the source code. I will most often use "`Consolas`" fixed width font for coding.

The Keywords TAB is a list of commands used for code colour highlighting in the IDE source files.

The FreeBASIC TAB shows the Paths and compiler commands.

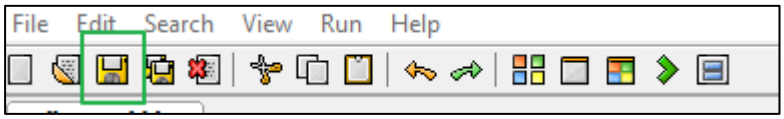


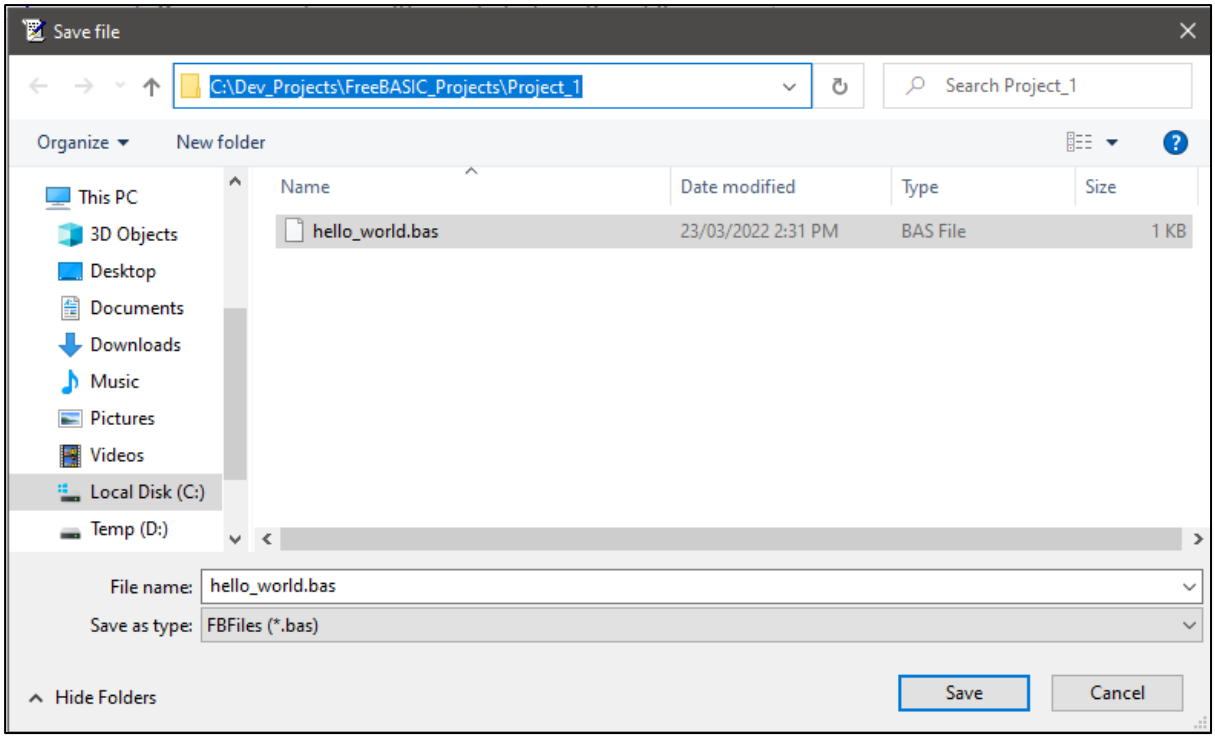
Using the Menu bar select “File” -> “New” and copy the following hello world into the source file.

Helloworld.bas

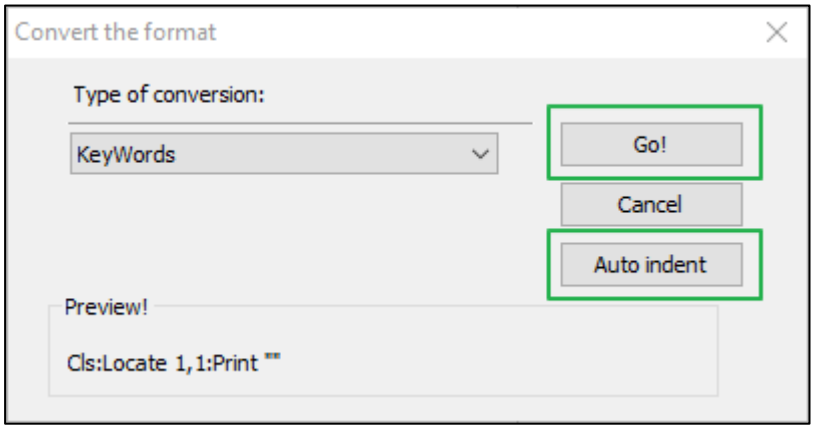
```
' Helloworld.bas

Declare Function main_procedure() As Integer
main_procedure()  ' Call main_procedure

function main_procedure() As Integer  ' Main procedure – “Formal Entry point”
print "Hello world!"
print "Press any key to continue..."
sleep  ' Pause execution so we can view the console output before it closes.
return 0
end Function
```

Select the Save button from the quick menu and create/navigate to a new project folder in “C:\Dev_Projects\FreeBASIC\”.





Next select from the menu bar View – Format.



Select [Go] to correct keyword styles, followed by [Auto indent] to correct code indentation.

Select cancel when finished.

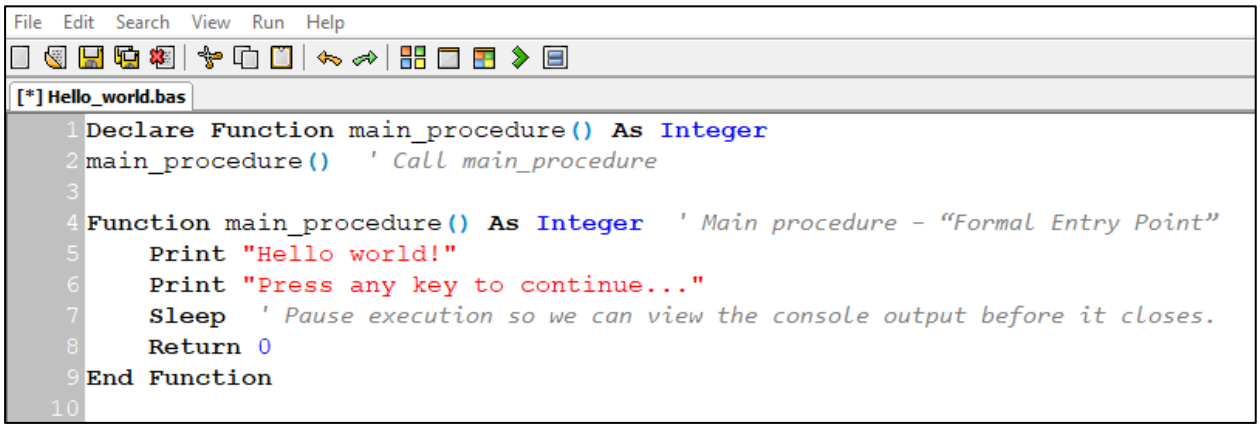
You should now see that your source keywords have been styled beginning with a capital letter and the source indentation has been corrected.

Helloworld.bas

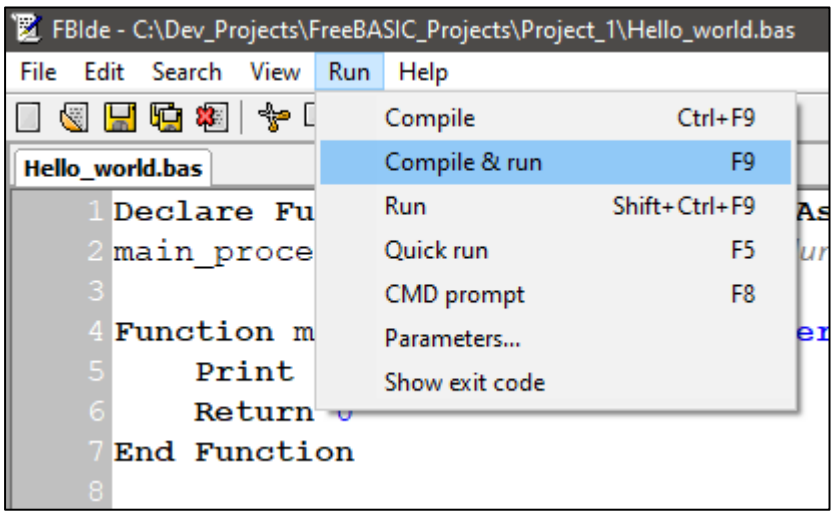
```
' Helloworld.bas

Declare Function main_procedure() As Integer
main_procedure()  ' Call main_procedure

function main_procedure() As Integer  ' Main procedure – “Formal Entry point”
    print "Hello world!"
    print "Press any key to continue..."
    sleep  ' Pause execution so we can view the console output before it closes.
```

```
    return 0
end Function
```

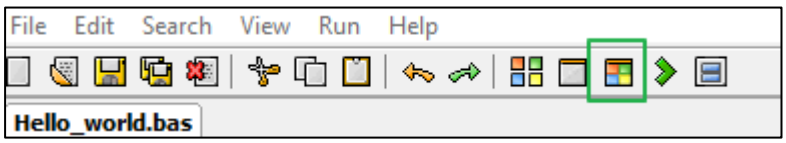
Any time you make a change to your code which has not yet been saved an asterisk [*] will be displayed next to the file name in the document TAB. Remember to save your source at regular intervals.



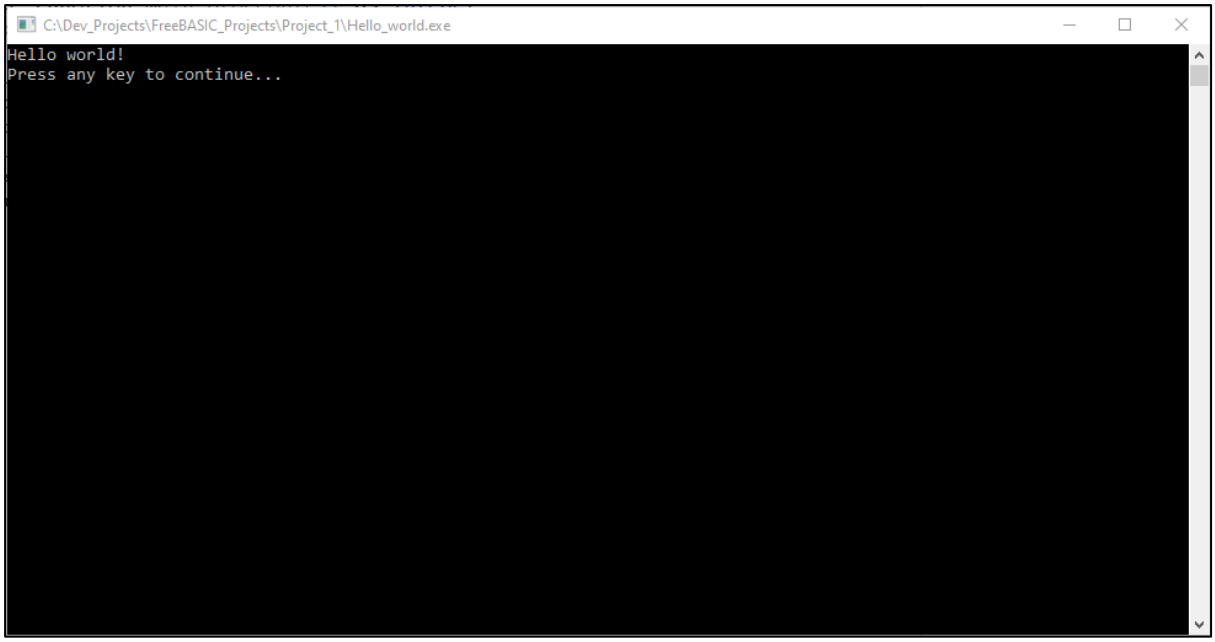
Next from the menu bar select “Run” -> “Compile & Run”



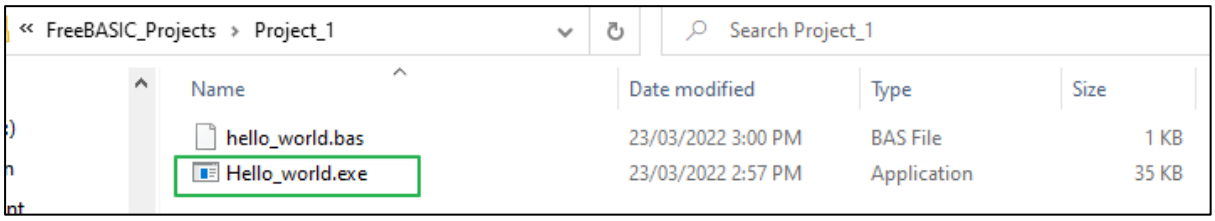
Alternatively you can use the quick menu buttons.



You have now compiled and run your first FreeBASIC application. If all worked well you should have a console output like the following.



If you navigate to your FreeBASIC project folder you will find the output executable file next to your source code.



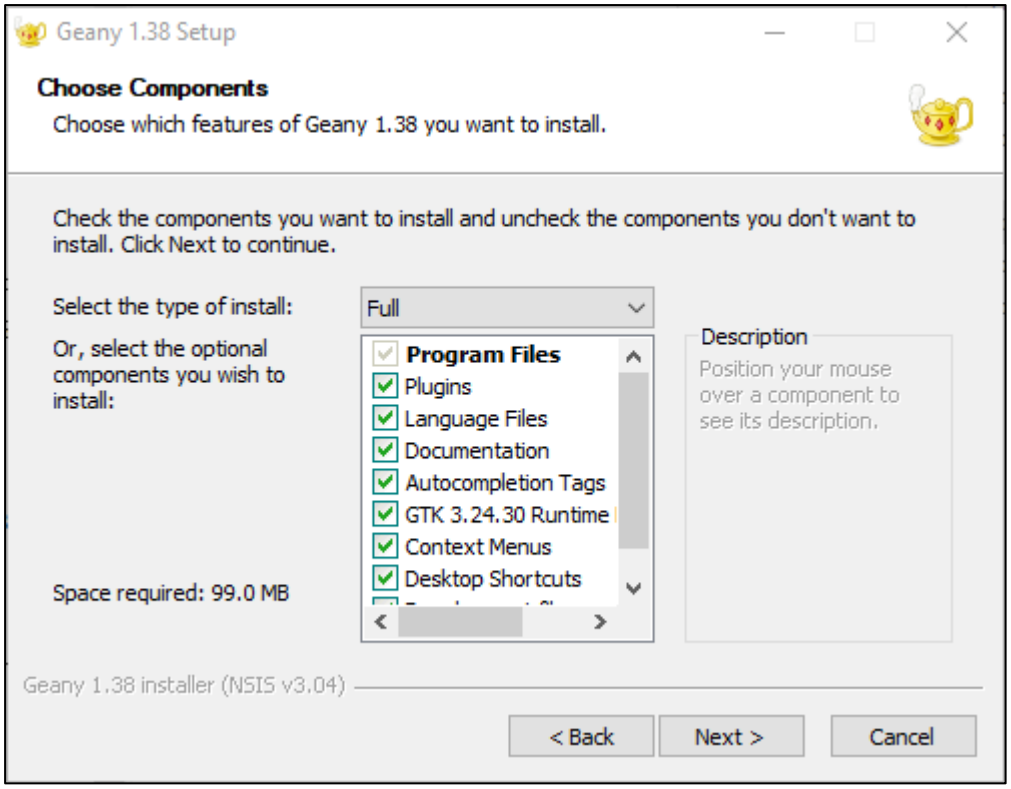
From here you can create your own source examples to experiment with and test, or try some of the examples from “A Beginners Guide To Programming”. I recommend making a backup of your source code “Revisions” from time to time in a separate sub folder. I use Beanland AutoVer 2.2.1 to monitor and create revisions of my source and document projects.

Unlike C languages you can name your FreeBASIC source file with any name that you prefer. I would still suggest a name that relates to the application.

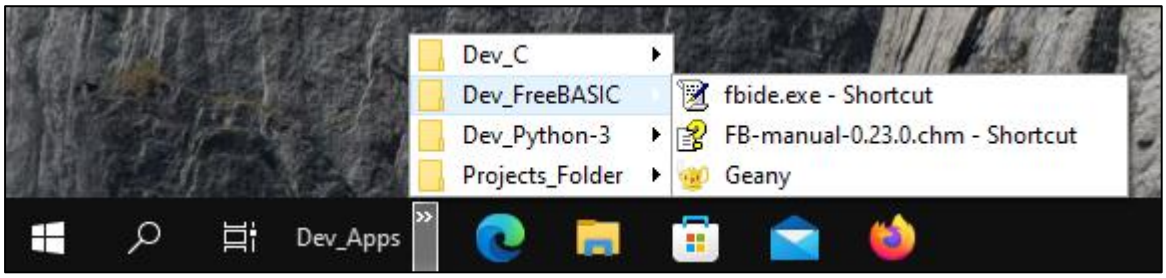
Geany IDE

Geany is an optional IDE that works well with the FreeBASIC compiler. Geany is the default IDE used for FreeBASIC on Ubuntu Linux, so if you plan on learning to code with FreeBASIC on Linux systems it may be less confusing to use the same IDE on both platforms.

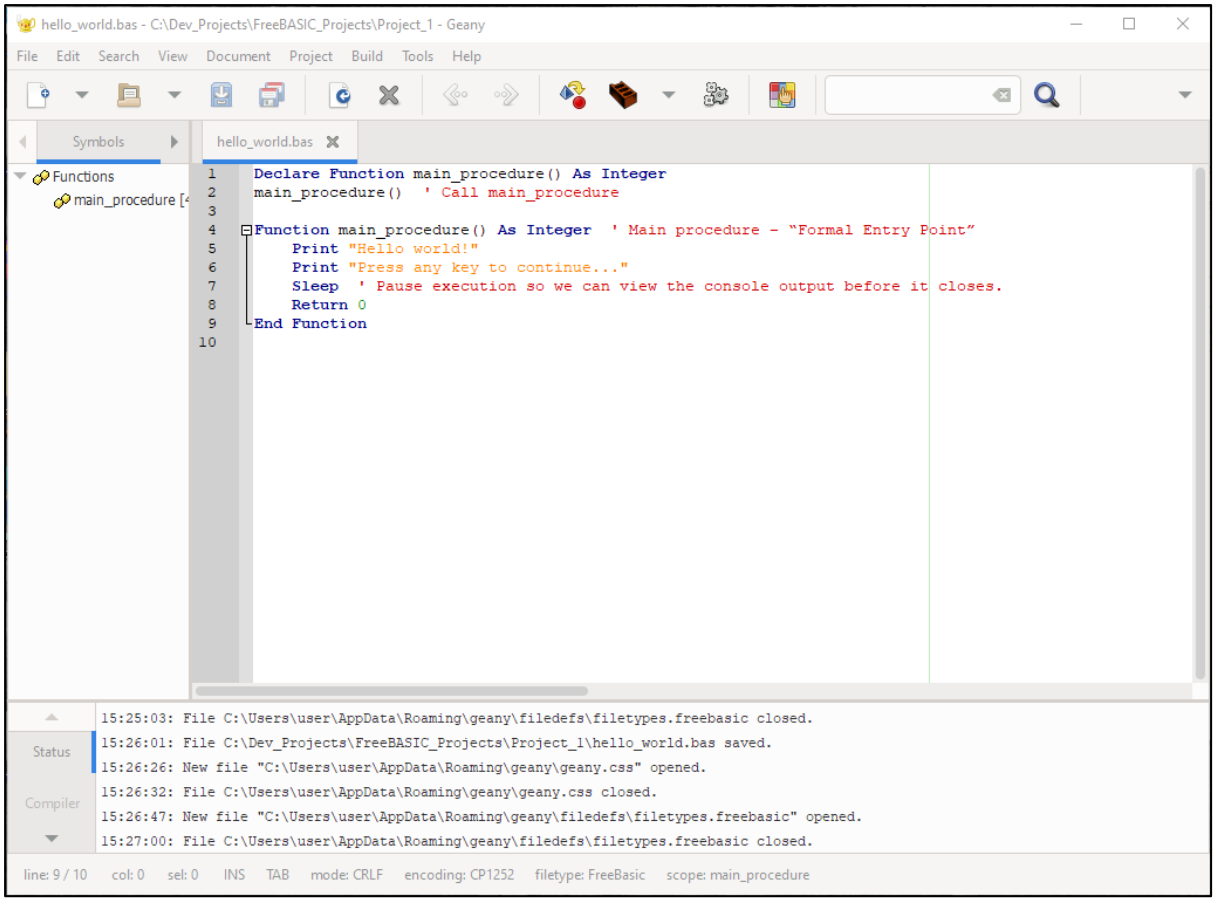
Open the ”geany-1.38_setup.exe” installer and follow the install steps using the default install path and components.



Once the install has completed you will have a shortcut on your desktop to launch the Geany IDE. You can copy this to a Quick launch menu if you find that more convenient.

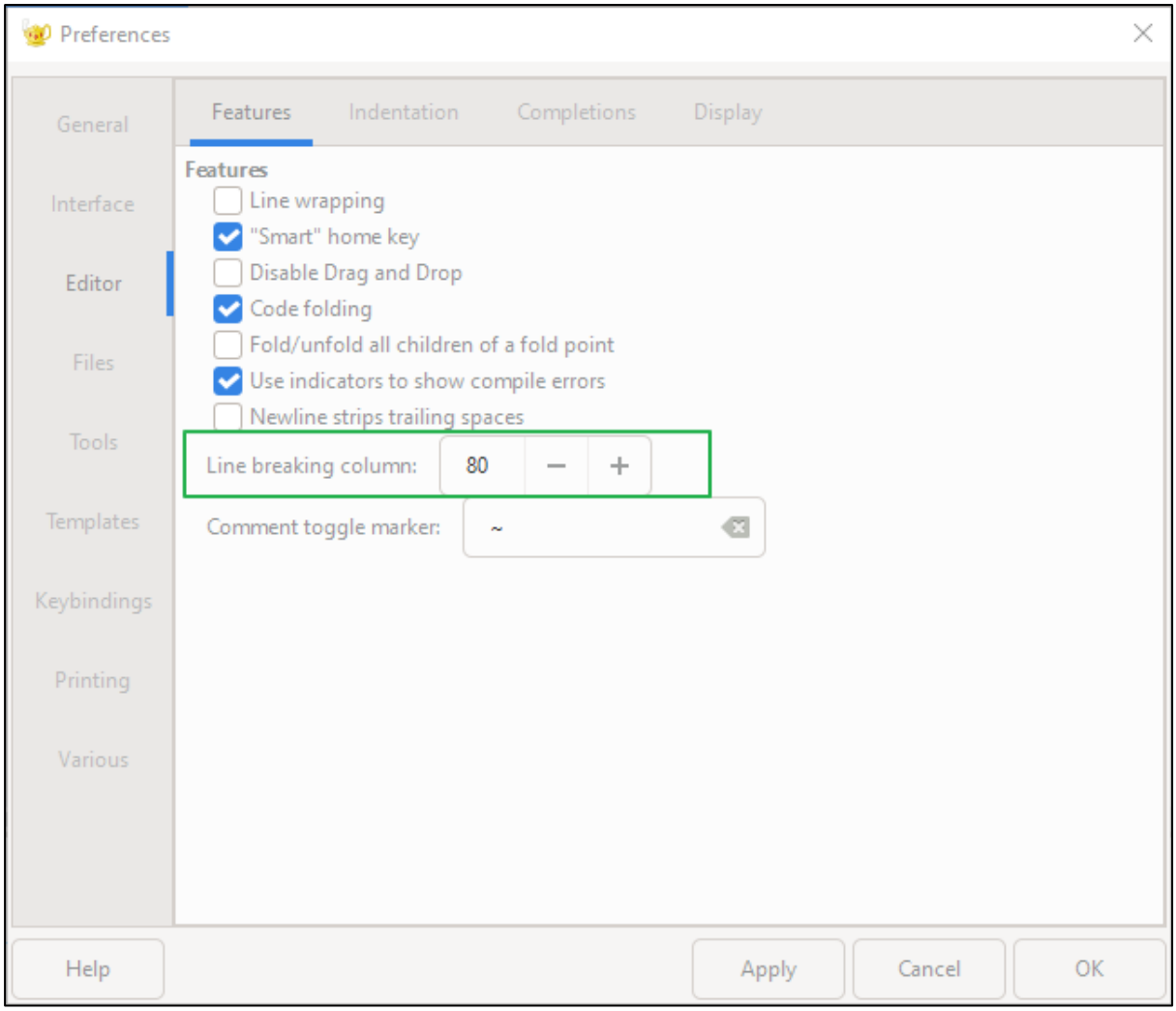


From the Menu bar select File Open and navigate to the “Hello_wold.bas” source file we created with FBIde.

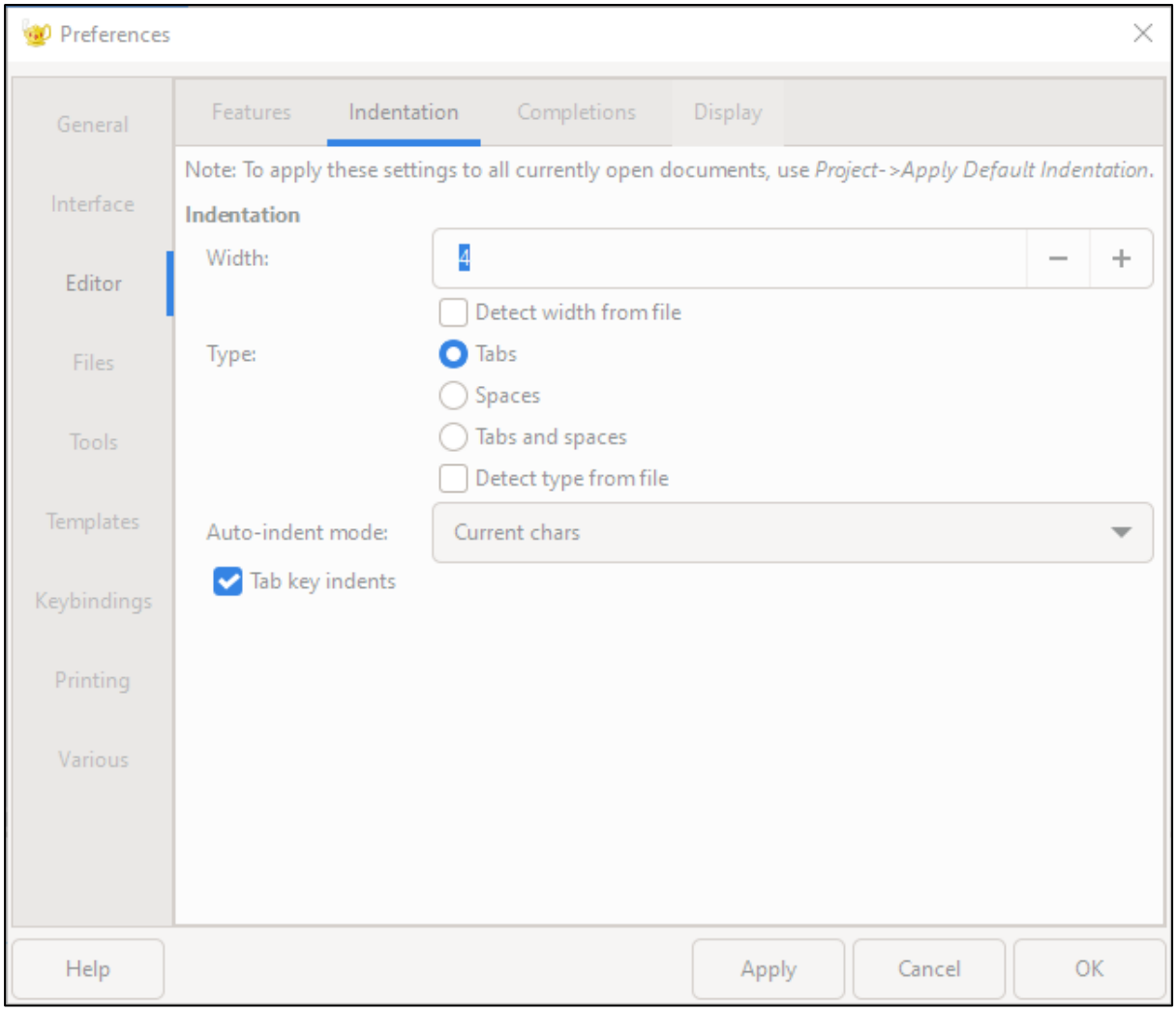


From the Menu bar open Edit – Preferences.

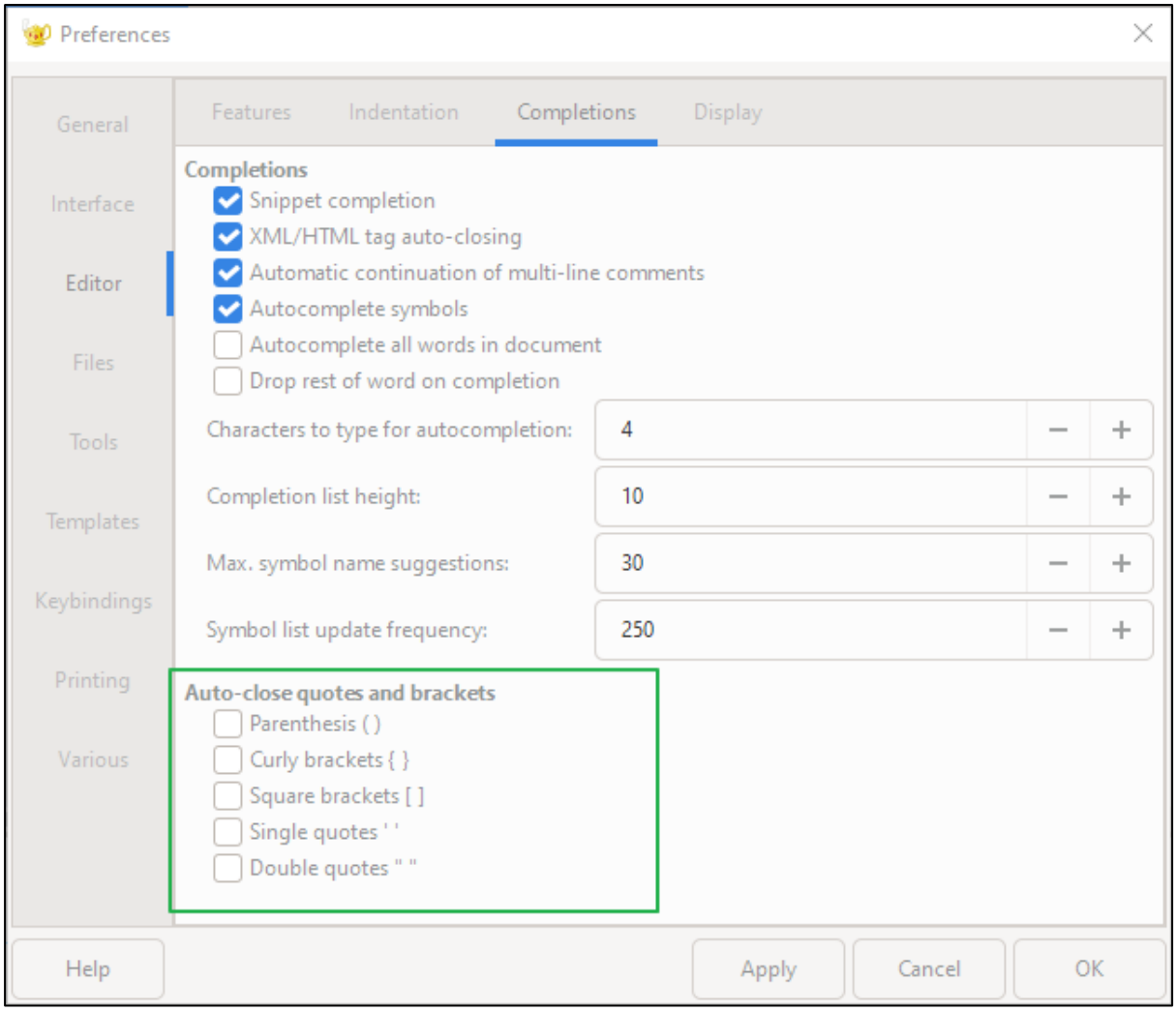
Navigate to the editor TAB on the left and change Line breaking columns to 80 and press [Apply].



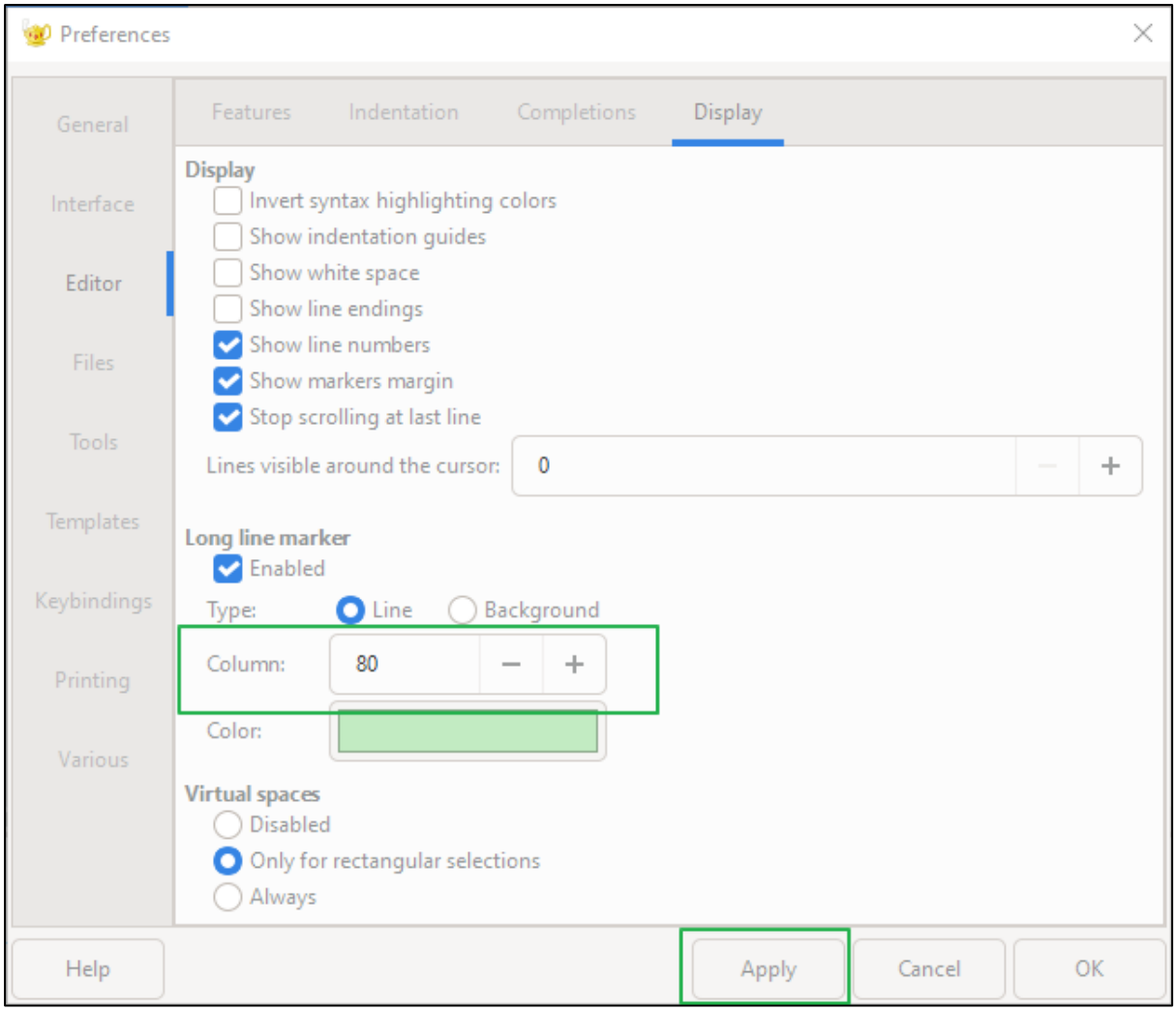
The Auto indentation for FreeBASIC doesn't appear to be working so you may have to manually check indentations.



In the Completions TAB you can alter the auto complete for quotes and brackets if you prefer.



In the Display TAB set the Long line marker to 80 then press [Apply].



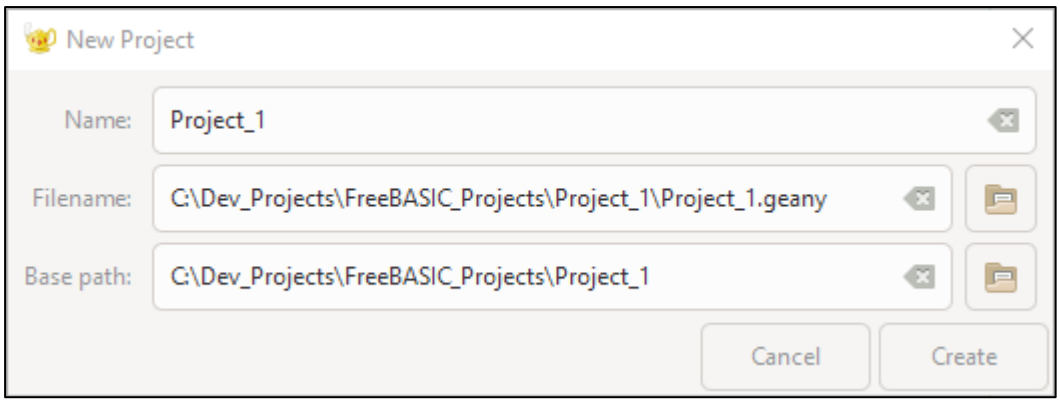
Close the Preferences window.

Geany allows you to create projects, but this is not required to create and test your code while learning. A project helps you to keep custom settings for individual projects that may differ from the IDEs global defaults. If you wish to create a project you can follow the steps below.

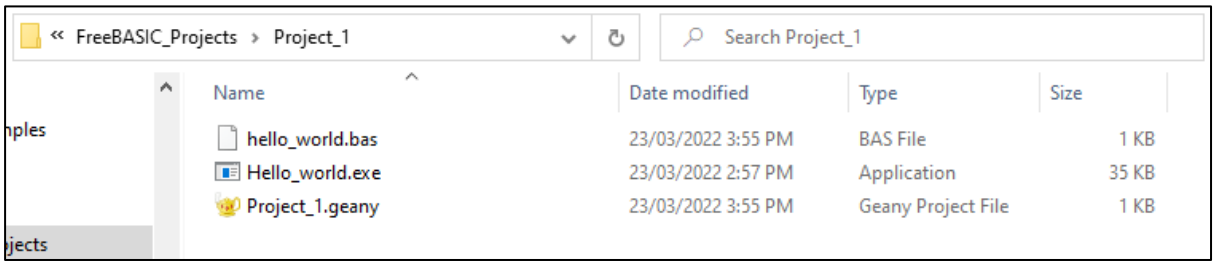
Next form the Menu bar select File – New Project.

Enter a project name as well as the path to your project and Geany project file name.

I have used the same as what I have created in FBIde.



Click on [Create] to create the project.



Next time you open Geany you can also open a project to see your files as well as have your custom setting available.

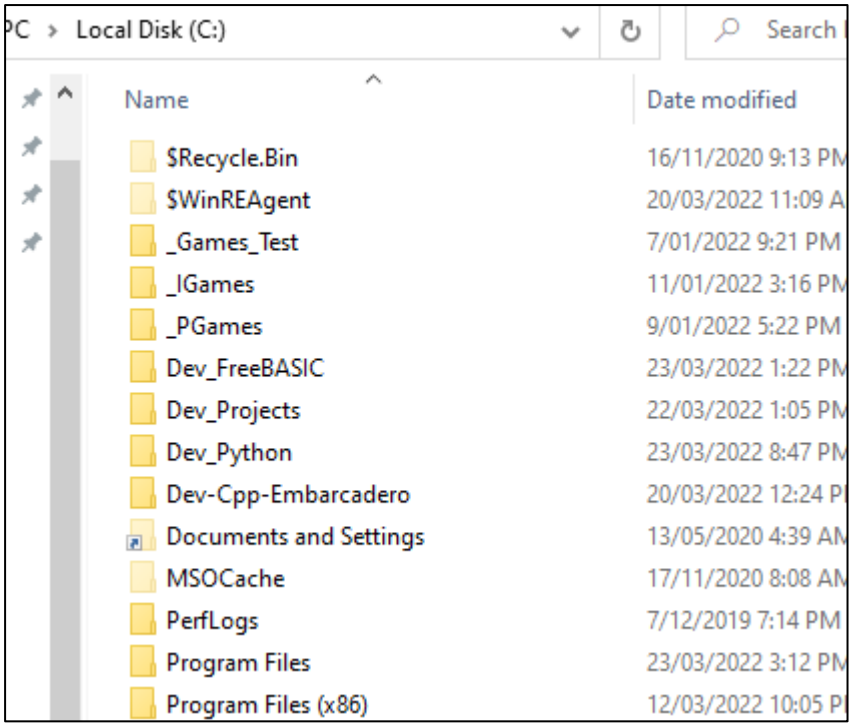
From here you can create your own source examples to experiment with and test, or try some of the examples from “A Beginners Guide To Programming”. I recommend making a backup of your source code “Revisions” from time to time in a separate sub folder. I use Beanland AutoVer 2.2.1 to monitor and create revisions of my source and document projects.

Python-3 and Thonny

If you are planning on using just Thonny as a learning tool then it is not a requirement to install the Python 3 library as Thonny contains a copy of python 3 within the application install. If you do choose to install Python 3 then it is a simple task to change Thonny to use the other Python 3 install.

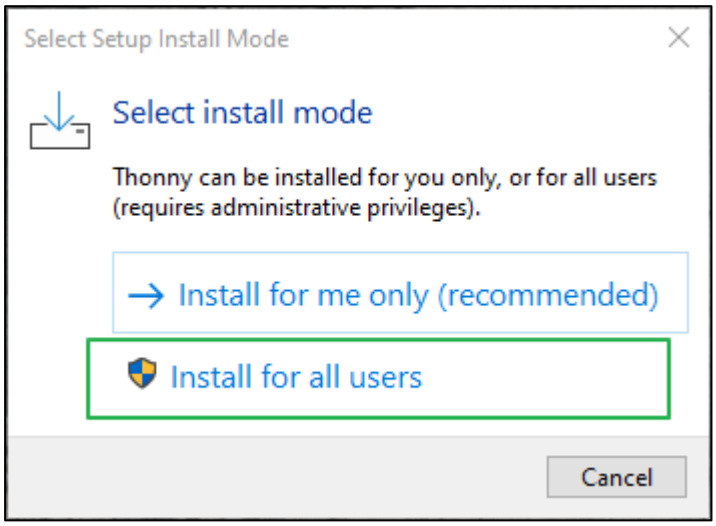
You can only set the system’s environment variables and path for 1 copy of Python, but most IDEs will allow you to use any available library that you choose without needing to have the system environment variables set. This just means that you will have to use the full paths when calling Python3 and your script when working from the command line.

I am going to install Thonny to the “`C:\Dev_Python\`” directory instead of the default program files and user account locations. I find it more practical to keep development tools and libraries under a common path. It may be more practical to create your Python development folder before beginning the installs. Navigate to your OS root drive, usually `C:\` and create a folder named “`\Dev_Python`”. This may look like “`C:\Dev_Python`” when you have created it.

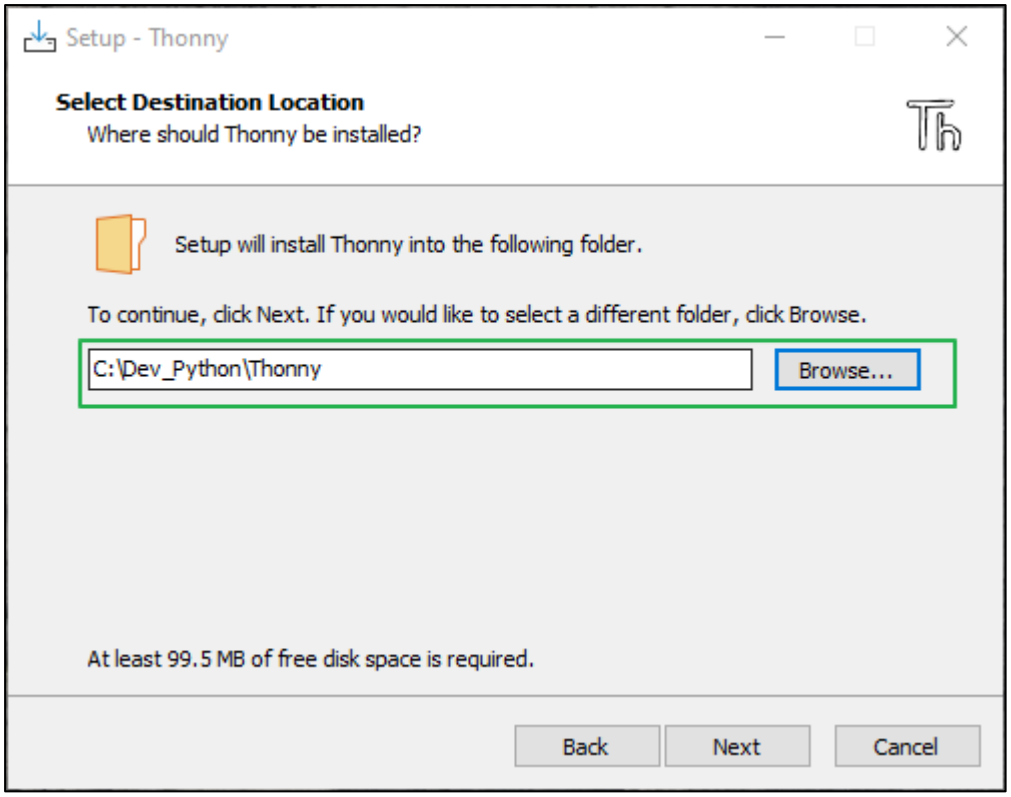


To install Thonny open the “thonny-3.3.13.exe” installer and follow the install steps with the following changes.

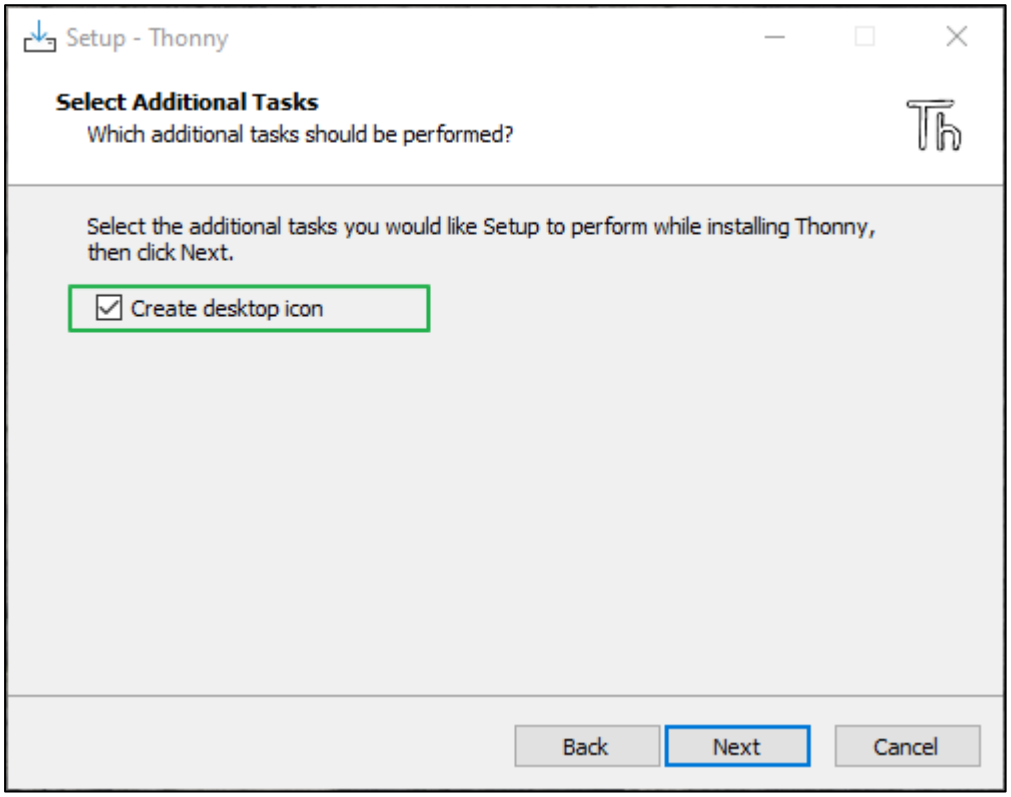
Select “Install for all users”.

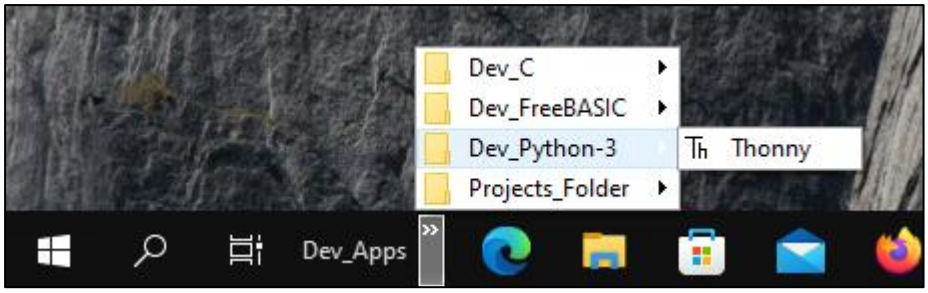


When you reach the Select Destination Location Dialog, change the path to your Dev_Python location. “`C:\Dev_Python\Thonny\*.*`”

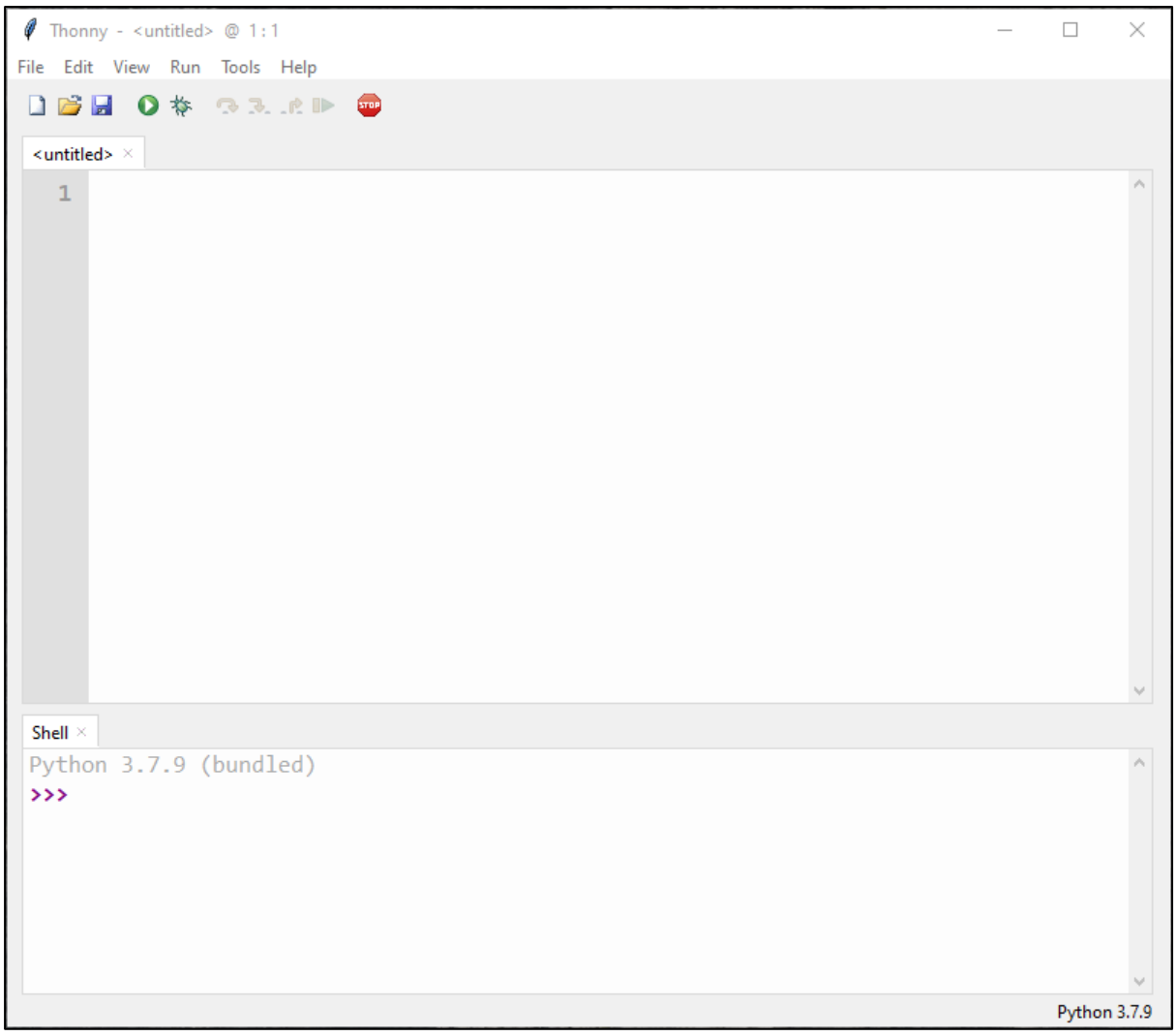


At the additional task screen select “Create desktop icon”. You can copy this to a quick start menu later if you like.



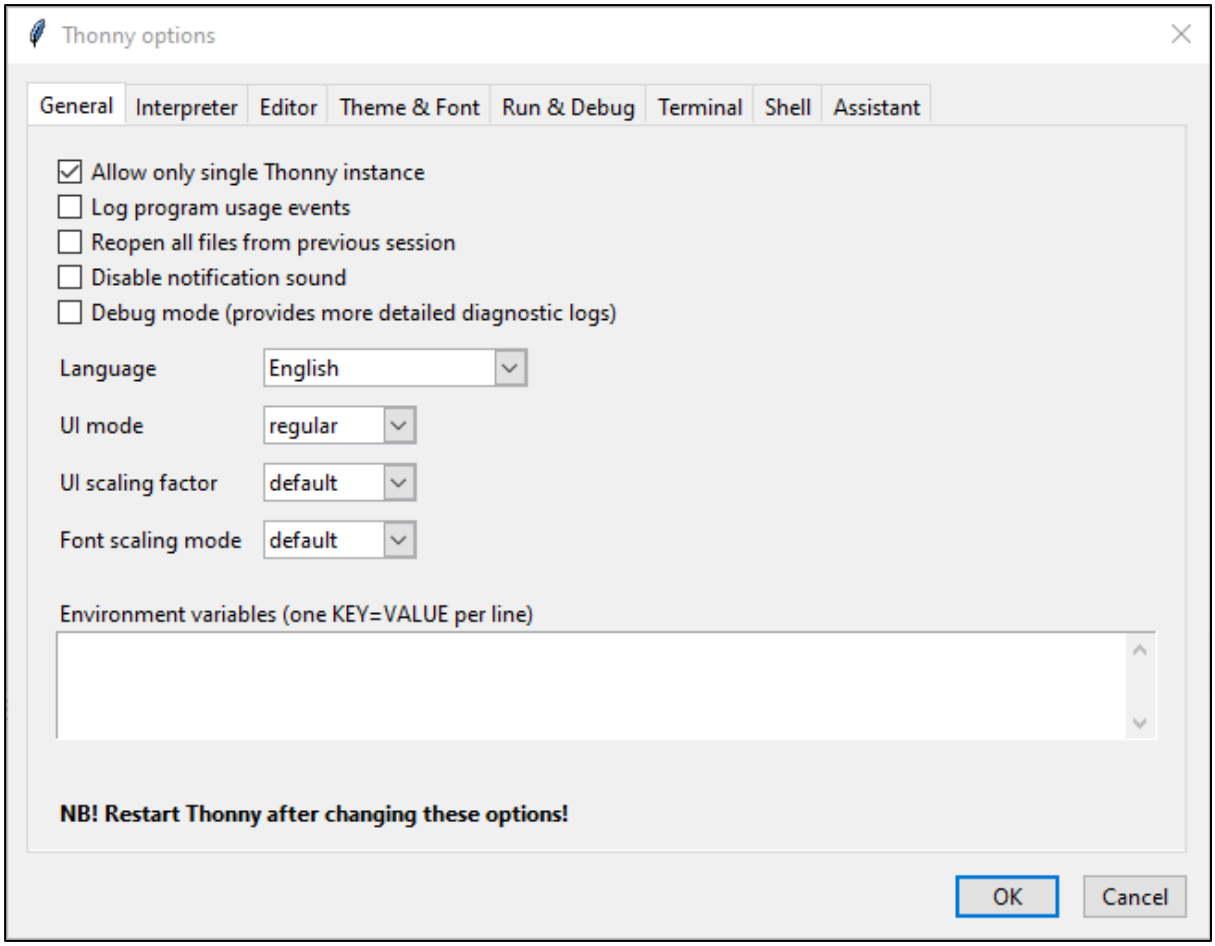


Launch the Thonny IDE application selecting your language and leaving Initial settings to Standard.

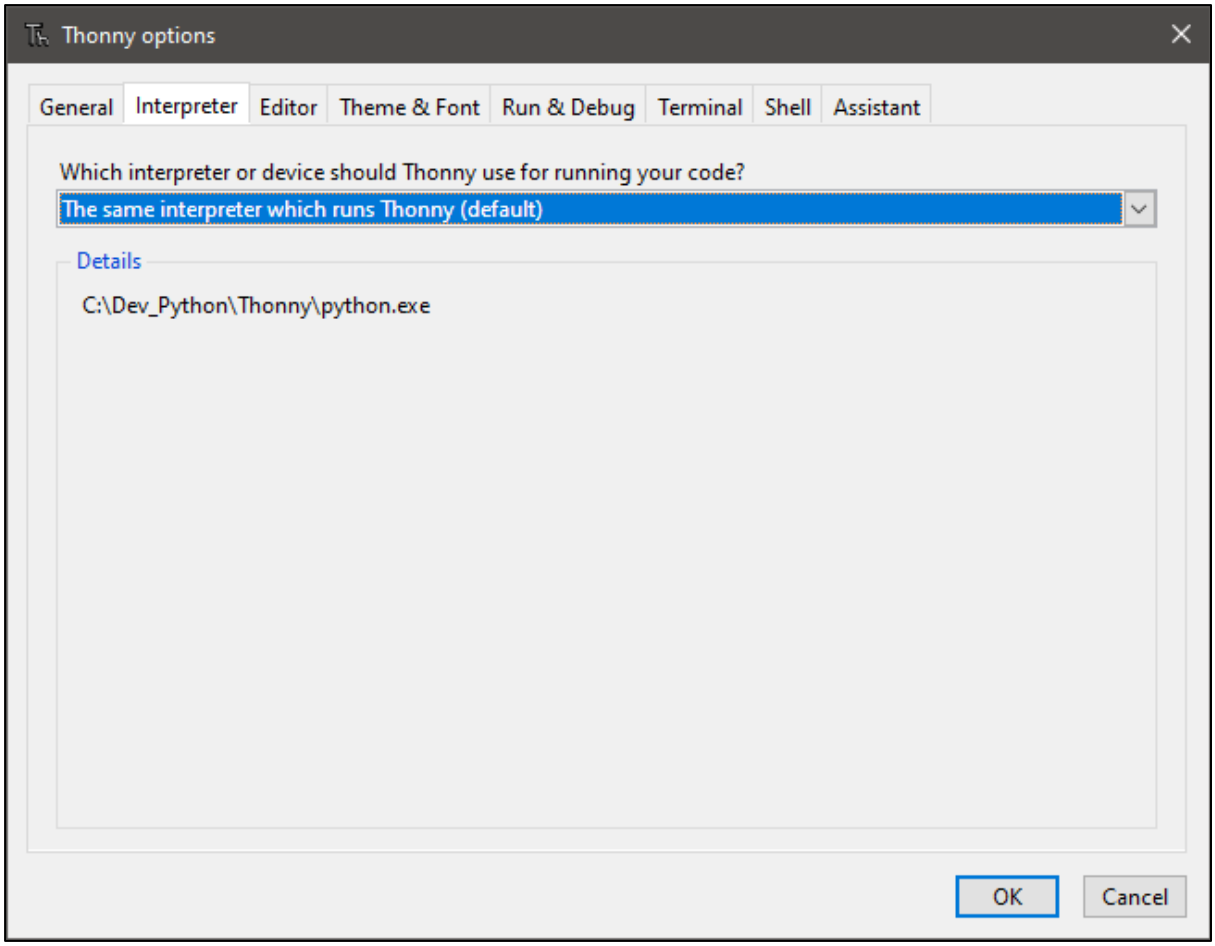


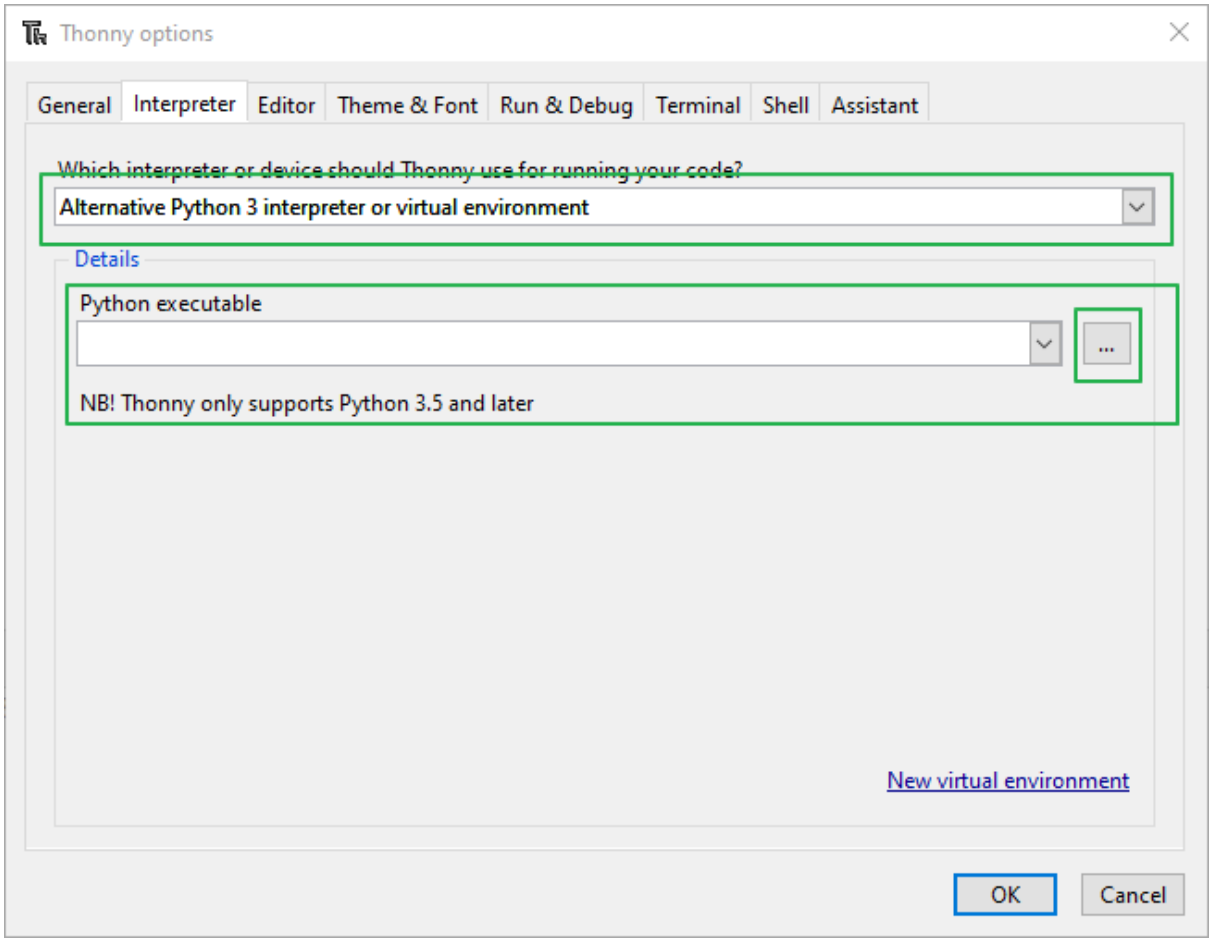
You can begin writing Python-3 scripts and testing them from this point if you wish, but would suggest the following changes to the application Options to make your learning process a little easier.

From the Menu bar select Tools -> Options…



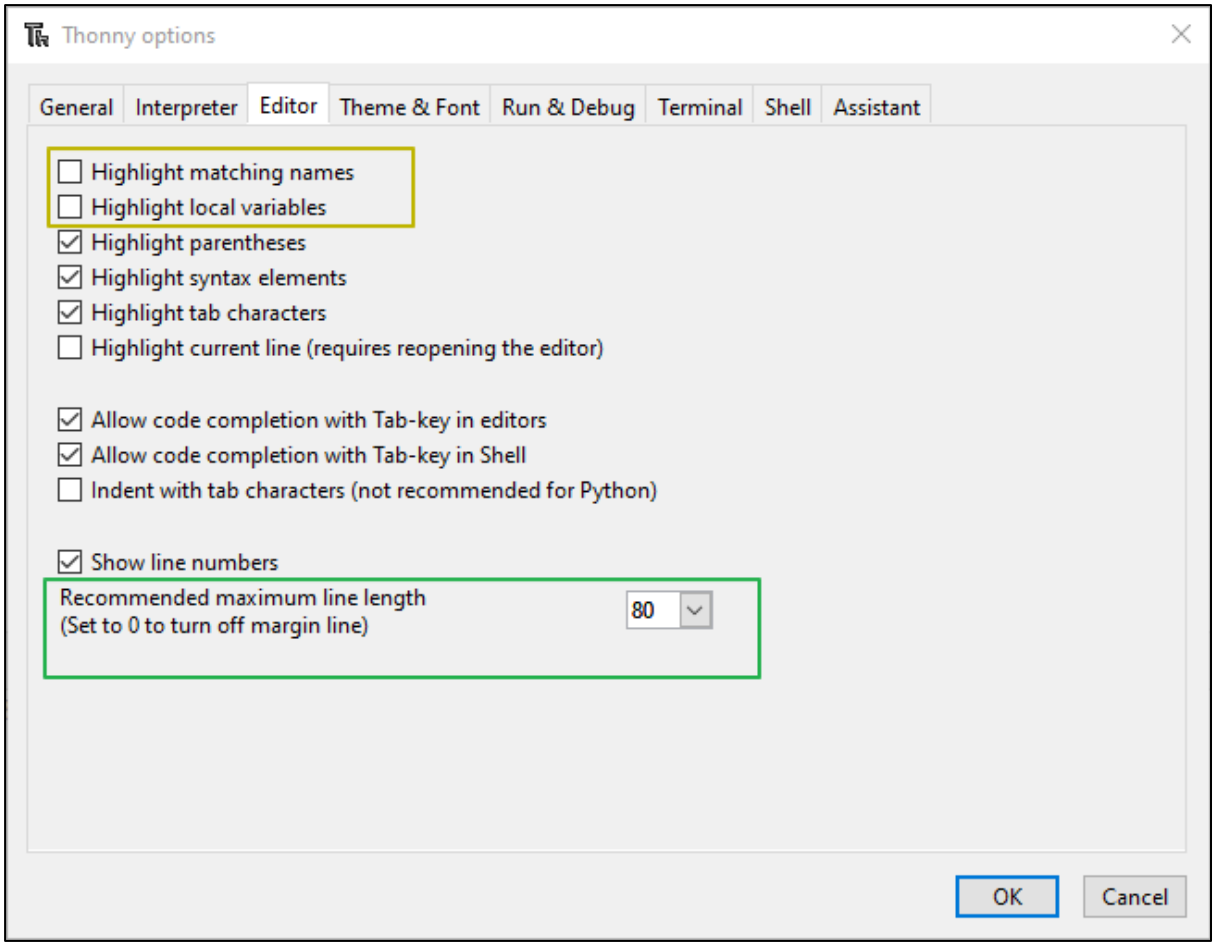
Change to the Interpreter TAB. We won't make any changes here at this time. If in the future you wish to use a separate Python 3 install rather than the Python interpreter built into Thonny, select "Alternative Python 3 interpreter or virtual environment" and enter the path to the Python 3 executable you wish to use. You can come back to this section after installing Python 3.


Thonny options
General
Interpreter
Editor
Theme & Font
Run & Debug
Terminal
Shell
Assistant
Which interpreter or device should Thonny use for running your code?
The same interpreter which runs Thonny (default)
Details
C:\Dev_Python\Thonny\python.exe
OK
Cancel



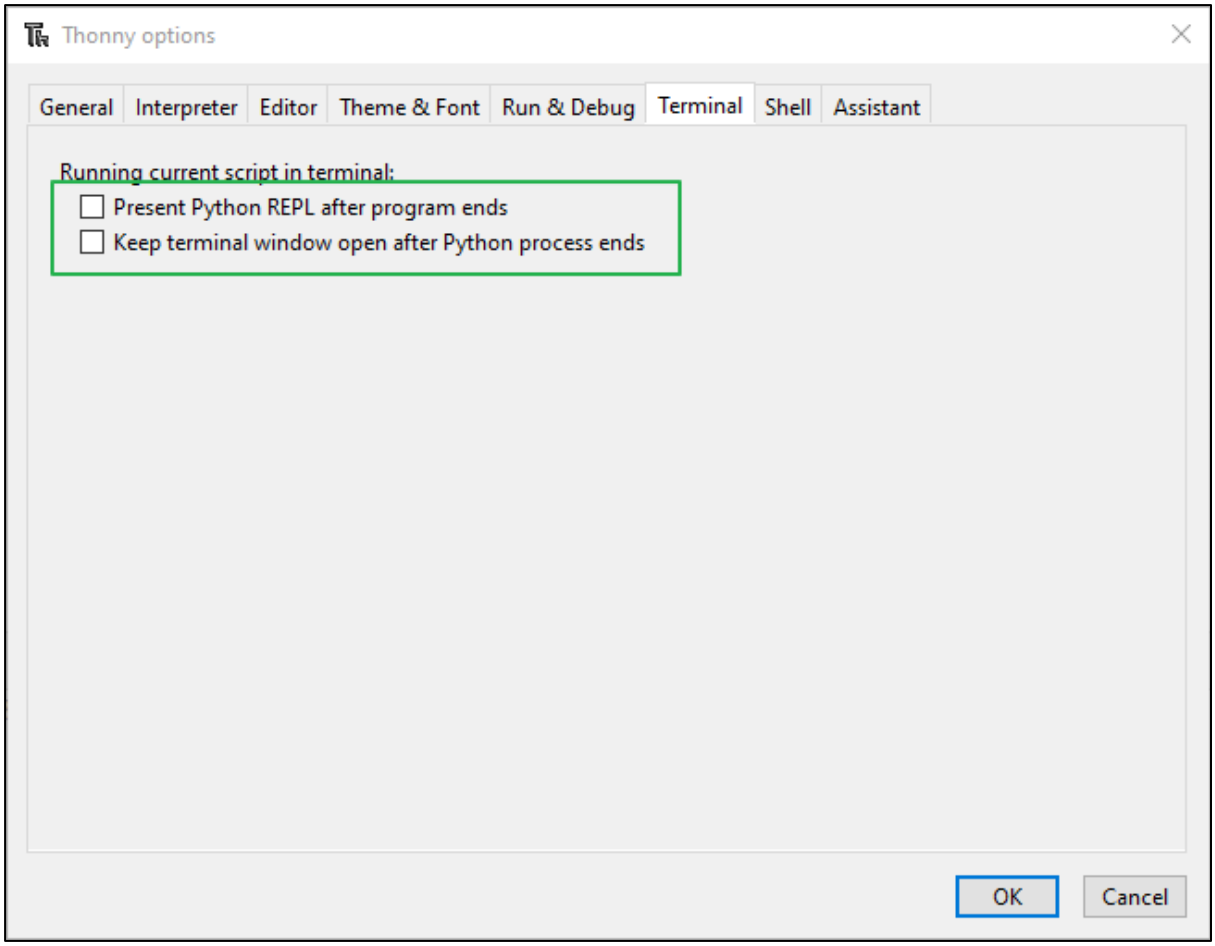
Change to the Editor TAB. The first 2 options can be helpful. I suggest coming back to this TAB once you have some lines of code to work with and try toggling the Highlight matching names and local variables to see which you are most comfortable with.

Set the last item “Recommend maximum line length to 80. This will create a right margin guide to help avoid writing excessively long code lines.



The “Theme and Font” TAB is set to a very usable code highlighting colour for beginners. You can try different code highlighting colours as you become more confident with coding.

Next select the Terminal TAB. Deselect [ ] both options here. If you leave the [ ] Keep terminal window open after the Python process ends you will need to close the terminal(Console) window by Typing quit() then [Enter] every time you run an application in a console window. I find it more convenient to place a “wait for keypress” at the end of my scripts as it only requires a single keypress of the spacebar or enter to exit the console. You can see examples of this in “A Beginners Guide To Programming”. Adding a “wait for keypress” also allows you to test your script directly from the command line while allowing the window to stay open so you can see the results.



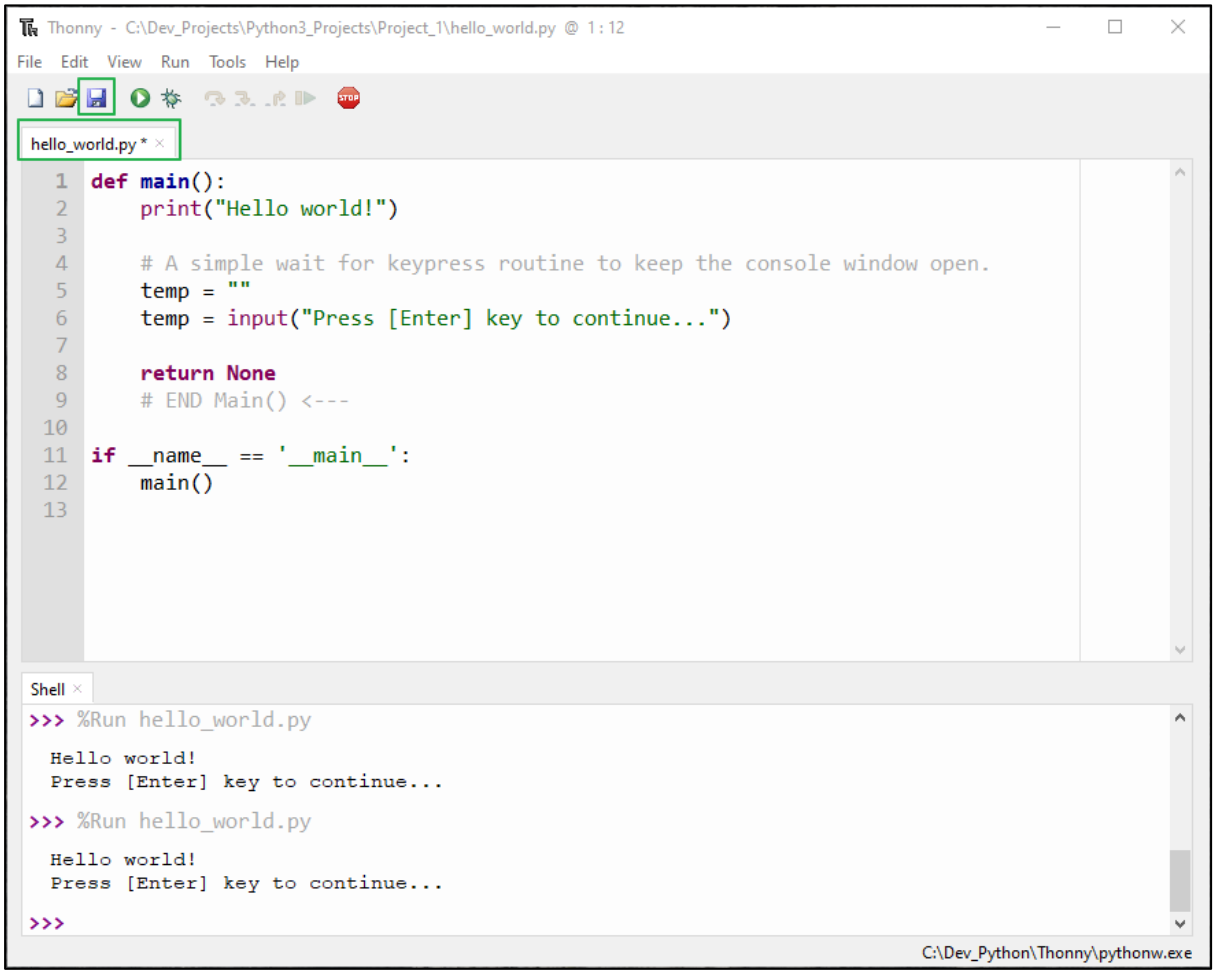
All other settings are fine at the defaults. Press OK to save the changes and exit the Options dialog. You are now ready to start coding.

A new source code document should already be open in Thonny. If not, use File -> New to create a new source document. Enter the following Hello world into the code area.

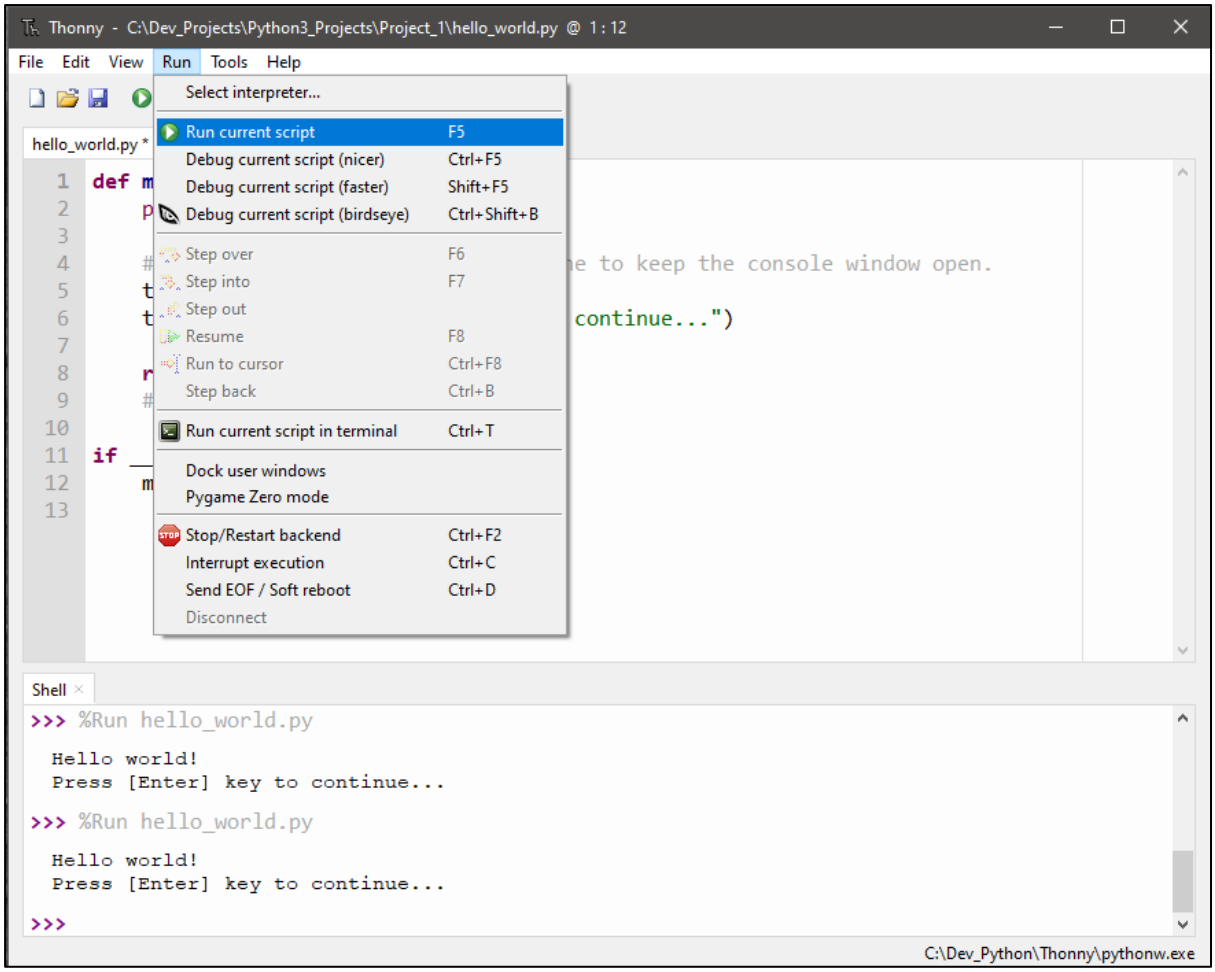
Helloworld.py

```
## Helloworld.py

def main():
    print("Hello world!")

    # A simple wait for keypress routine to keep the console window open.
    temp = ""
    temp = input("Press [Enter] key to continue...")

    return None
    # END Main() <---

if __name__ == '__main__':
    main()
```

Click on save in the tool bar and navigate to a new project folder in the "\Dev_Projects\Python3_Projects" and save as "hello_world.py". An unsaved document will have an asterisks * next to the file name.

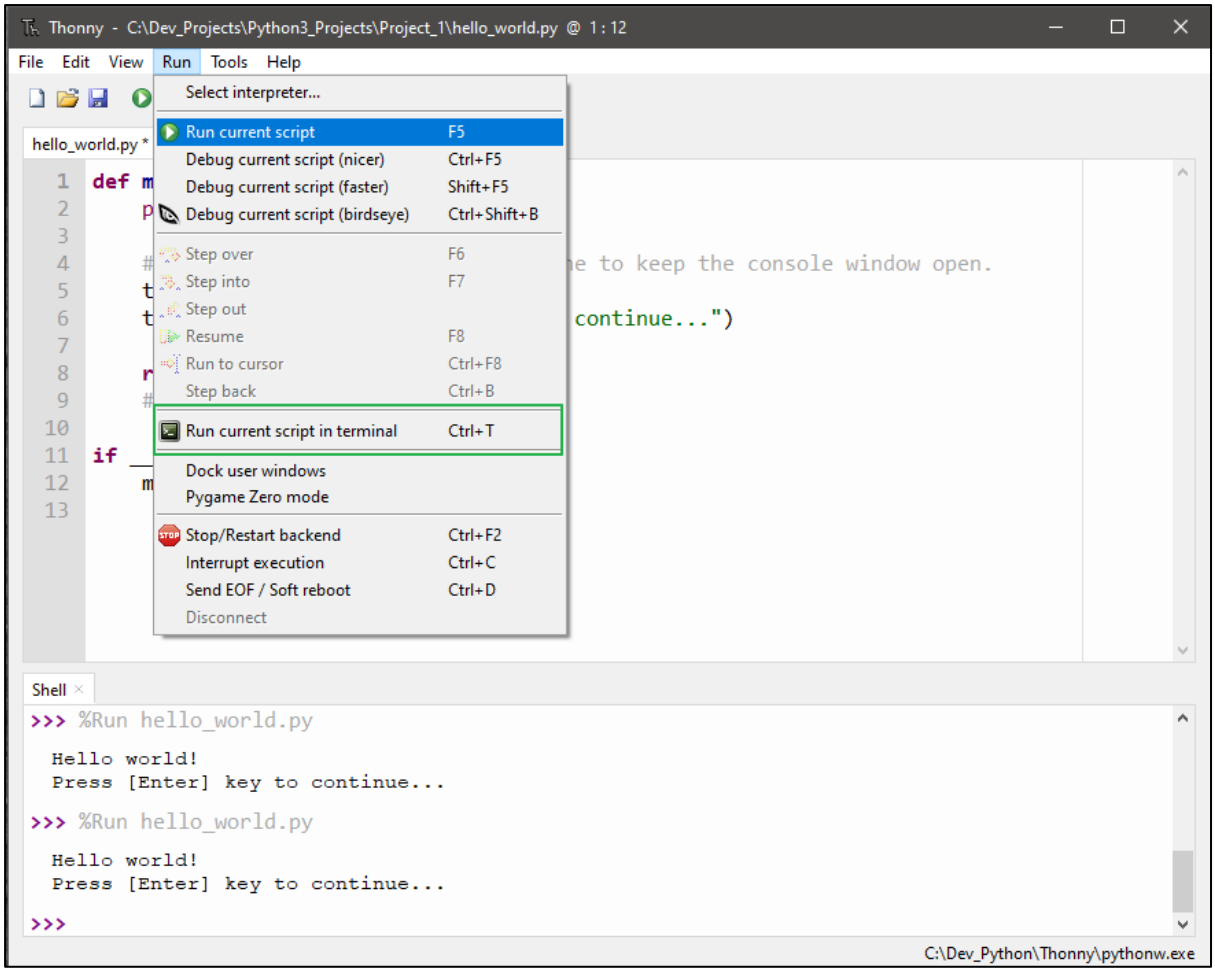


Next select the green run button to test the hello world script in Thonnys' REPL output window under the shell TAB in the lower part of the IDE. Or from the Menu bar select Run – Run current script.

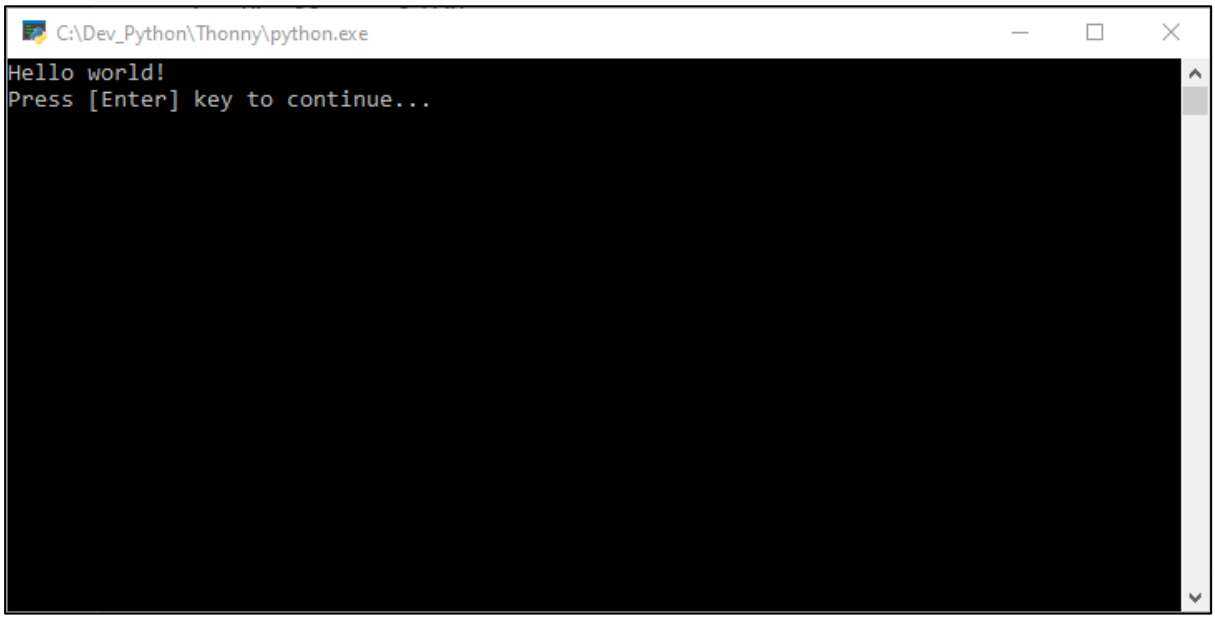


You should see the Hello world! Followed by the Press [Enter] key to continue... in the lower screen.

To test your application in the native console environment select run -> Run current script in terminal.



If all has worked well you should see the console window with Hello world! Followed by the Press [Enter] key to continue...



Press enter to close the console window.

From here you can create your own source examples to experiment with and test, or try some of the examples from “A Beginners Guide To Programming”. I recommend making a backup of your source code “Revisions” from time to time in a separate sub folder. I use Beanland AutoVer 2.2.1 to monitor and create revisions of my source and document projects.

Install Python3

This is optional as Python 3 is already built into Thonny.

If you choose to use a different IDE such as PyScripter (What I use) then you will also need to install Python 3 separately.

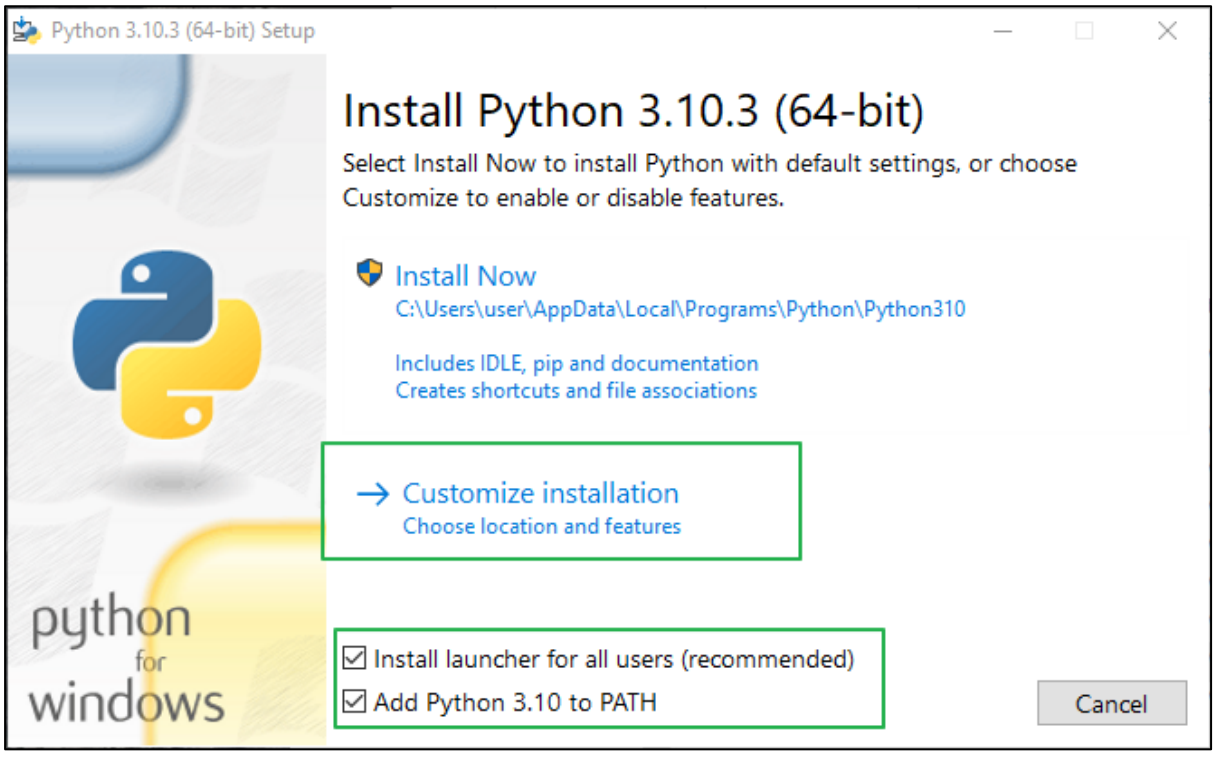
Installing the Python 3 development tools follows the same process for 32 bit and 64 bit. If you are using a 64 bit operating system you will only need to install the 64 bit version of python 3.

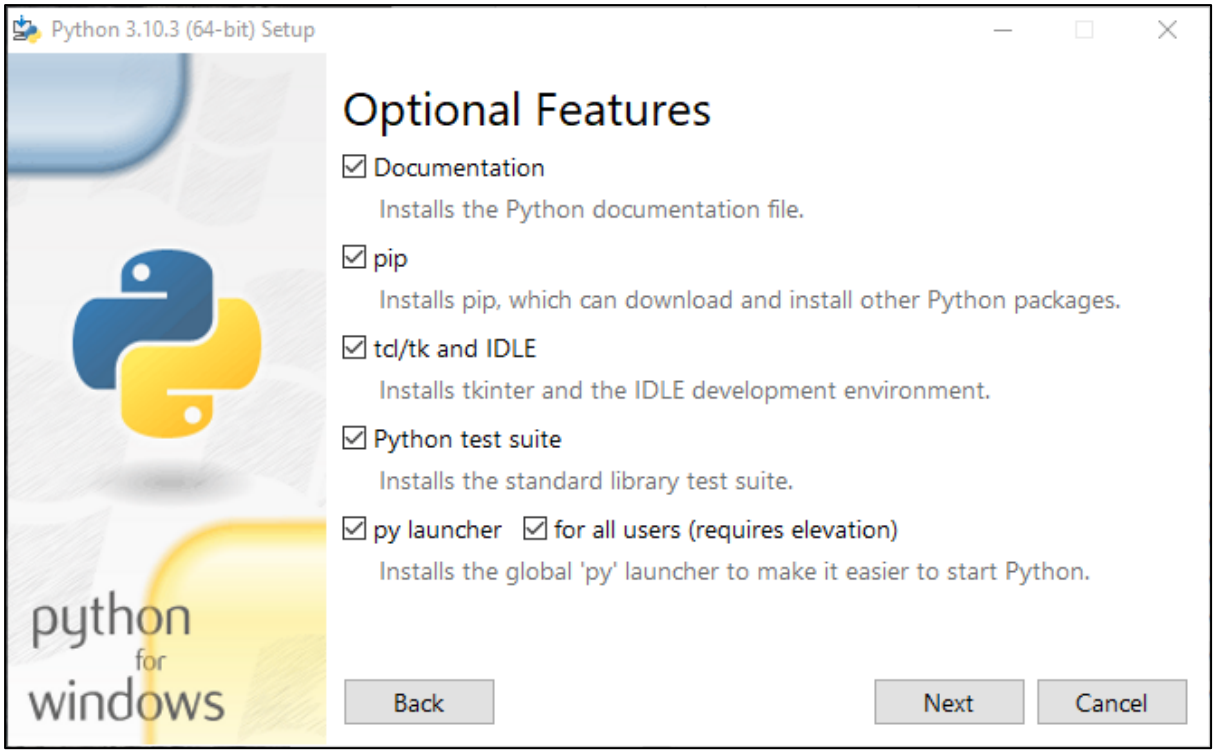
Adding the path variable to the OS environment will allow you to run a Python script from anywhere in the system without having to supply the full path to the interpreter. If you install more than one Python development stack then you can only set the path to a single version at a time.

If you do have intentions of installing more than one Python development environment then you may need to use a slightly altered method to what I will describe below.

I will install python 3 into the Dev_ folders in the root of my OS system drive, usually `C:\`. The native install path places files in a number of different places including “Program Files” as well as the user account files. This can lead to library path conflicts as well as issues with spaces in file names. It is just a common practice for many to use the OS drive root for development tools.

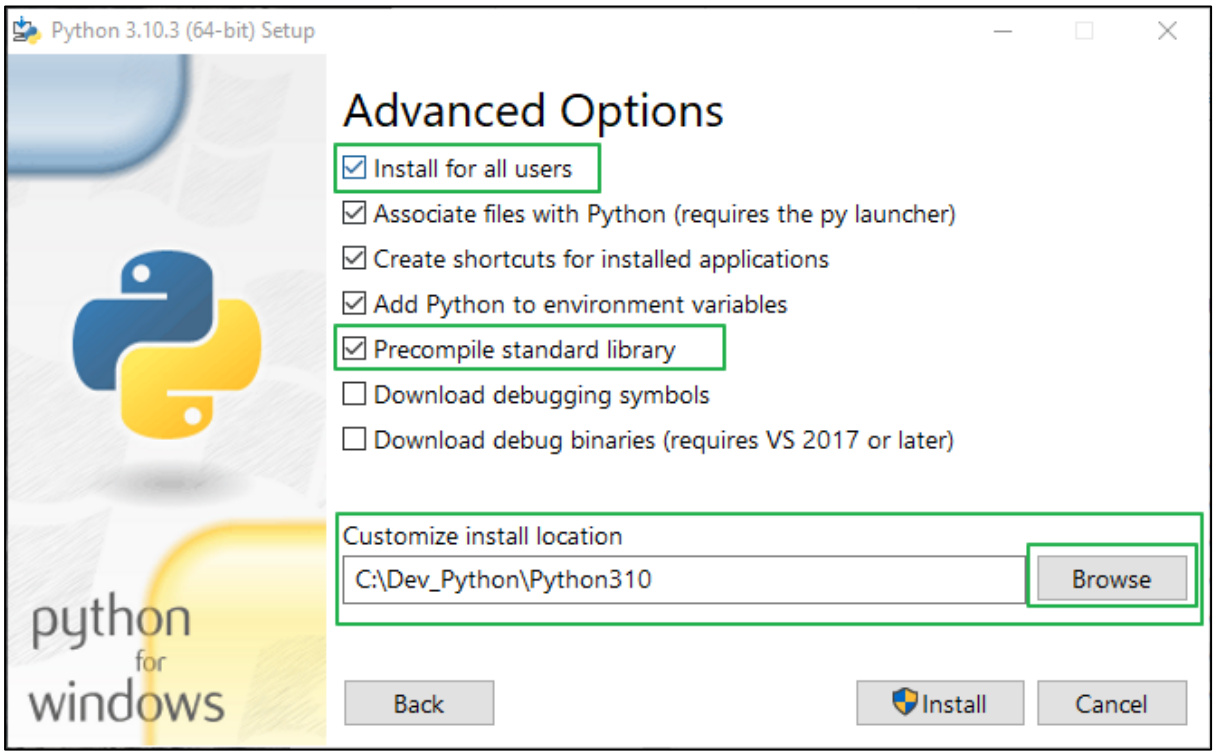
Open ”python-3.10.3-amd64.exe” (or the most recent version) and follow the install dialog with the following changes.



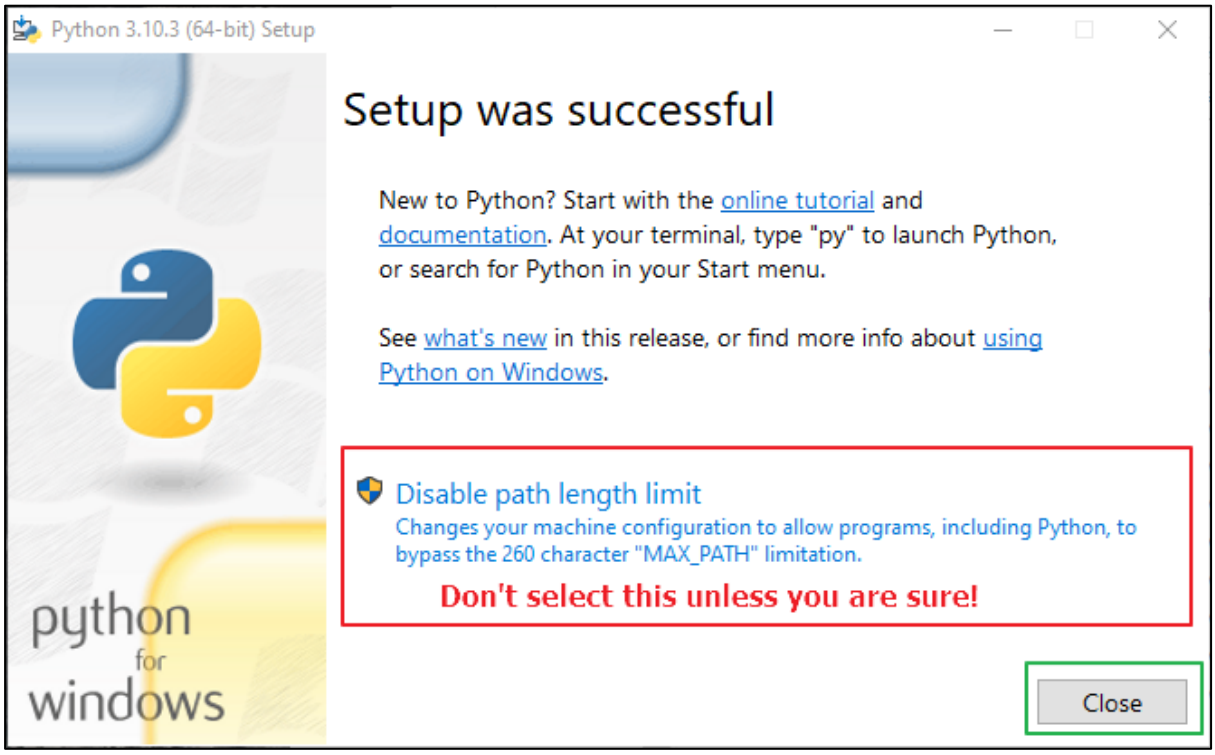


In the next dialog you will need to click browse to navigate to your "`C:\Dev_Python`" directory. You will need to add "`\Python310`" to the end of the path returned so it will look like "`C:\Dev_Python\Python310`".

"Install for all users" forces the entire python stack to be installed under a single directory path. No files should be added to the user account profile.

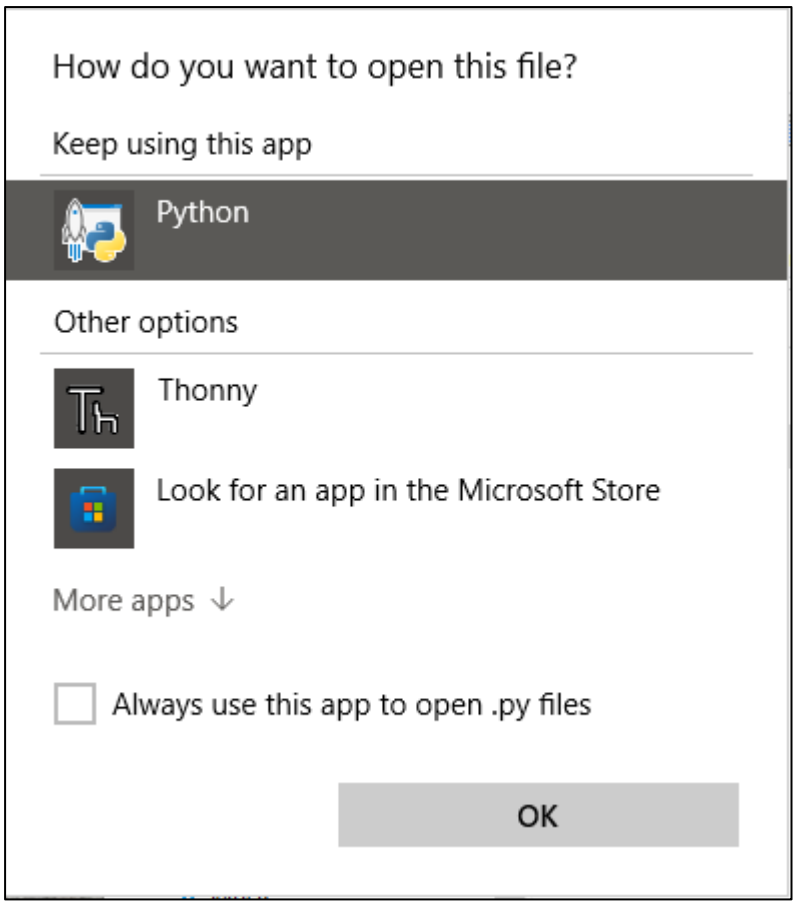


The final screen is optional and requires some research on your personal requirement for your system. Windows has a 260 character maximum limit for path names. The install method I have shown above places all python development files directly under the OS `C:\` root so the chance of a "File Name Too Long" conflict is highly unlikely. I recommend leaving the default 260 character limit intact if you are unsure.

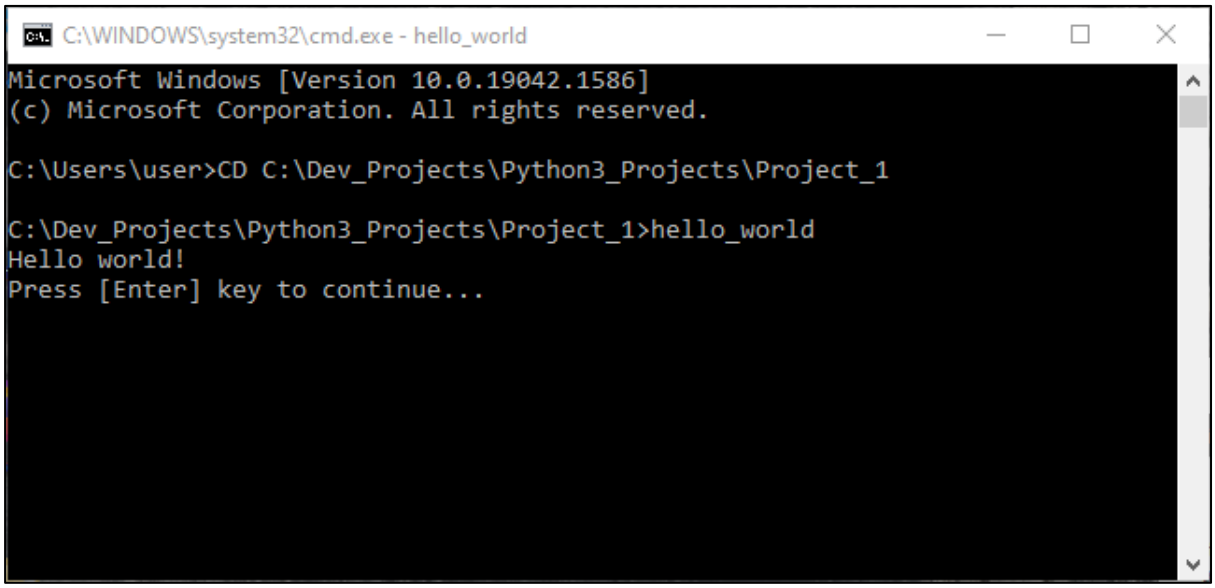


Navigate to the projects directory where you saved the "hello_world.py".
"`C:\Dev_Projects\Python3_Projects\Project_1\ hello_world.py`".

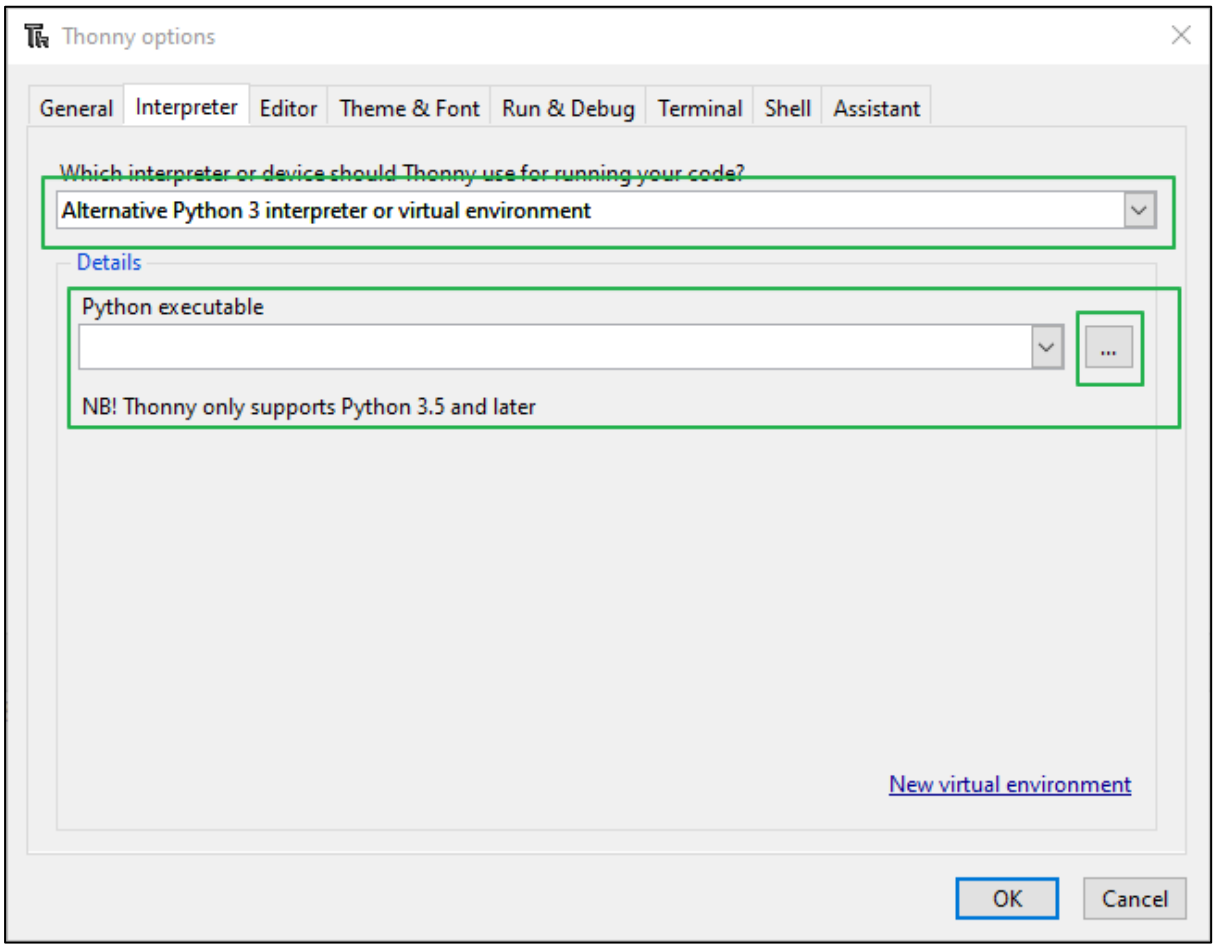
Open the script file "hello_world.py" and select "Keep using this app" - Python and OK if you are asked to confirm.



You should see your hello world script open in a console window. You can also open a console window and navigate to your projects folder and open your script by typing hello_word.py directly to the console.



You now have the option of selecting between Thonny's built in Python 3 interpreter or the installed python 3 development stack.



PyScripter IDE

PyScripter is a free and open-source Python Integrated Development Environment (IDE) created with the ambition to become competitive in functionality with commercial Windows-based IDEs available for other languages.

This is an alternative IDE to Thonny.

PyScripter requires that a recent version of Python exist on the system.

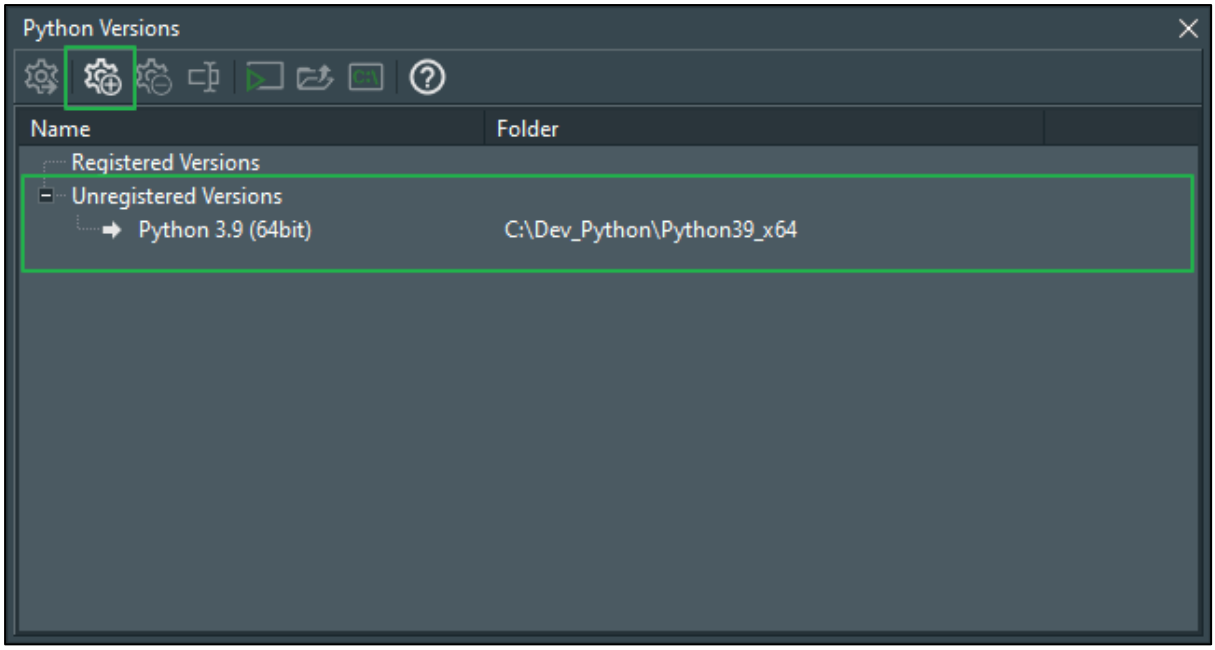
Open the " PyScripter-4.2.5-x64-Setup.exe" installer and follow the install steps using the default install path and components.

If you prefer you can install PyScripter to a custom path as I have done for other development environment setups; for example "C:\Dev_Python\ PyScripter *.*"

PyScripter will automatically detect the Python install if it has been set into the systems environment path. If not you can set the Python directory as follows.

T set the path to the installed Python 3 –x64 interpreter directory. Open “Run -> Python Versions -> Setup Python Versions...”

Select the “Add a new Python version” icon at the top and navigate to the root folder of you Python installation you want to use. For this guide I am using a custom directory “C:\Dev_Python\Python39_x64”.



Close the dialog box [X] to save.

Follow the guides in “Help -> PyScripter -> Contents” for customising the editor.

That is the end of the install guides for getting your essential development tools ready for trying the example in “A Beginners Guide To Programming” under a Windows Operating system.

Happy coding ☺

Ubuntu 64 bit Development environments

You do not have to install all of the IDEs and only choose what you require. If you would like to attempt all the code examples from "A Beginners Guide To Programming" then you will need to install all of the IDEs.

If you are an experienced Linux user you can follow the install order below adding the development tools that you require.

I am using the lite Ubuntu flavour Lubuntu 20.04 x64 for the install instructions. Lubuntu offers an easy to use desktop and UI and may feel a little more familiar if you are coming from Windows with little Linux experience. Lubuntu is also a very lightweight system with low resource use and is shipped with only a minimal amount of pre-installed applications making it ideal for use in a virtual machine environment.

The following install instructions are valid for any of the Ubuntu 20.04 x64 mainstream distributions.

If you are planning on using a guest OS in a VirtualBox environment please be sure to read the section under "Supplemental".

Requirements (or most recent versions)

- **Perform system updates.**
  ```
  sudo apt-get update
  sudo apt-get upgrade
  ```
- **p7zip**
  ```
  sudo apt install p7zip
  ```
- **Alacarte Menu Editor**
  ```
  sudo apt install alacarte
  ```
- **xCHM**
  Compiled help viewer
  ```
  sudo apt-get install xchm
  ```
- **Okular**
  PDF/ebook reader
  ```
  sudo apt install okular
  ```
- ~~**General Purpose Mouse**~~
  ~~`apt install gpm`~~
- ~~**Midnight Commander**~~
  ~~`apt install mc`~~
- **htop**
  ```
  sudo apt install htop
  ```
- **cppcheck (CLI)**
  ```
  sudo apt-get install cppcheck
  sudo apt-get install cppcheck-gui
  ```
- ~~**SciTe**~~
  ~~`sudo apt-get install -y scite`~~

Geany preferred (Geany and scite both hide '_' in some fonts)
Also consider feather pad that comes with Lubuntu

- **C/C++ Build environment**
  sudo apt install build-essential
- **Code::Blocks (20.03)**
  sudo apt install codeblocks

- **FreeBASIC Compiler**
  https://sourceforge.net/projects/fbc/files/
  https://sourceforge.net/projects/fbc/files/FreeBASIC-1.09.0/Binaries-Linux/
  Download for your Ubuntu version.
  ubuntu-20.04/ FreeBASIC-1.09.0-ubuntu-20.04-x86_64.tar.gz
  Additional Libraries:
  sudo apt install -y gcc libncurses5-dev libffi-dev libgl1-mesa-dev libx11-dev libxext-dev libxrender-dev libxrandr-dev libxpm-dev

- **geany-1.38_setup.exe**
  Works out of the box for FB :)
  apt-get install geany

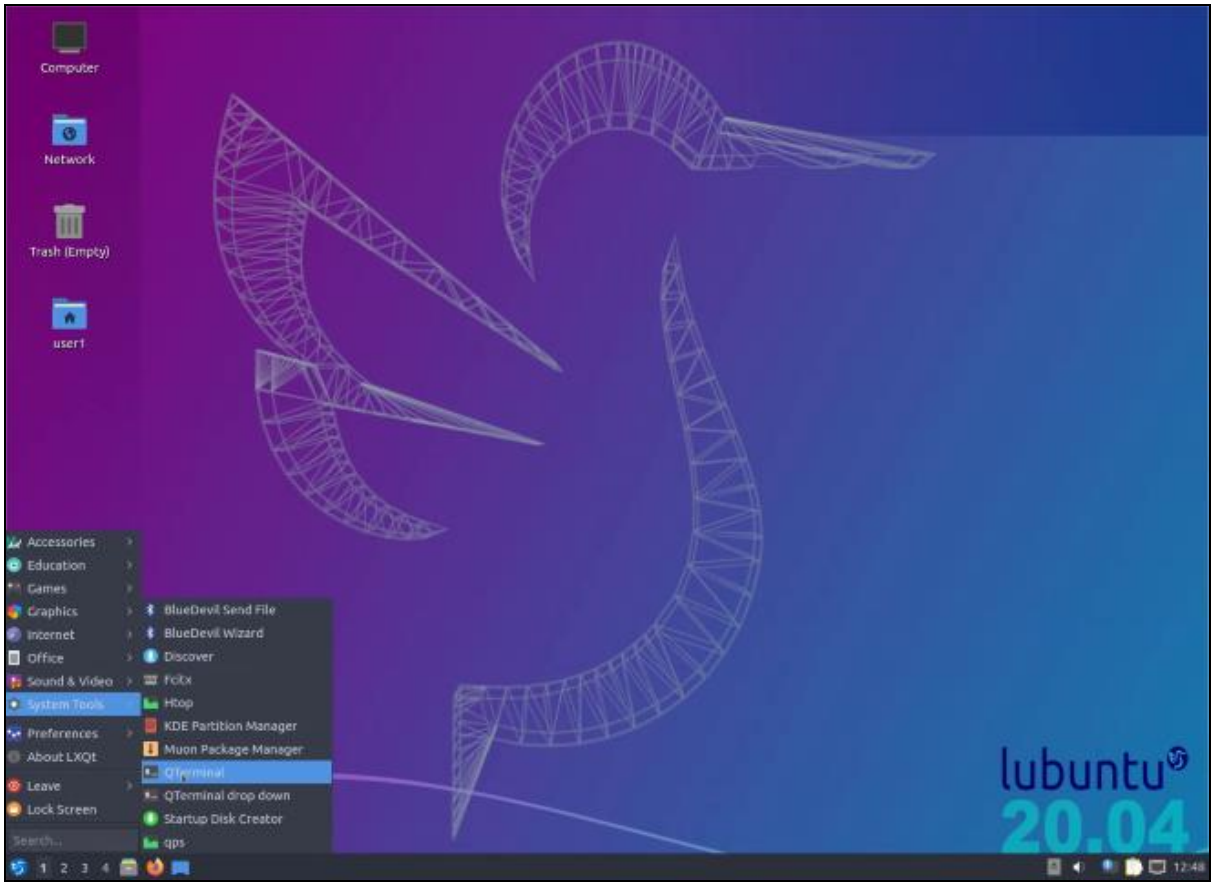
- **python-3.10.3-amd64.exe** This is not mandatory if using Thonny as Thonny has the Python 3 development bundle built into the application install.

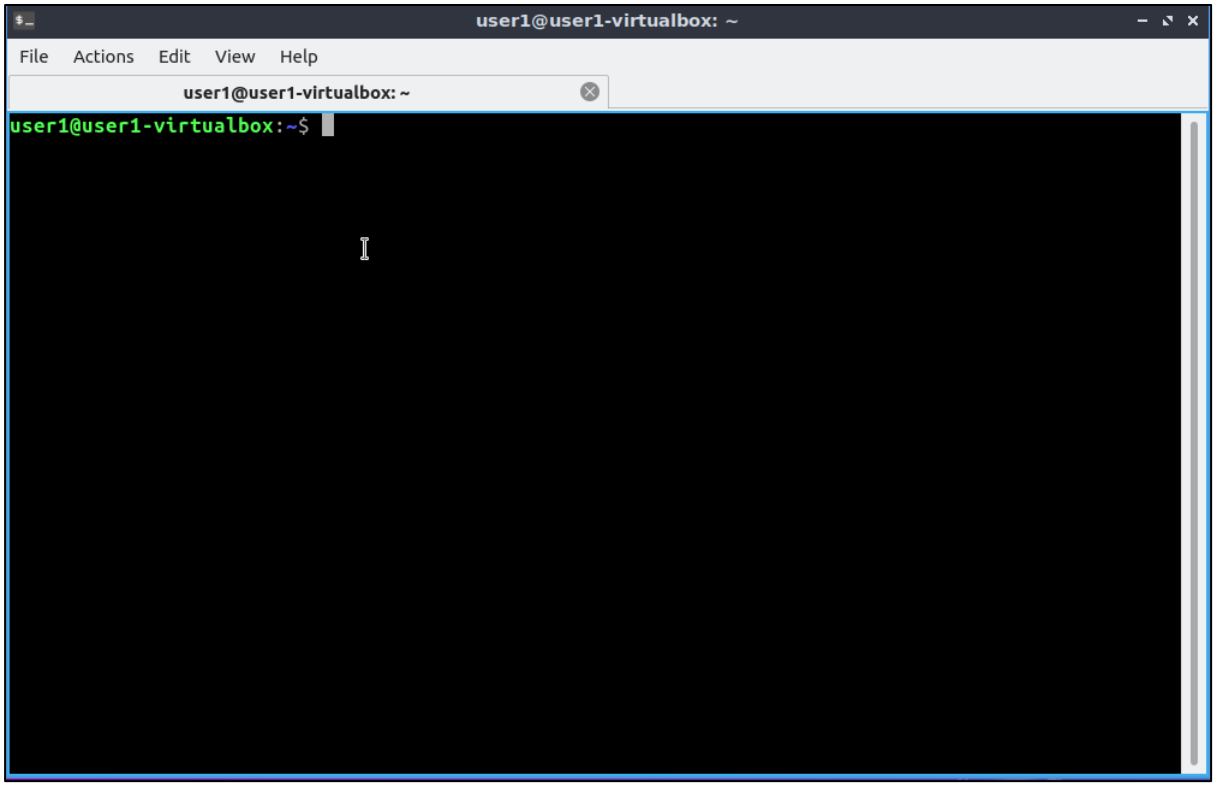
- **Thonny (3.2.7.1)**
  sudo apt install python3-tk thonny

Running the system update and installing the essential utilities.

Perform system updates.

Open a command terminal and enter the following.





Run App packages update.

```
sudo apt-get update
```

You will need to enter the password to continue…

```
user1@user1-virtualbox:~$ sudo apt-get update
[sudo] password for user1:
Hit:1 http://au.archive.ubuntu.com/ubuntu focal InRelease
Get:2 http://au.archive.ubuntu.com/ubuntu focal-updates InRelease [114 kB]
Get:3 http://au.archive.ubuntu.com/ubuntu focal-backports InRelease [108 kB]
Get:4 http://au.archive.ubuntu.com/ubuntu focal-updates/main i386 Packages [630 kB]
Get:5 http://security.ubuntu.com/ubuntu focal-security InRelease [114 kB]
Get:6 http://au.archive.ubuntu.com/ubuntu focal-updates/main amd64 Packages [1,713 kB]
Get:7 http://au.archive.ubuntu.com/ubuntu focal-updates/main Translation-en [321 kB]
Get:8 http://au.archive.ubuntu.com/ubuntu focal-updates/main amd64 DEP-11 Metadata [278 kB]
Get:9 http://au.archive.ubuntu.com/ubuntu focal-updates/main DEP-11 48x48 Icons [60.8 kB]
Get:10 http://au.archive.ubuntu.com/ubuntu focal-updates/main DEP-11 64x64 Icons [98.3 kB]
Get:11 http://au.archive.ubuntu.com/ubuntu focal-updates/main DEP-11 128x128 Icons [202 kB]
Get:12 http://au.archive.ubuntu.com/ubuntu focal-updates/main amd64 c-n-f Metadata [14.9 kB]
Get:13 http://au.archive.ubuntu.com/ubuntu focal-updates/restricted amd64 Packages [917 kB]
Get:14 http://au.archive.ubuntu.com/ubuntu focal-updates/restricted i386 Packages [24.3 kB]
Get:15 http://au.archive.ubuntu.com/ubuntu focal-updates/restricted Translation-en [131 kB]
Get:16 http://au.archive.ubuntu.com/ubuntu focal-updates/restricted amd64 c-n-f Metadata [528 B]
Get:17 http://au.archive.ubuntu.com/ubuntu focal-updates/universe amd64 Packages [918 kB]
Get:18 http://au.archive.ubuntu.com/ubuntu focal-updates/universe i386 Packages [677 kB]
Get:19 http://au.archive.ubuntu.com/ubuntu focal-updates/universe Translation-en [205 kB]
Get:20 http://au.archive.ubuntu.com/ubuntu focal-updates/universe amd64 DEP-11 Metadata [391 kB]
Get:21 http://au.archive.ubuntu.com/ubuntu focal-updates/universe DEP-11 48x48 Icons [257 kB]
Get:22 http://au.archive.ubuntu.com/ubuntu focal-updates/universe DEP-11 64x64 Icons [458 kB]
Get:23 http://au.archive.ubuntu.com/ubuntu focal-updates/universe DEP-11 128x128 Icons [1,036 kB]
Get:24 http://au.archive.ubuntu.com/ubuntu focal-updates/universe amd64 c-n-f Metadata [20.6 kB]
Get:25 http://au.archive.ubuntu.com/ubuntu focal-updates/multiverse i386 Packages [8,432 B]
```

Upgrade Ubuntu packages to the latest version.

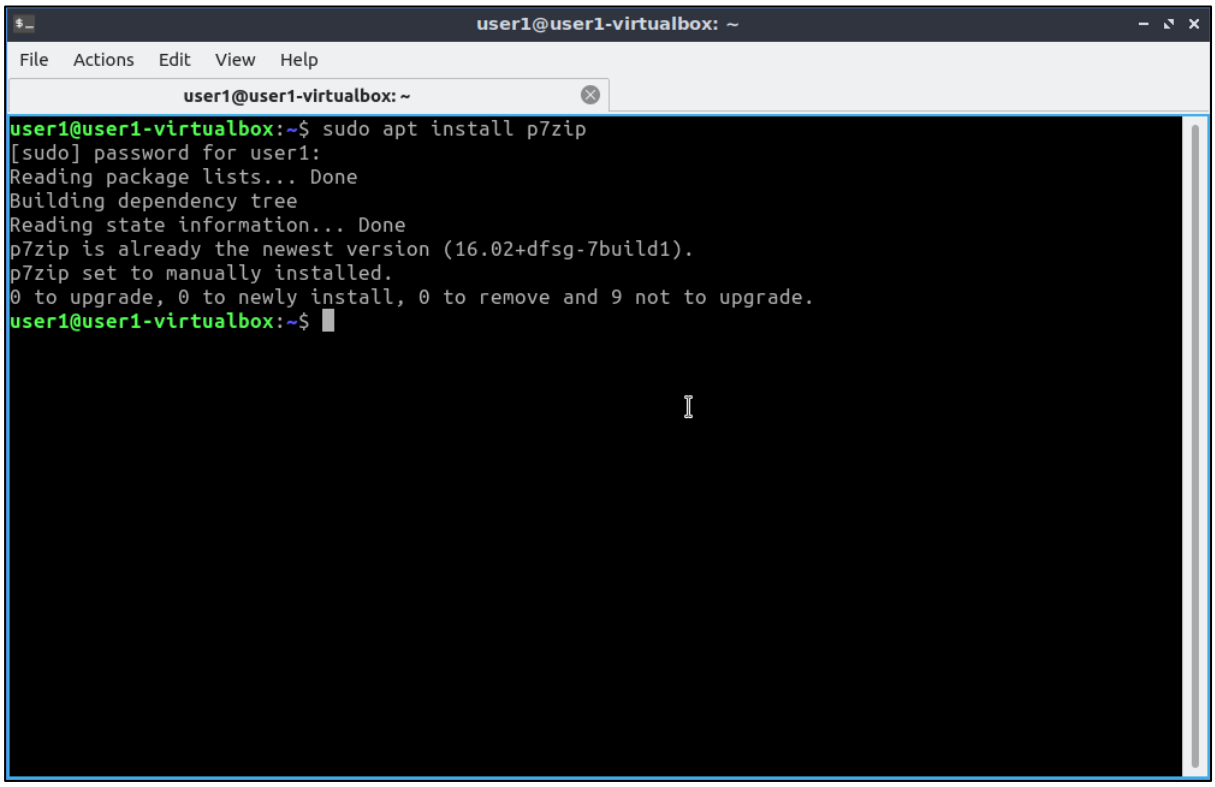
`sudo apt-get upgrade`

Select ‘Y’ to continue updates…

```
user1@user1-virtualbox: ~
File  Actions  Edit  View  Help
user1@user1-virtualbox: ~
Unpacking linux-libc-dev:amd64 (5.4.0-107.121) over (5.4.0-88.99) ...
Preparing to unpack .../libc6_2.31-0ubuntu9.7_amd64.deb ...
Unpacking libc6:amd64 (2.31-0ubuntu9.7) over (2.31-0ubuntu9.2) ...
Setting up libc6:amd64 (2.31-0ubuntu9.7) ...
(Reading database ... 275691 files and directories currently installed.)
Preparing to unpack .../base-files_11ubuntu5.5_amd64.deb ...
Warning: Stopping motd-news.service, but it can still be activated by:
  motd-news.timer
Unpacking base-files (11ubuntu5.5) over (11ubuntu5.4) ...
Setting up base-files (11ubuntu5.5) ...
Installing new version of config file /etc/issue ...
Installing new version of config file /etc/issue.net ...
Installing new version of config file /etc/lsb-release ...
motd-news.service is a disabled or a static unit, not starting it.
(Reading database ... 275691 files and directories currently installed.)
Preparing to unpack .../bsdutils_1%3a2.34-0.1ubuntu9.3_amd64.deb ...
Unpacking bsdutils (1:2.34-0.1ubuntu9.3) over (1:2.34-0.1ubuntu9.1) ...
Setting up bsdutils (1:2.34-0.1ubuntu9.3) ...
(Reading database ... 275691 files and directories currently installed.)
Preparing to unpack .../gzip_1.10-0ubuntu4.1_amd64.deb ...
Unpacking gzip (1.10-0ubuntu4.1) over (1.10-0ubuntu4) ...
Setting up gzip (1.10-0ubuntu4.1) ...
(Reading database ... 275691 files and directories currently installed.)
Preparing to unpack .../tar_1.30+dfsg-7ubuntu0.20.04.2_amd64.deb ...
Unpacking tar (1.30+dfsg-7ubuntu0.20.04.2) over (1.30+dfsg-7ubuntu0.20.04.1) ...
Setting up tar (1.30+dfsg-7ubuntu0.20.04.2) ...
(Reading database ... 275691 files and directories currently installed.)
```

Install p7zip for working with file archives. (Already installed by default in Lubuntu)

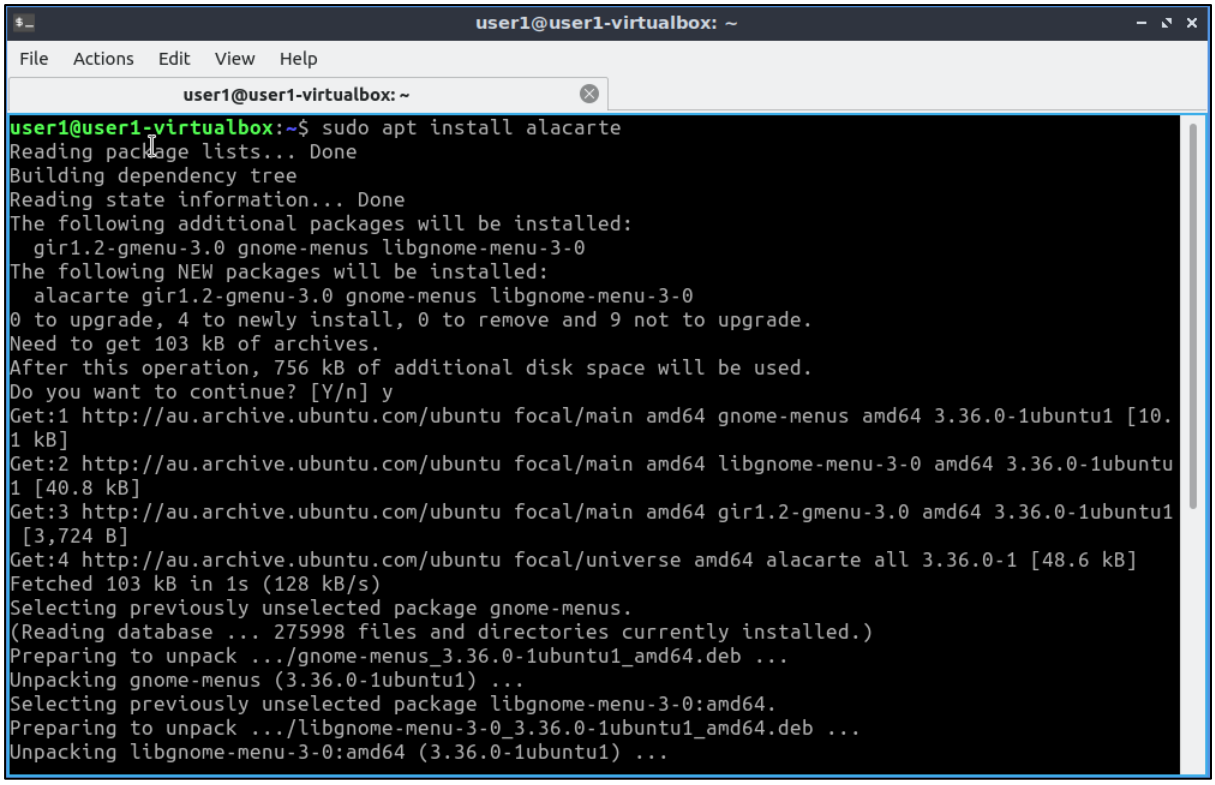
```
sudo apt install p7zip
```



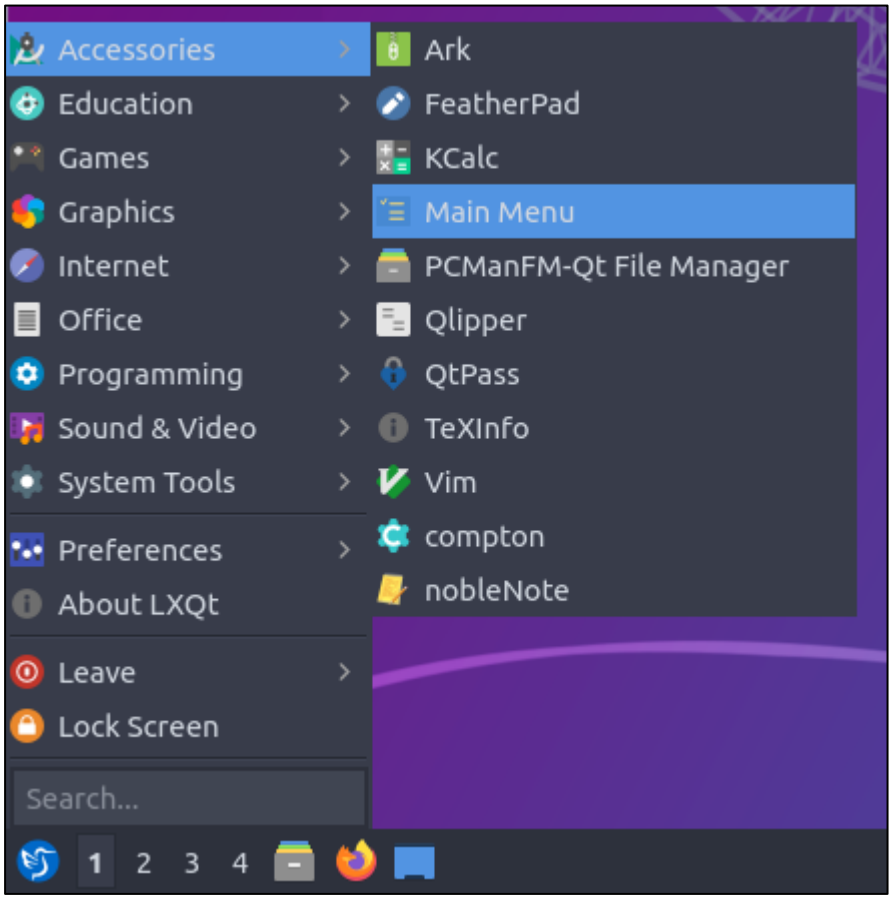
Install Alacarte Menu Editor.

This will make it easier to edit and create custom start menu entries.

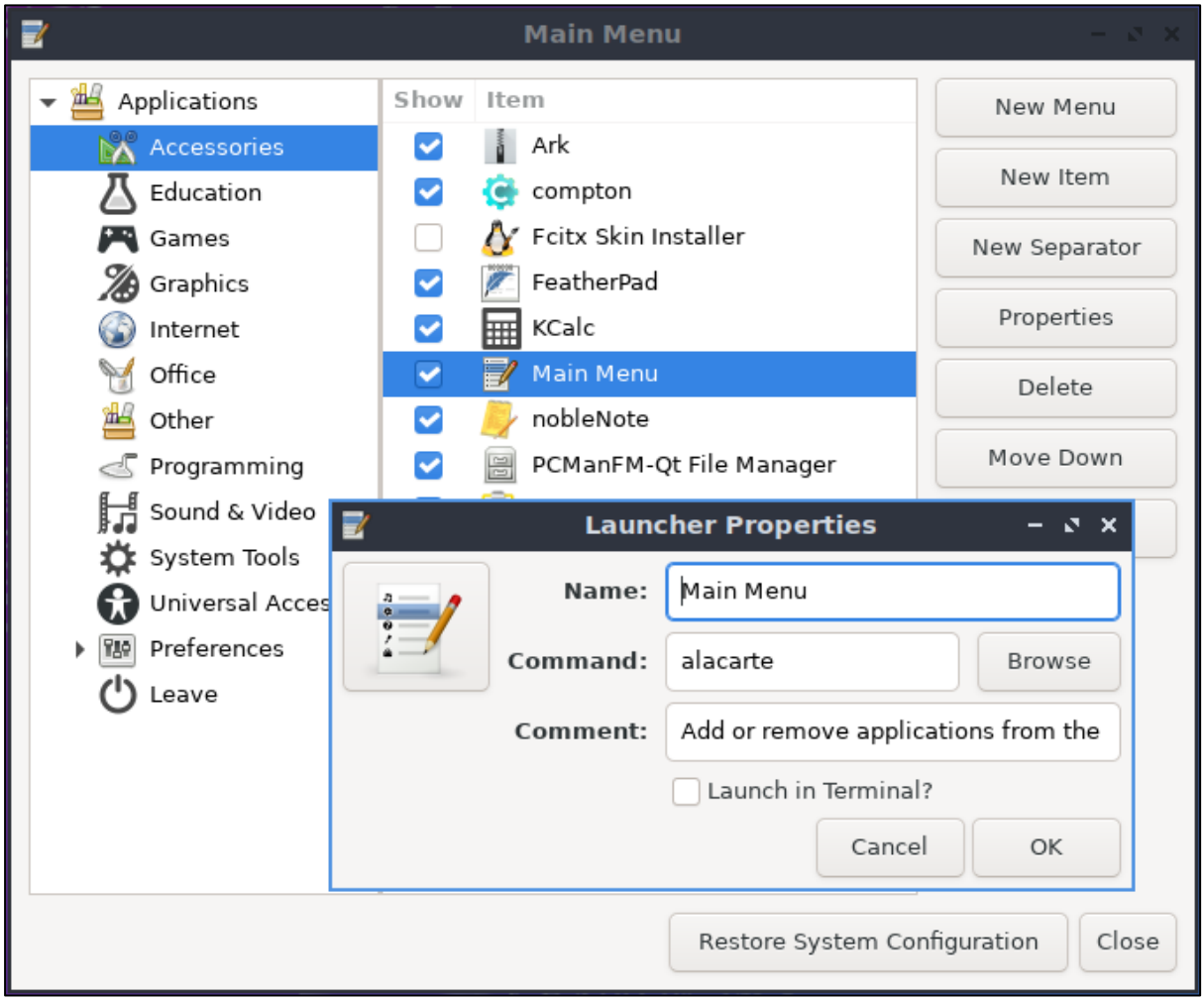
```
sudo apt install alacarte
```

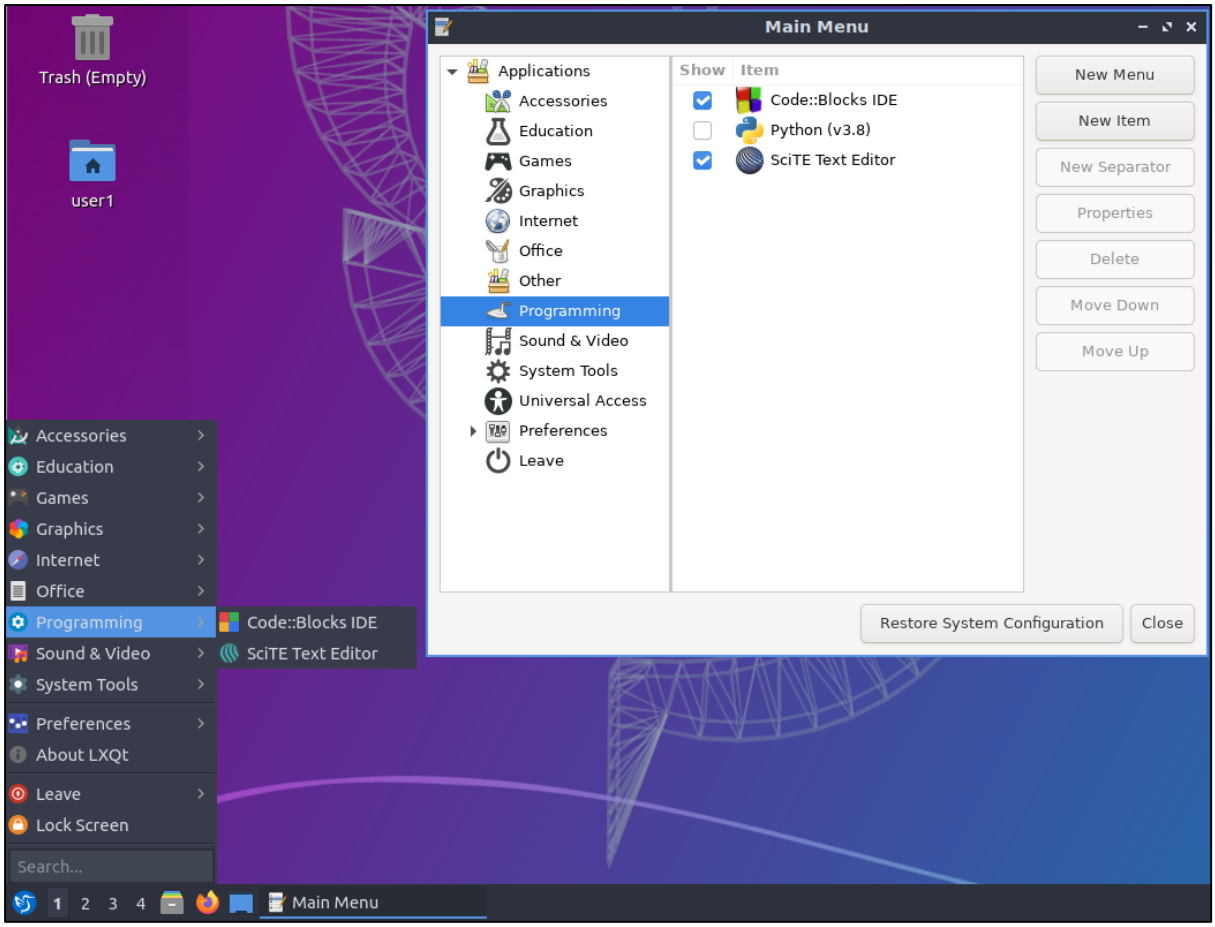


To open and use Alacarte… “Main Menu”.

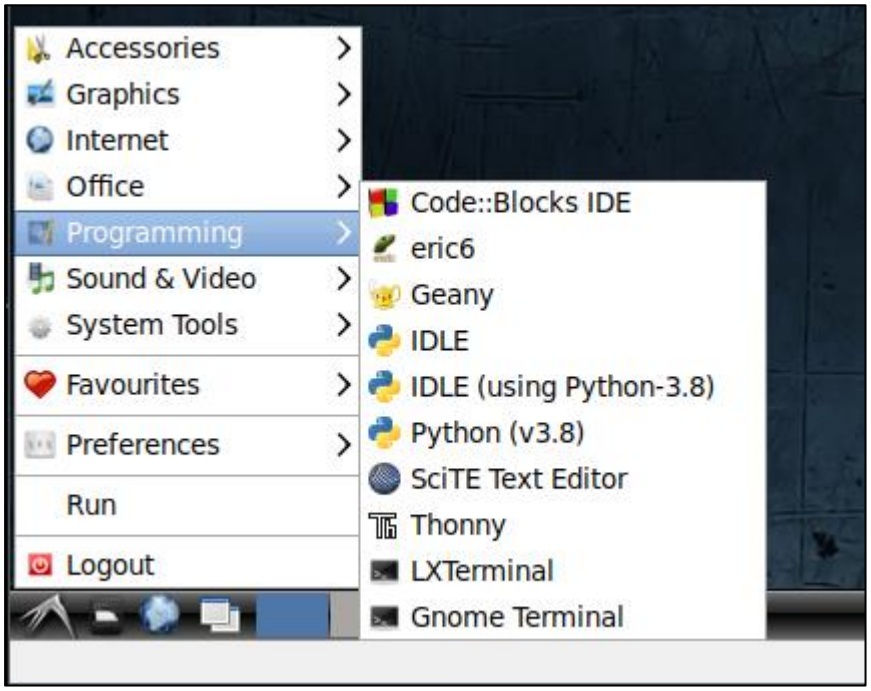


You cannot drag and drop, so best to open a text document and keep a record of the properties for each menu entry that you wish to recreate in “Launcher Properties”. I tend to make copies a few extra applications under Programming so that I have most tools in the one menu entry.





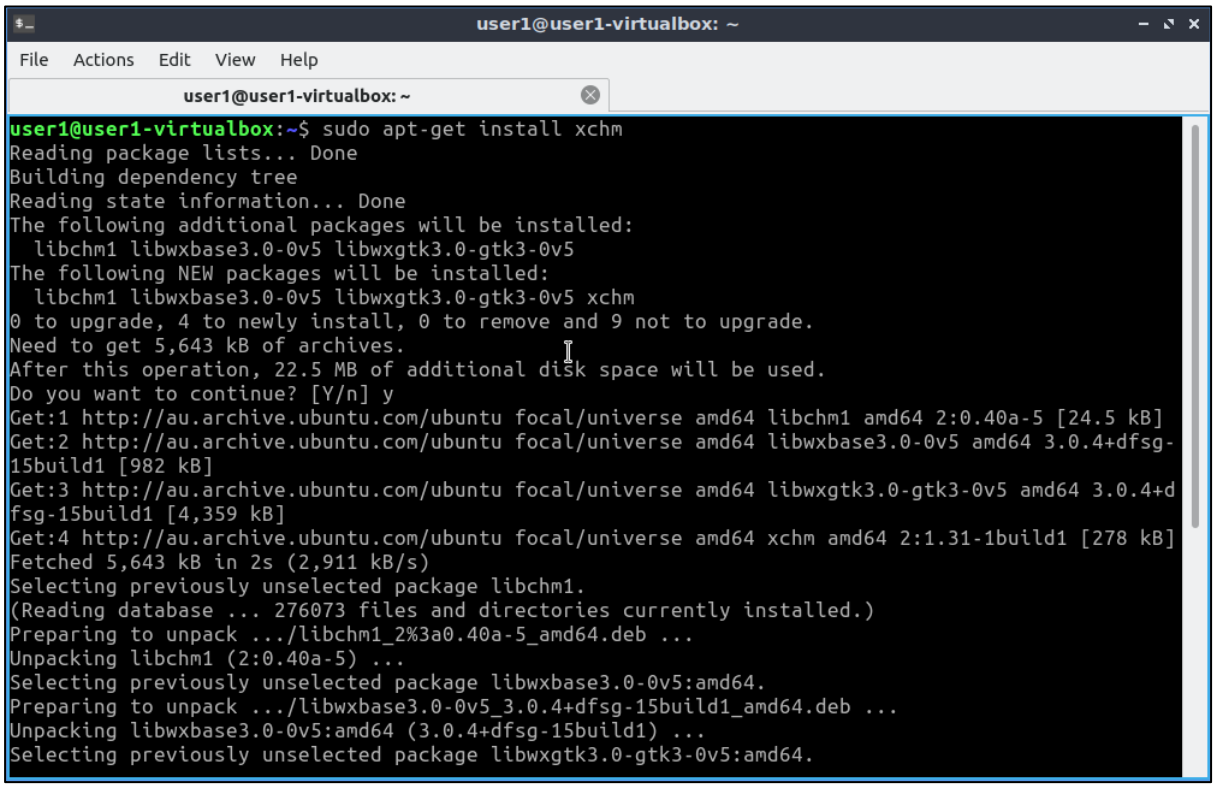
From my own personal Linux build.



Install xCHM to allow viewing of compressed html help files.

Compiled help viewer

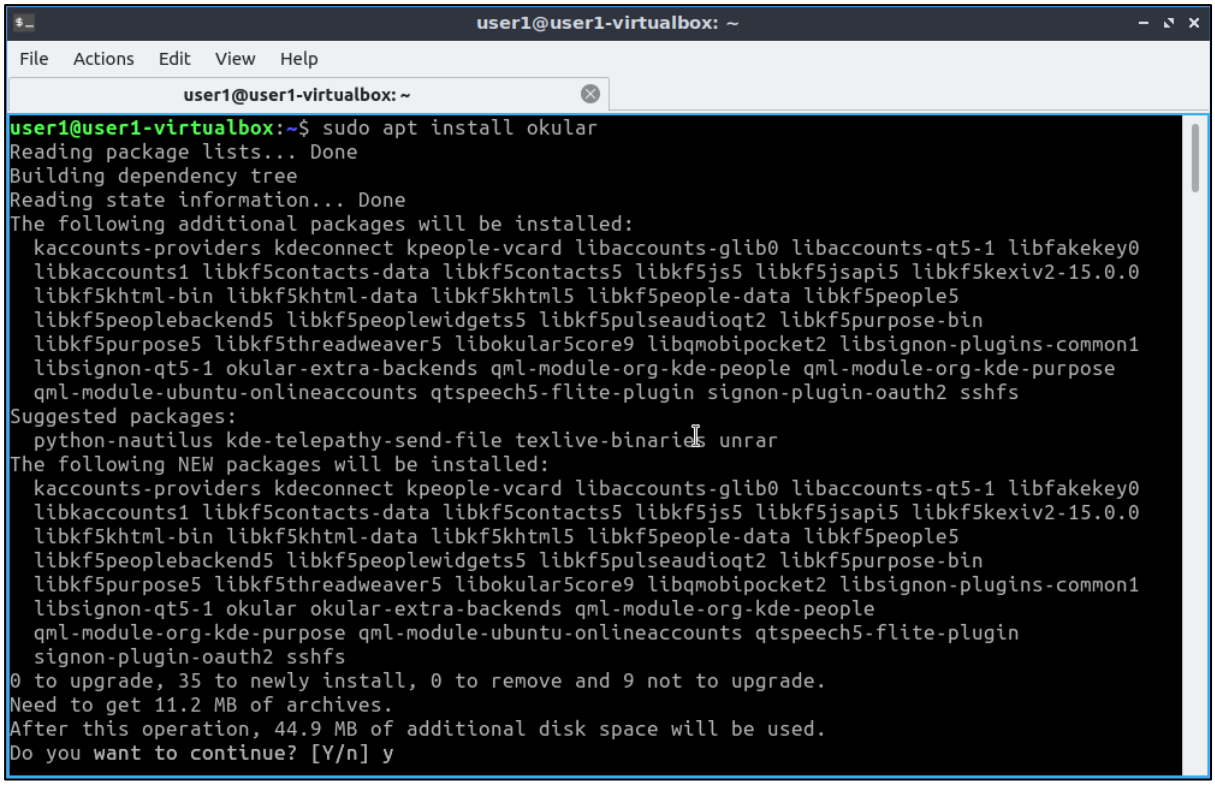
```
sudo apt-get install xchm
```



Install Okular to allow the viewing of PDF and other help files.

Okular is a PDF/ebook reader.

```
sudo apt install okular
```



~~**General Purpose Mouse**~~

~~Only useful if you need to spend a lot of time working in the Linux terminal without the desktop GUI tools.~~

~~`apt install gpm`~~

~~**Install Midnight Commander terminal based file system browser**~~

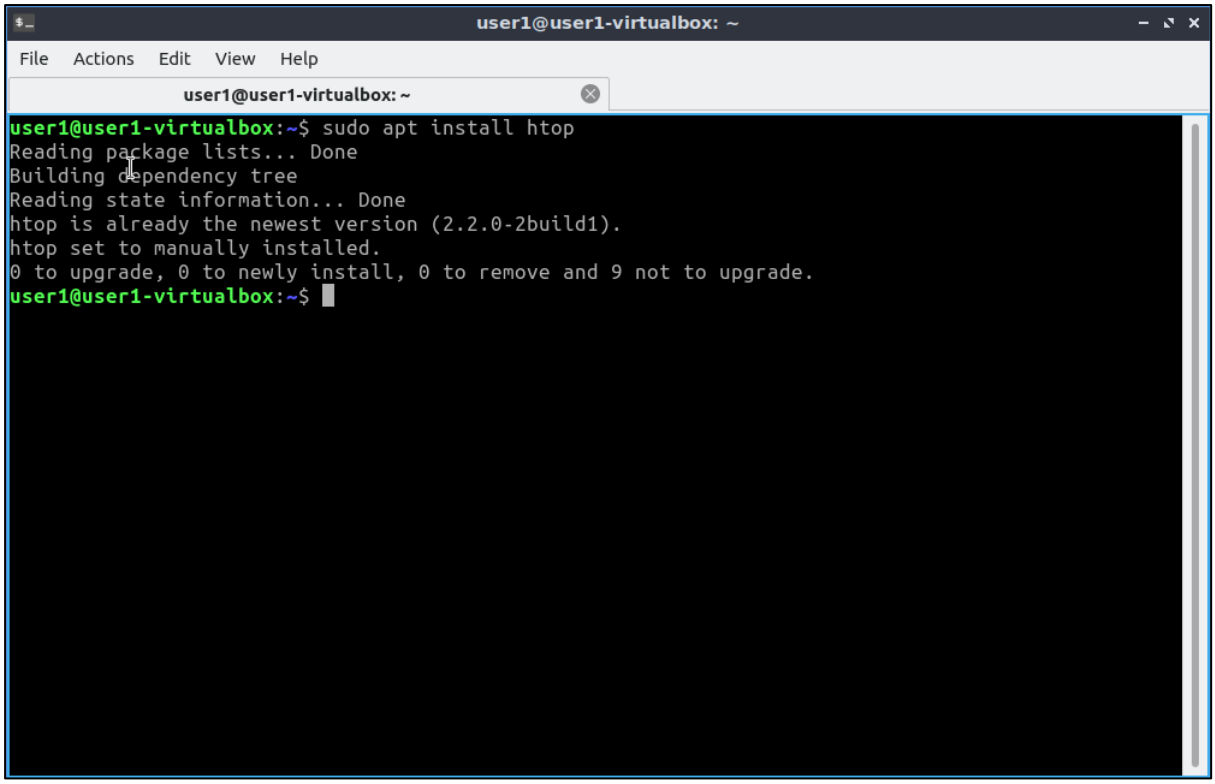
~~Only useful if you need to spend a lot of time working in the Linux terminal without the desktop GUI tools.~~

~~`apt install mc`~~

Install htop (Already installed by default in Lubuntu)

Htop is a cross-platform interactive process viewer and system monitoring tool. It is a text-mode application (for console or X terminals). Helpful to monitor the system resources used by the applications that you create.

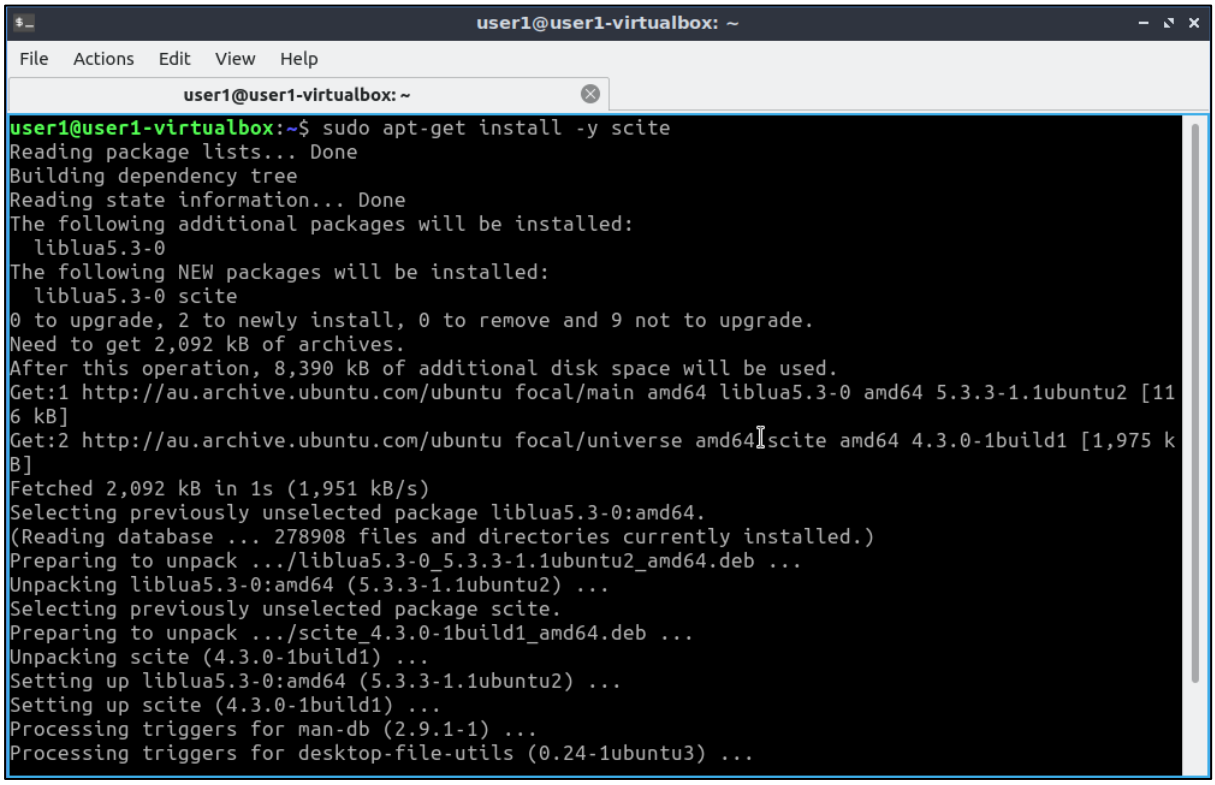
`sudo apt install htop`



Install SciTe Text editor

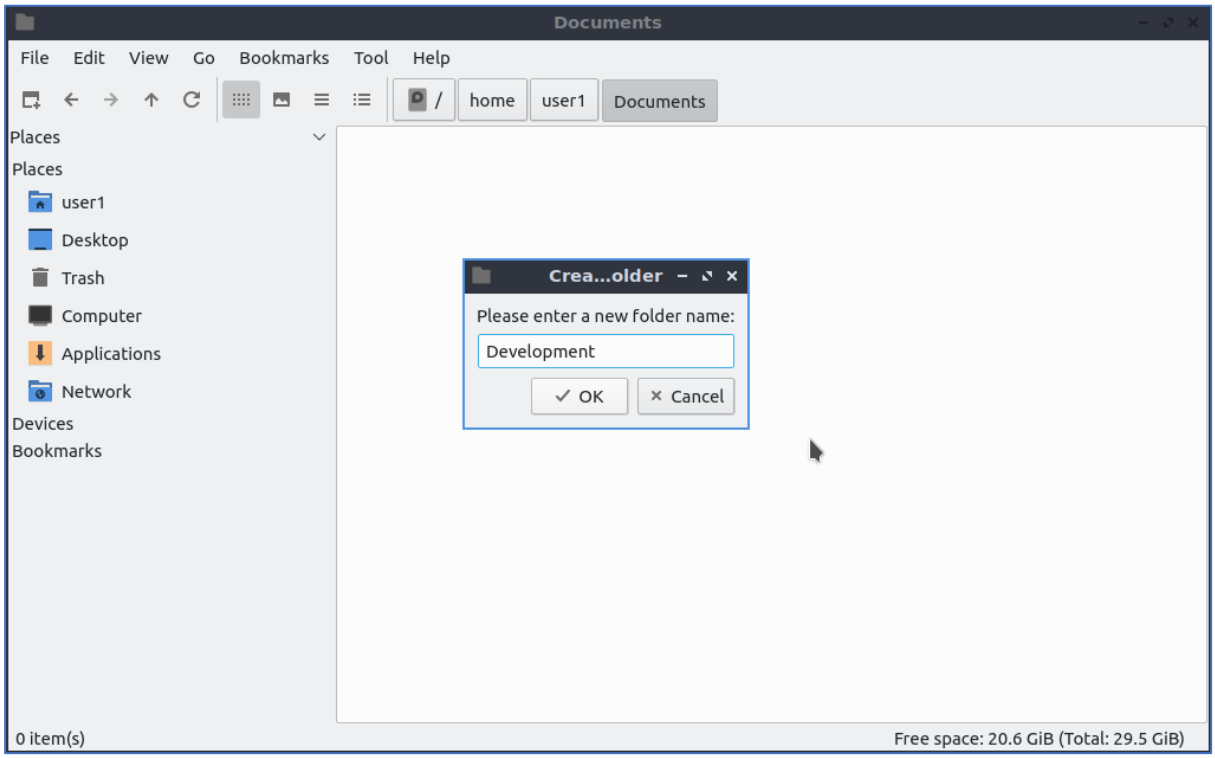
Scite is a general purpose text editor with syntax highlighting for many different source code formats.

`sudo apt-get install -y scite`

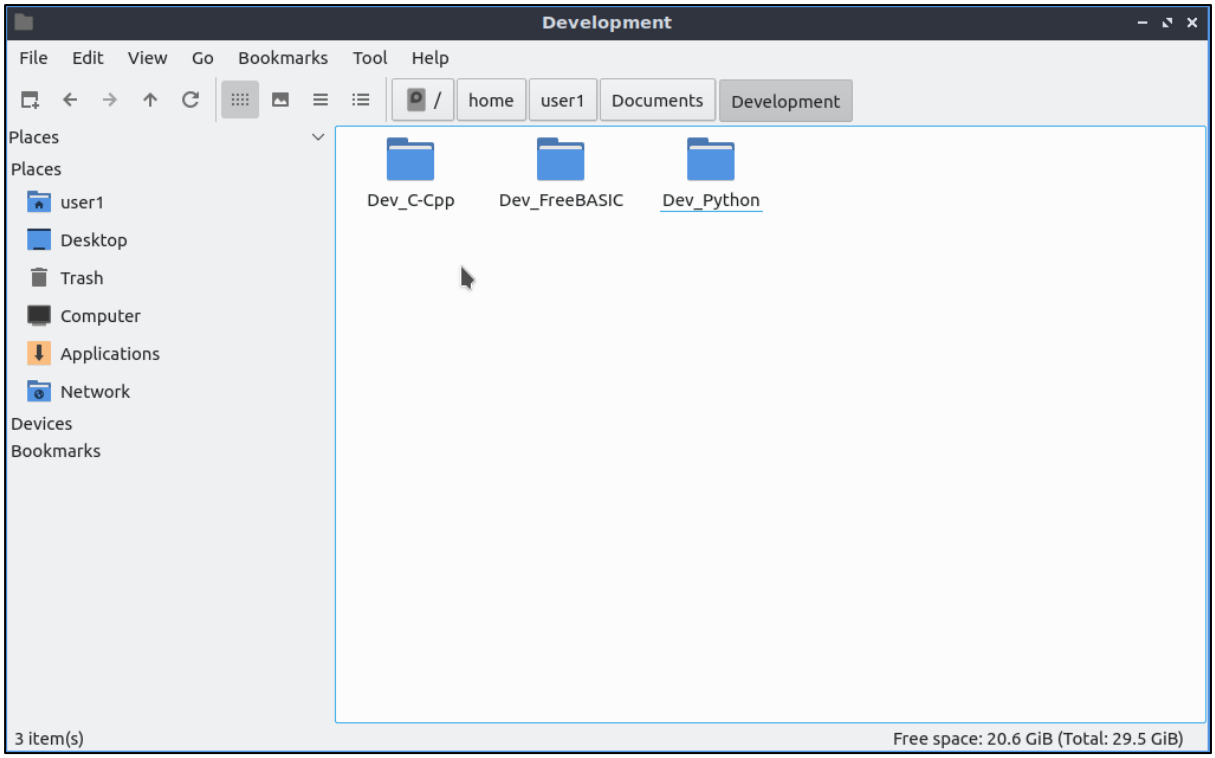


Creating base project folders

Open the PCMan-FM file manager and navigate to the "`/home/username/Documents`" directory and create a new directory named "`/Development`". (Right click, Create New -> Folder)



Inside of your /Development folder create another 3 folders to hold your source code and projects for the 3 programming languages; Dev_C-Cpp, Dev_FreeBASIC and Dev_Python.



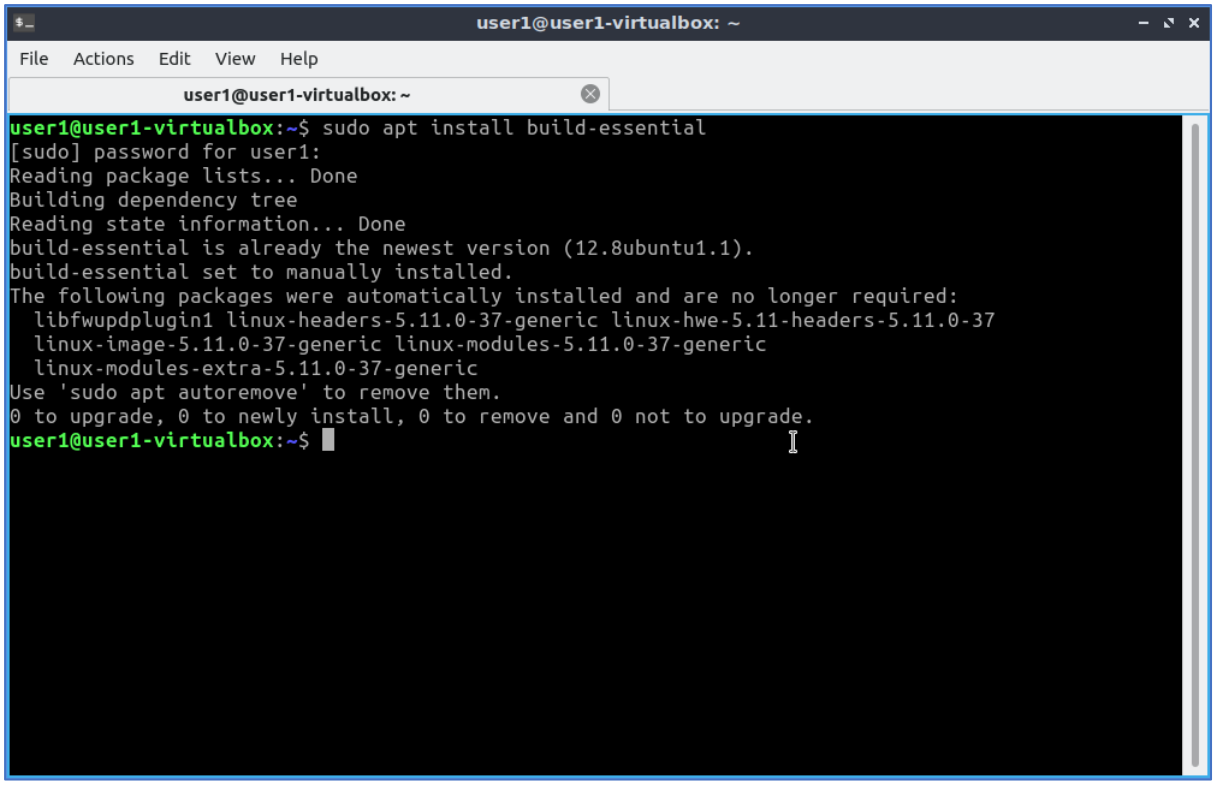
We will make use of these folders later as we set up our 3 IDEs for each language.

Install C and GCC development tools

C/C++ Build environment
The following will install the essential GCC (Gnu C Compiler), libraries and tool chains required for C/C++ software development

Place the following into the command line terminal.

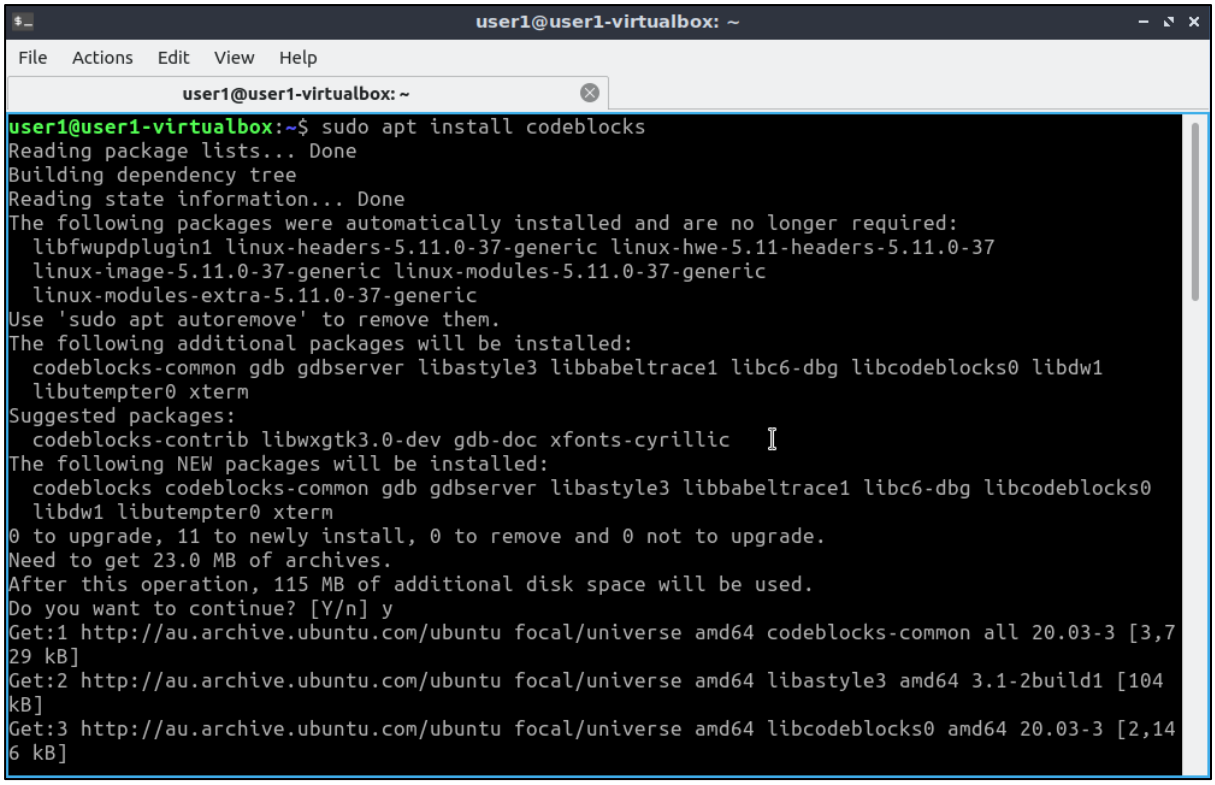
```
sudo apt install build-essential
```



Code::Blocks (20.03)

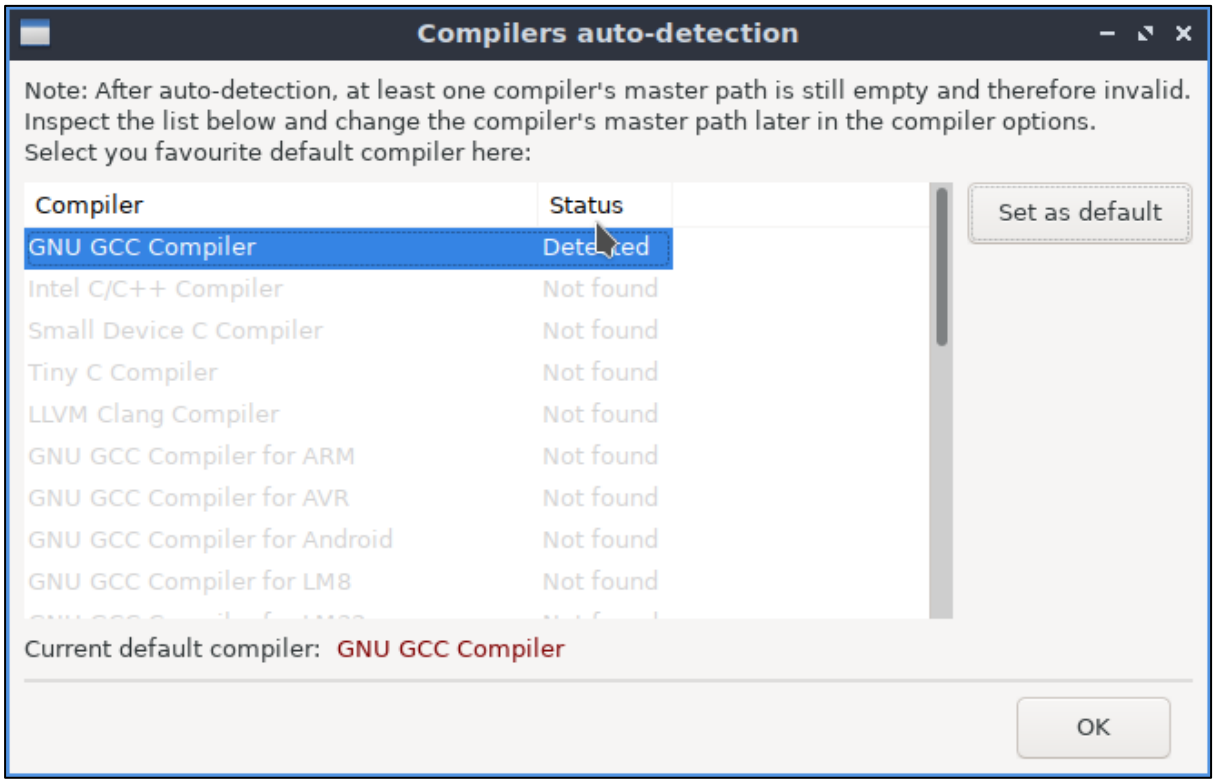
Code::Blocks is the IDE that I am going to use for Linux C C++ development. Copy the following install command into the console.

```
sudo apt install codeblocks
```

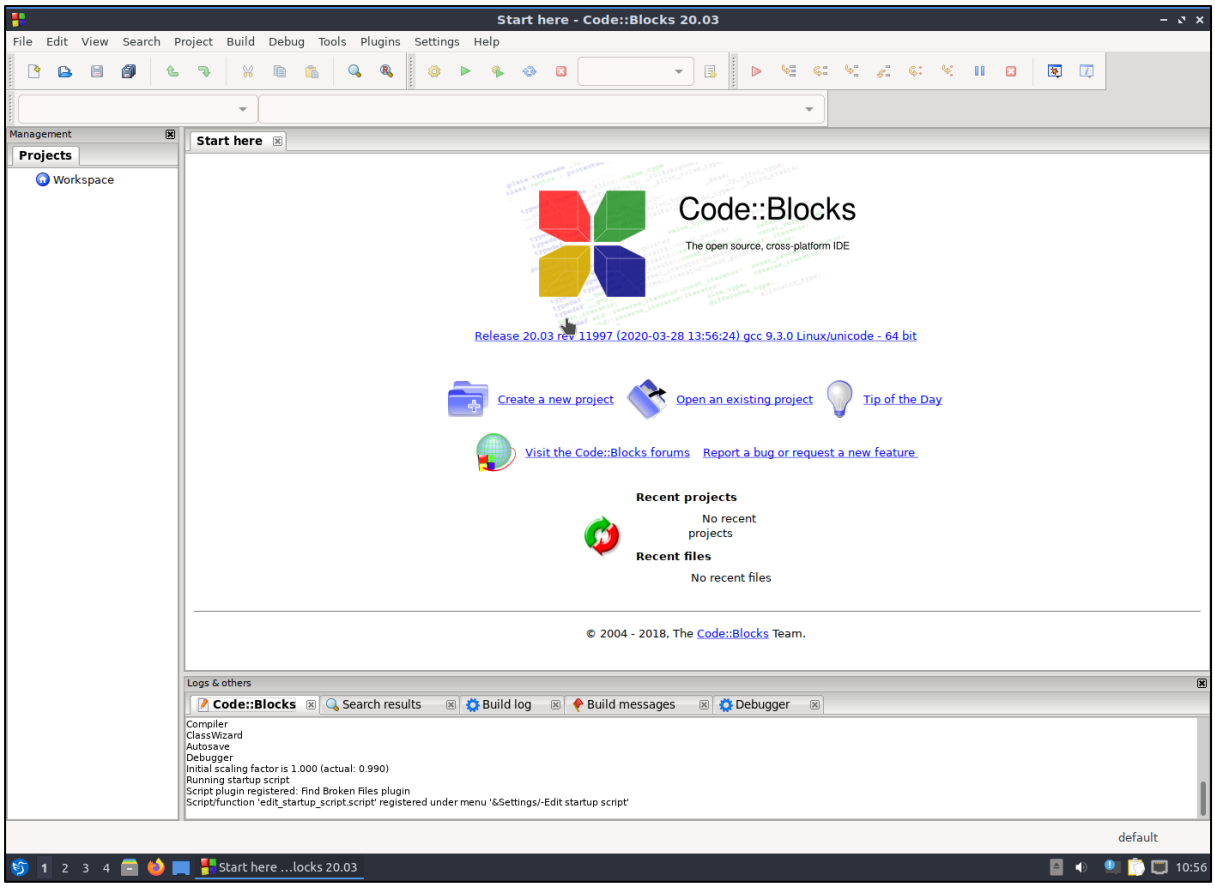


Exit the console window and open the Code::Blocks IDE from the start menu. You will find it under the menu label “Development”.

In the first screen ensure GNU GCC Compiler is selected and then select [Set as default] and then select [OK].



You will now see the Code::Blocks IDE with the default workspace open.



First we are going to go through and make sure the essential settings are correct before opening a project or creating a source code document.

Before I go any further it is important to note the differences between Dev-C++ and Code::Blocks. In Dev-C++ each “New” project will inherit a copy of this “Global” set of compiler options. This means there are 2 sets of compiler options, 1 global and 1 local to the project. Setting the global options makes it a little easier to just automatically “Inherit” the options into a new project, although it is still good practice to check though all of the settings to ensure they are correct for each project. Having the global settings in place also allows you to run individual source files outside of the project in a “Scratch Pad”. Any source that is not imported to the project will fall back to the global compiler settings.

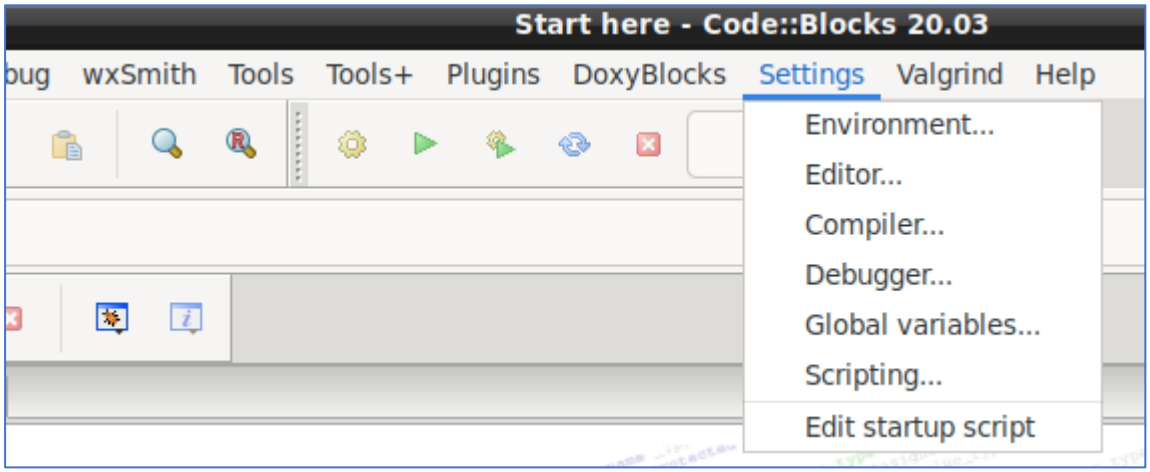
In Dev-C++ the project only uses the compiler setting in the project and there is no risk of conflict with the global settings.

Dev-C++ uses a different system in this regard compared to Code::Blocks. As well as not allowing for code to be compiled outside of a projects, Code::Blocks also appends the Local project compiler setting to the end of the global compiler setting creating a set of duplicate flags set to the compiler. In other words Code::Blocks uses the Global setting plus the Local project settings. For Code::Blocks

it is safer to leave the global setting blank and only set the project compiler options to avoid conflicts. All IDEs have their differences and quirks.

For all of our C exercises in this book I will be static linking to any libraries. The exercises only use the standard library so these will be compiled into the final executable.
In Book 3 - Libraries Overview I will go into more detail about using static and shared libraries.

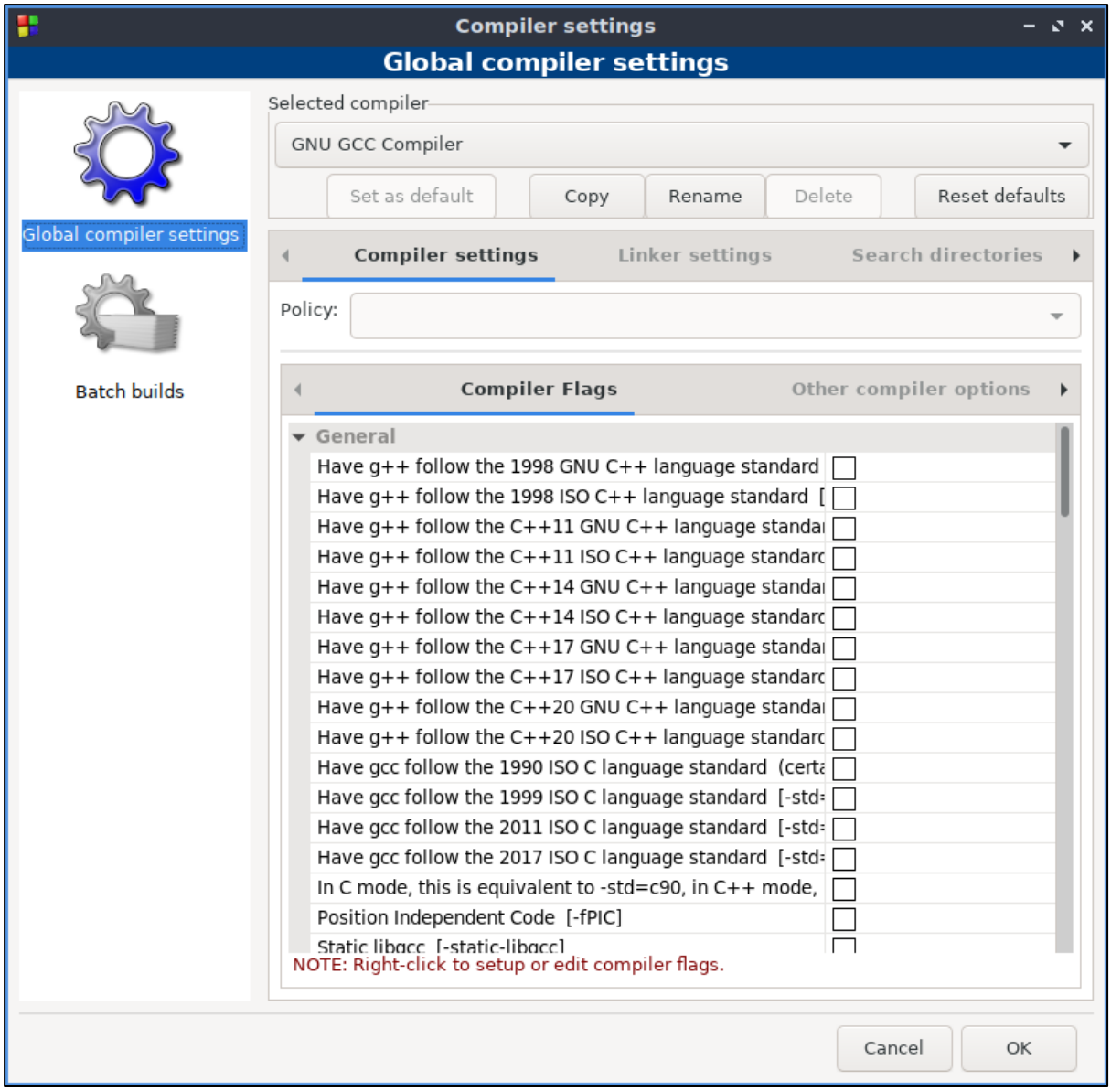
All of the settings that follow can be found in the “Menu Bar” under “Settings”.



From the Menu Bar open the Settings -> “Compiler…”.

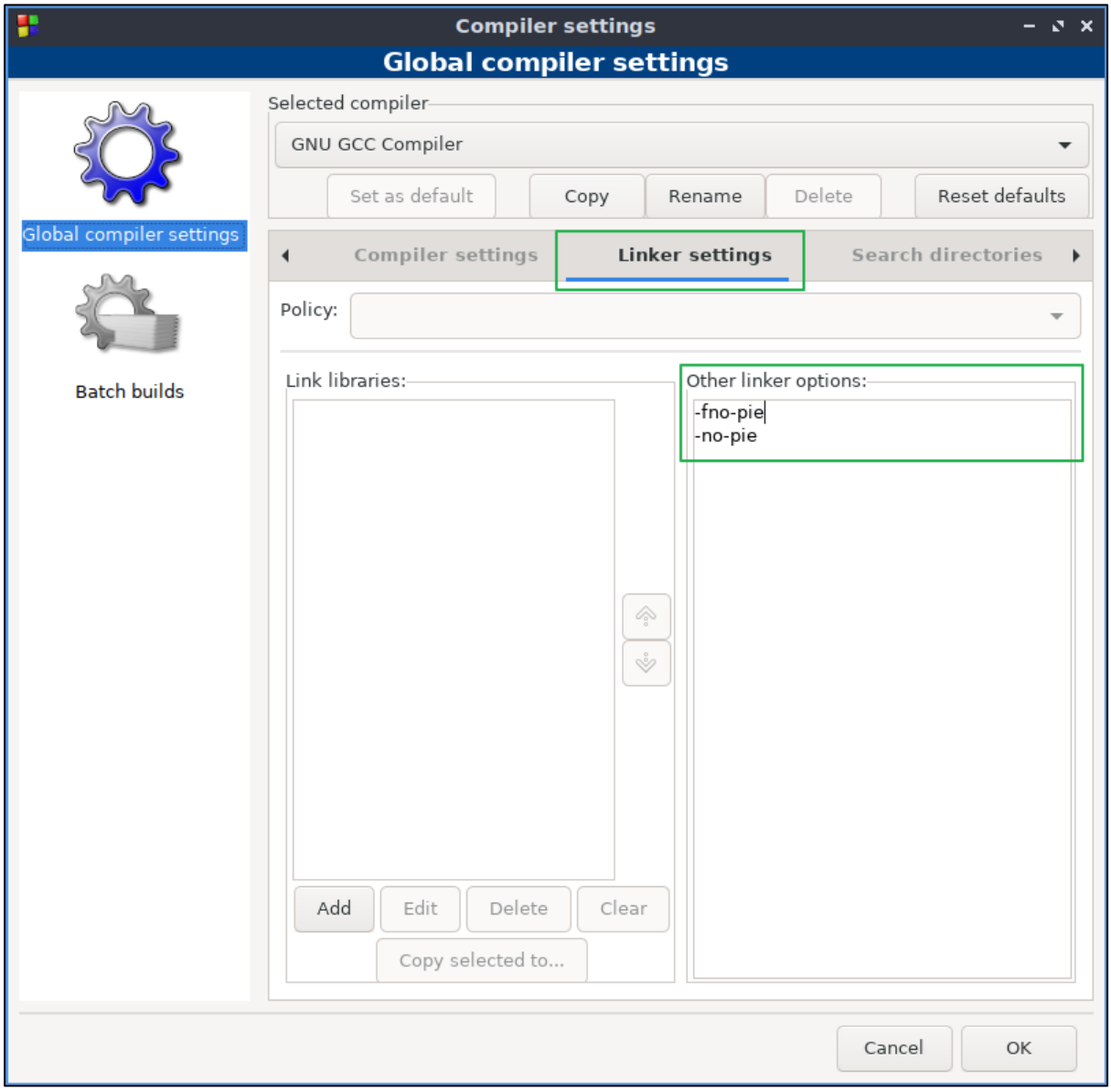
Learning to understand the different compiler settings and flags in GCC can be something of a learning curve so I will give you the basic settings to compile a standard Linux executable.

Ensure that all of the following global compiler settings are unselected. Unlike Dev-C++ we will need to create the correct compiler and linker settings for each profile in the project.



The Compiler is set by default to create position-independent executables (PIE). The output file will appear as a shared object although the file will still be executable. Linux does not use file extensions and compiling a source document that contains **int main(){ …; return 0; }** as a shared object will still be seen as a form of executable file. The source code for Shared Objects or Dynamic Linked Libraries are created with a different entry point and do not use **main()**.
This is part of the Address space layout randomization (ASLR) that has been introduced for security reasons.
“Address space layout randomization (ASLR) is a memory-protection process for operating systems (OSes) that guards against buffer-overflow attacks by randomizing the location where system executables are loaded into memory.”
For our exercises I am going to disable PIE which will allow us to build a standard executable file under Linux. This requires us to add 2 additional switches to the linker -fno-pie -no-pie.
To run compiled executables from the command line you will need to place “./” in front of the executable name. For example “./mytestexe” without the “”.

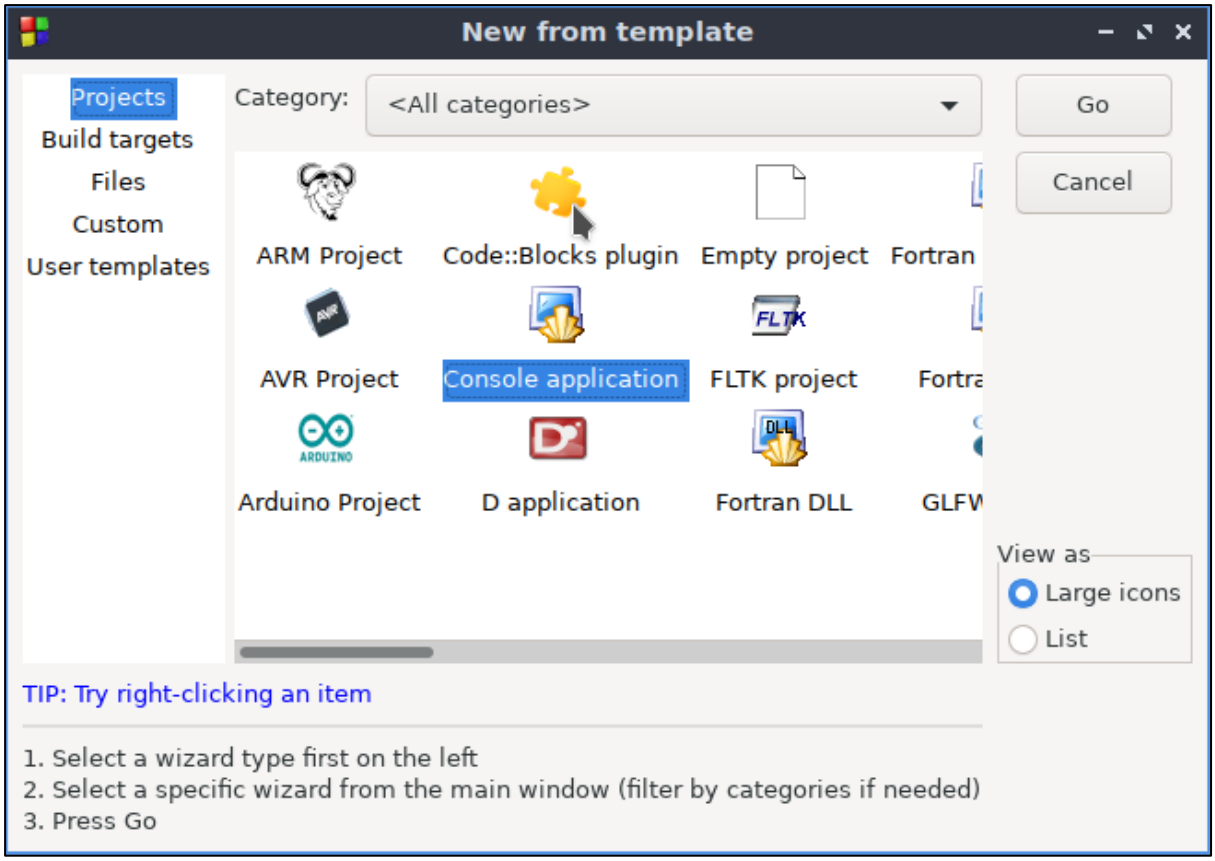
Under the “Linker settings” tab add **-fno-pie** and **-no-pie**. This will result in the output of a standard Linux executable that can be launched from the file manager. This global setting will be prepended to all project settings and will not be seen in the individual project compiler settings.



Select [OK] to save the settings.

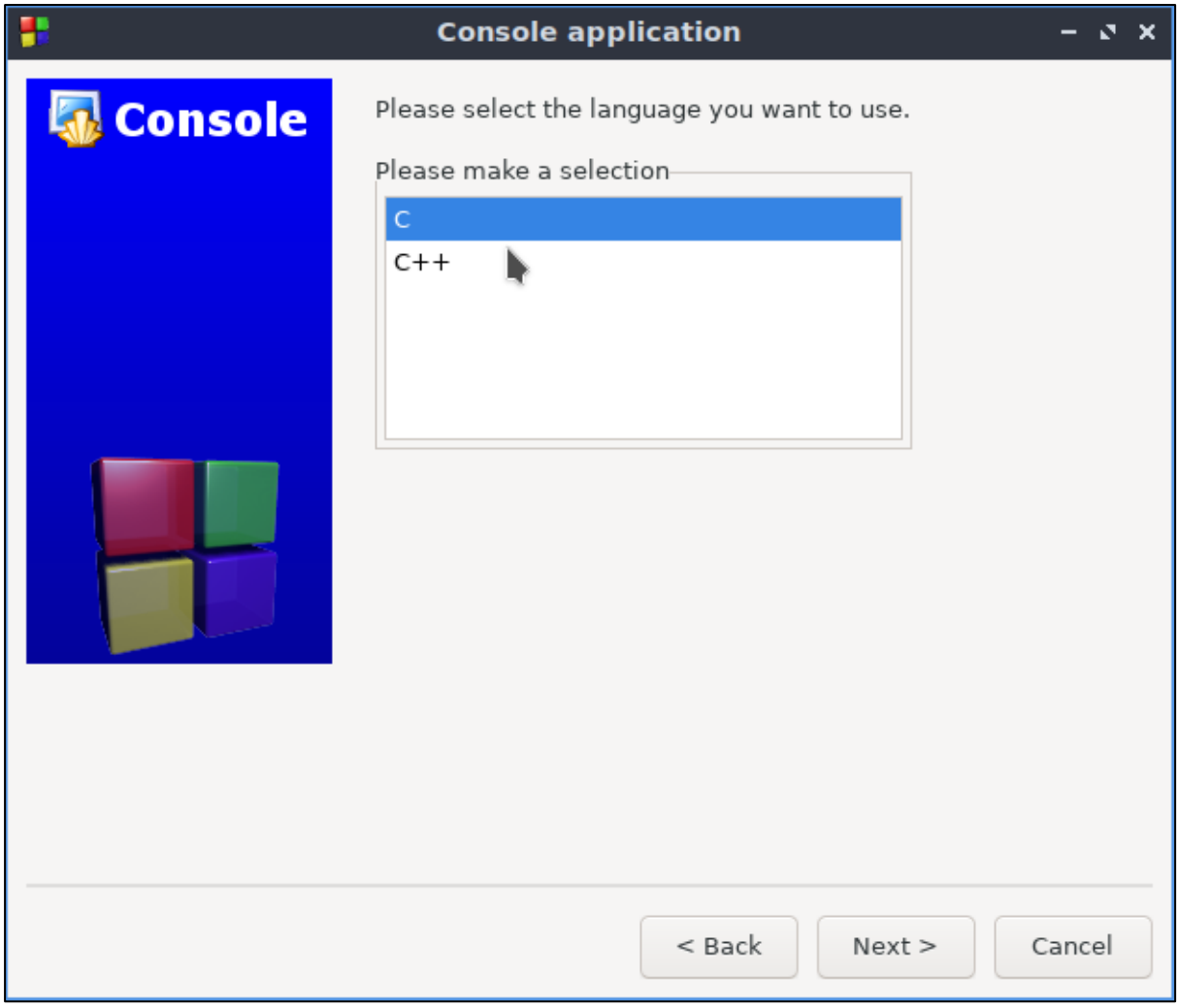
In the open TAB or from the tools menu select File -> New -> Project…

Select “Console application” and then [Go].

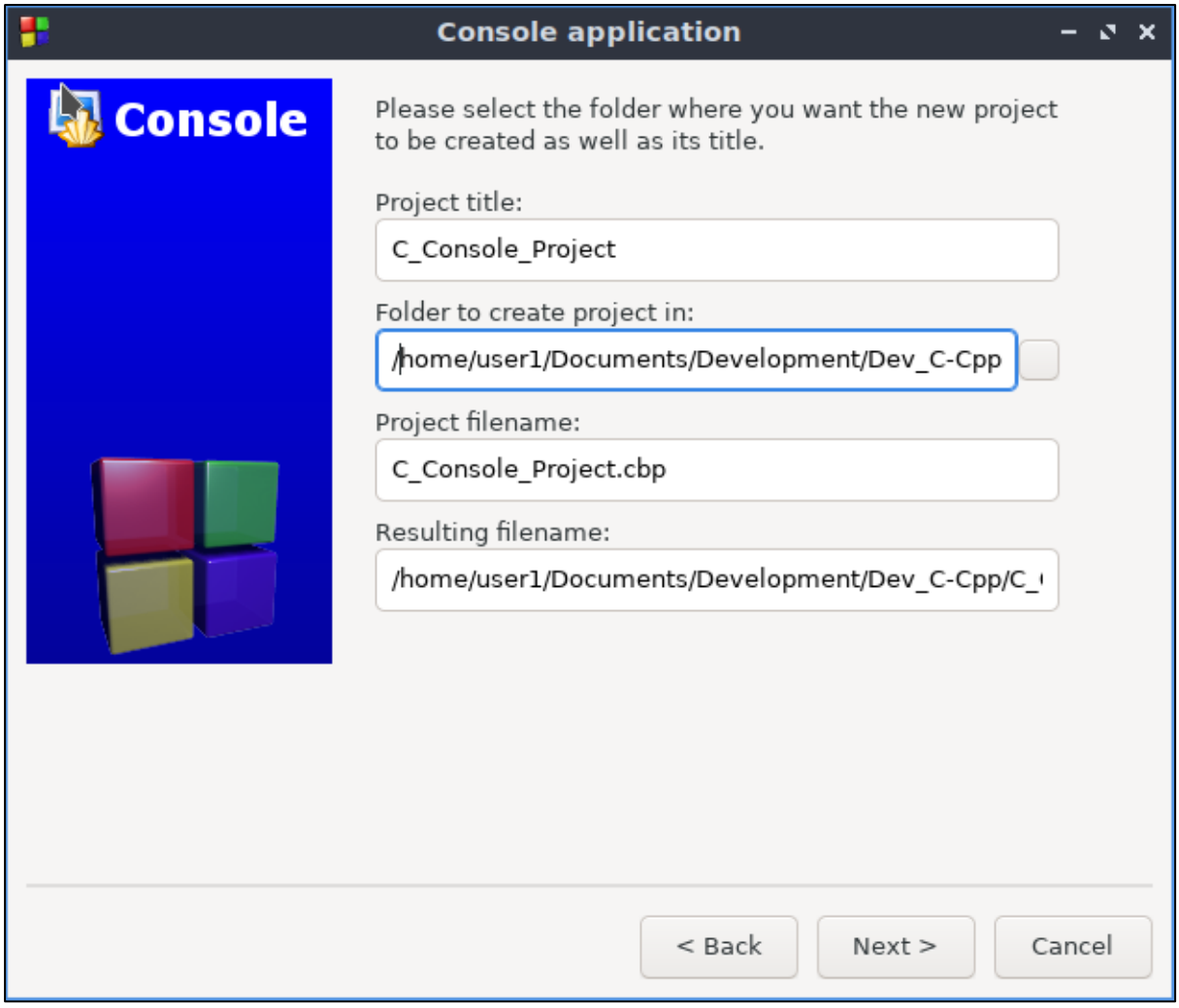


In the next screen select [Next >]

Select the language as C and then select [Next >]

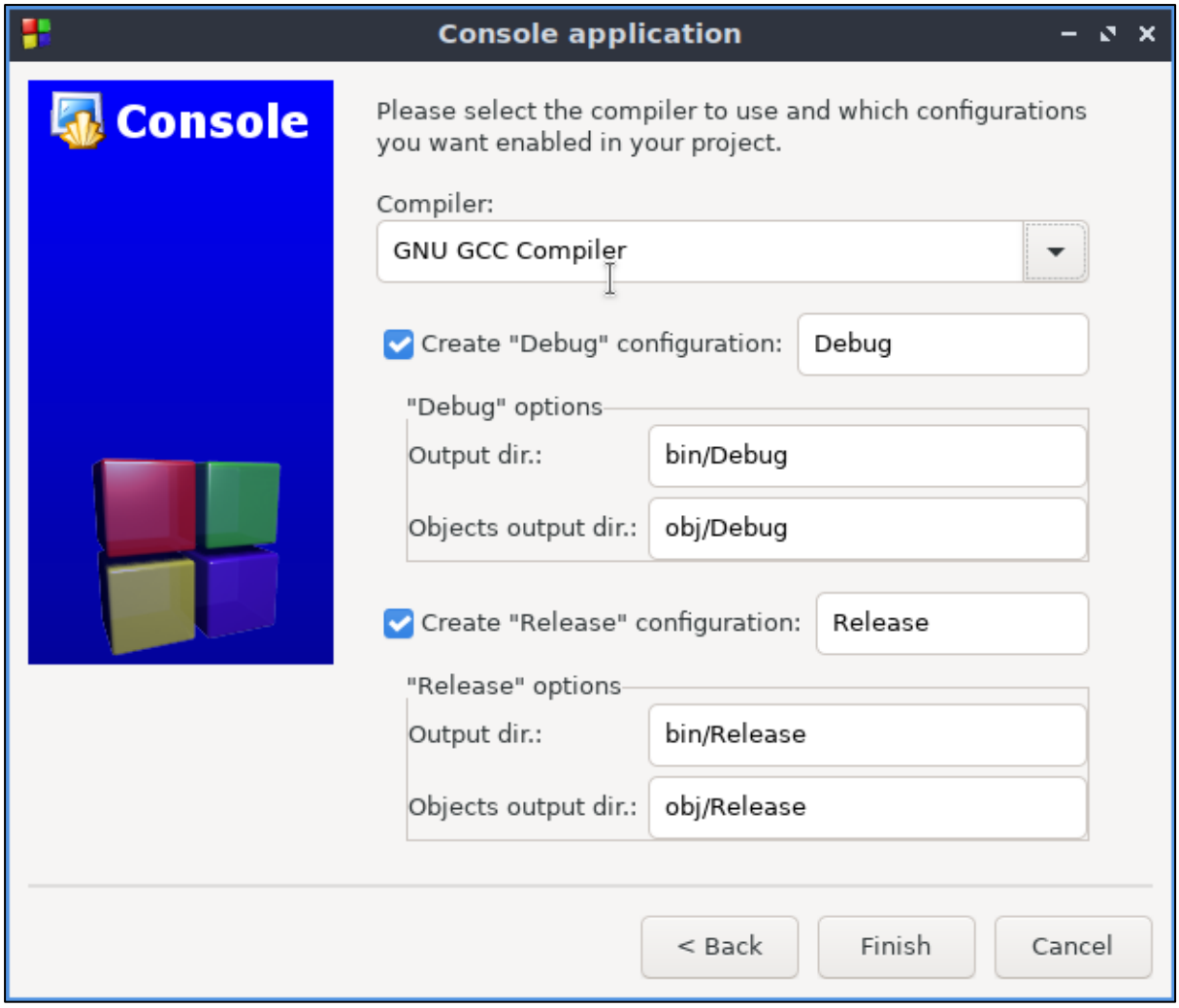


Give your project a “title”. The project “file name” will be automatically generated from the title name. This name will be reflected in the name of the output executable file when compiled, but you can change it to a more suitable name at a later time if you wish.

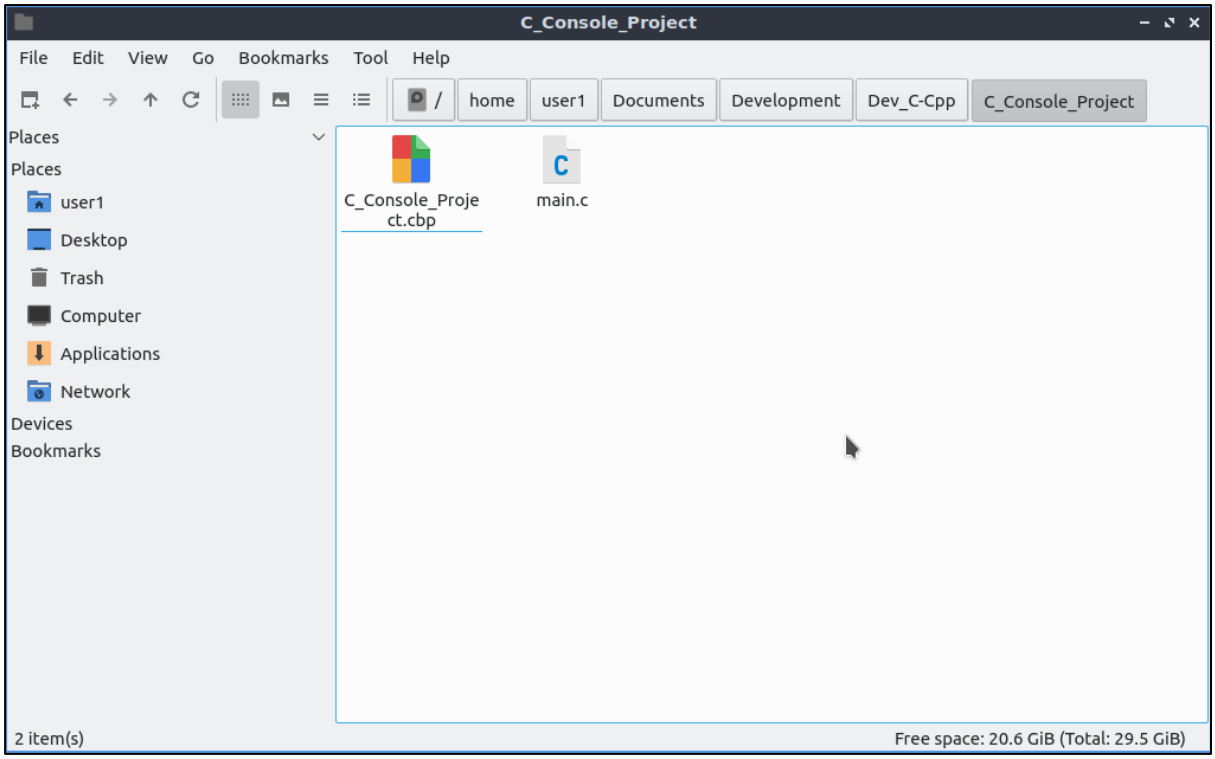


Navigate to the "`/home/username/Documents/Development/Dev_C_Cpp`" folder that we created earlier and select that as the path for the project. Select [Next >].

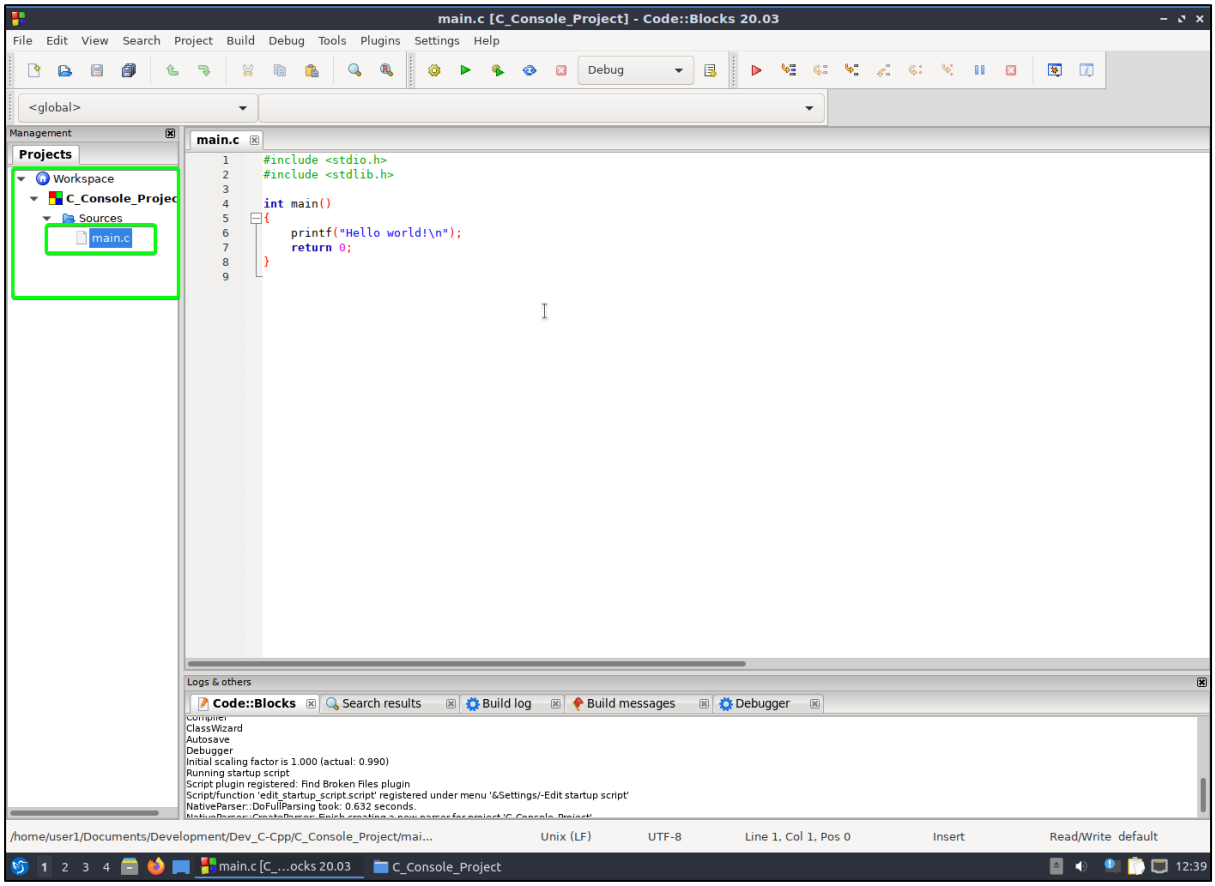
Leave the default setting for the next screen and select [Next >]. This will create 2 subfolders where your "debug" executable and "release" executable will be created. In practice the "Release" executable would be stripped of all debugging information as well as optimised for size and speed of execution.



Select [Finish] to create the project and project files. If you open the file manager and navigate to the **../Development/Dev_C-Cpp** directory you will now see the project folder as well as the **Code::Block project file .cbp** and a **main.c** source file.



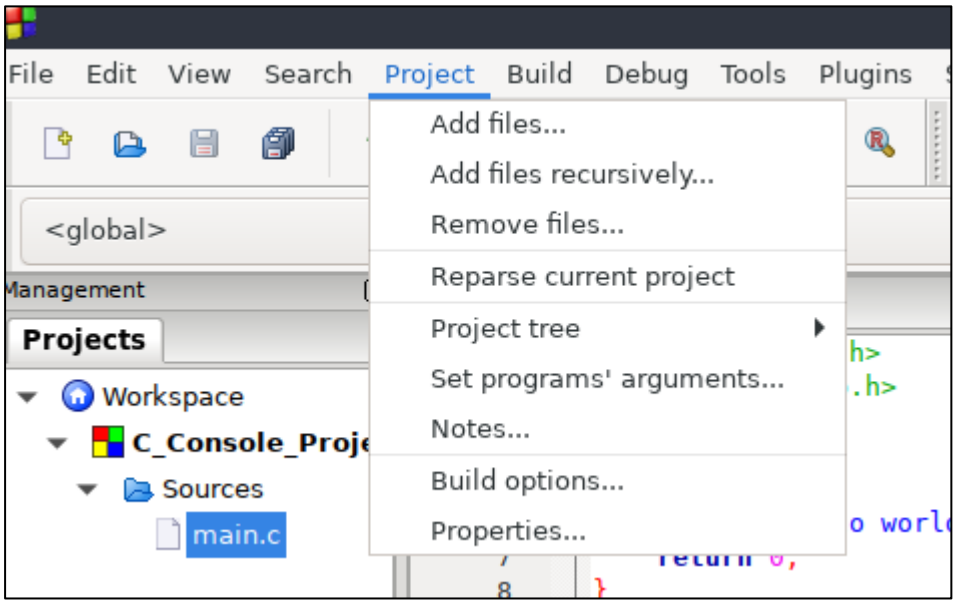
Return to your Code::Blocks Workspace and open the Sources folder in the projects menu on the left. Double click on the **main.c** file to open it in the IDE editor TAB.



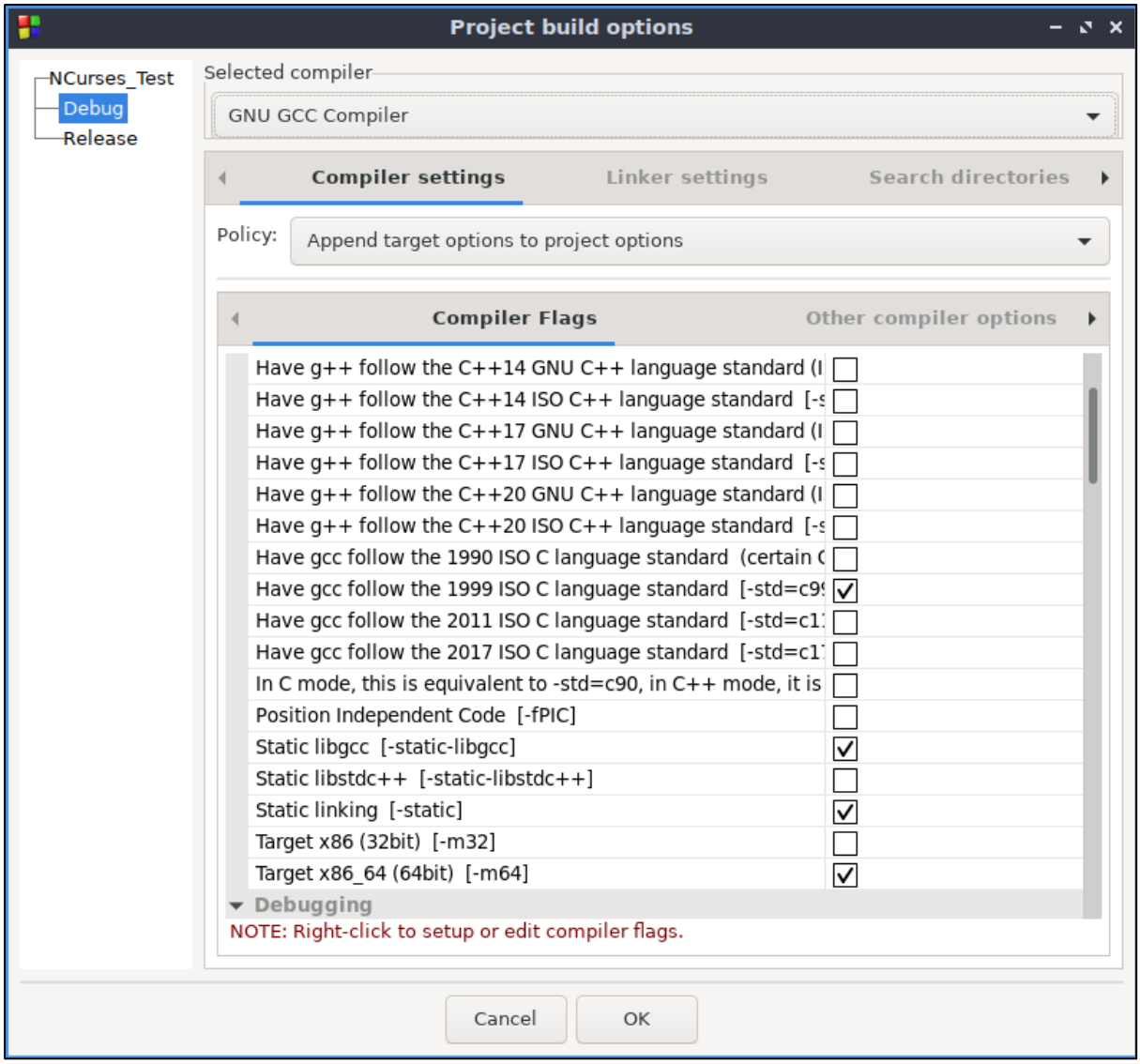
Code:Blocks has created a default Hello world! source file to get started with.

In the main tool menu select “Project” -> “Build options…”

Unlike Dev-C++ you will need to manually set the correct flags for the Debug” and Release” profiles every time you create a new project in Code:Blocks.



Select the Debug option on the left and tick the following compiler flags.



1999 ISO C [-std=C99] is the highest standard available that is compatible with the MSVCRT.DLL. All source code that you write on Windows with GCC and MinGw will follow this standard, so I am setting it the same so that your code can be easily cross compiled between Windows and Linux. Windows 10 onwards can use the (UCRT Universal C Runtime). This is a different compiler and library in Windows, and I am not yet certain of the Linux compatibilities.

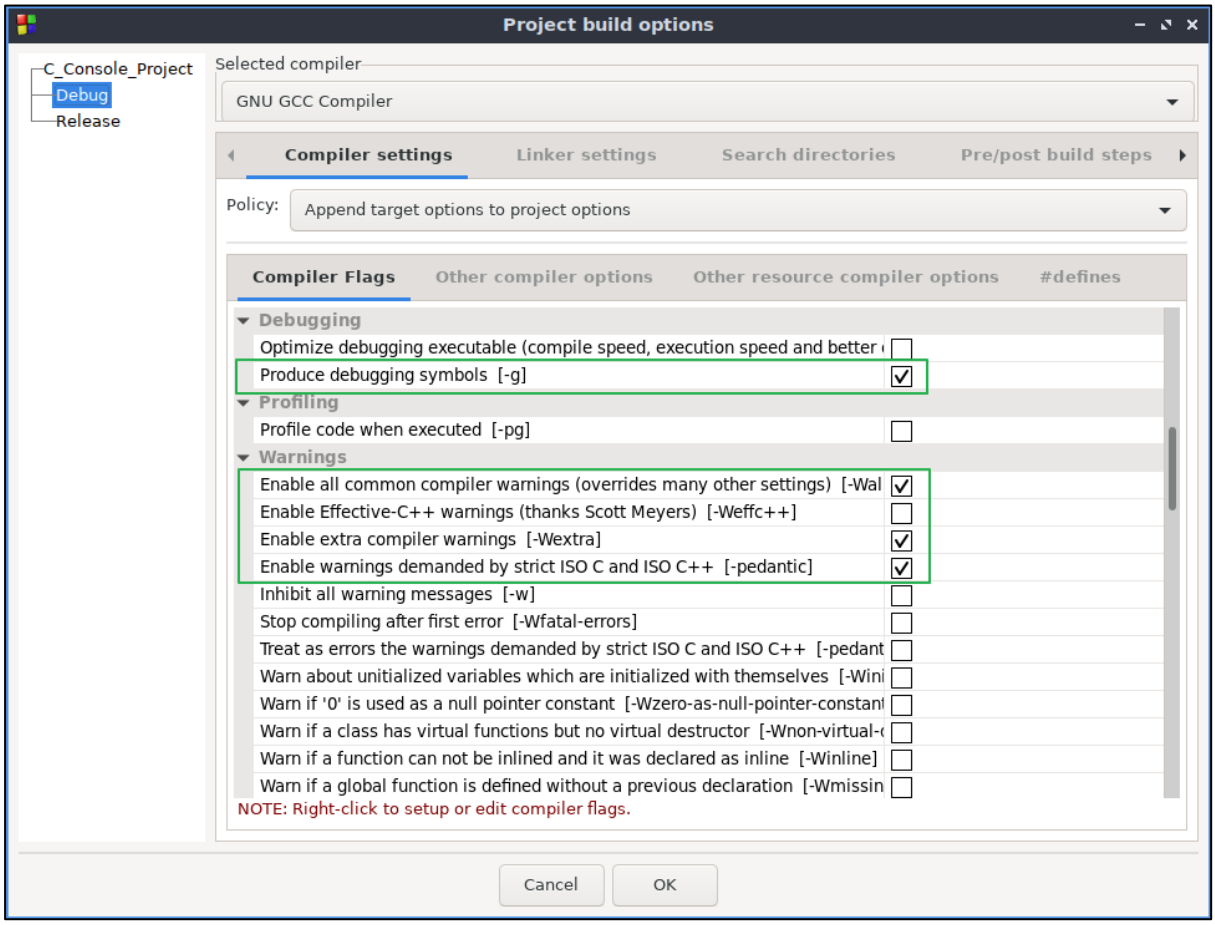
[-static-libgcc] will compile the standard C libraries into the final executable so that you don't have to transport the Runtime libraries with your project. In most instances the Shared objects may already be installed on the target system.

[-static] Is much the same as above and will force most additional libraries to be included in the final executable. This will fail on systems that only have a shared object ".so"available and no library archive .a. Again, this just means that you may need to install those shared libraries on the target computer.
If the -static flag is set then tinfo (libtinfo) will need to be added to the "Link libraries". This file stores the location information for static libraries on Linux systems.

[-m64] will select the pointer width for the target platform. All modern Linux and Windows are

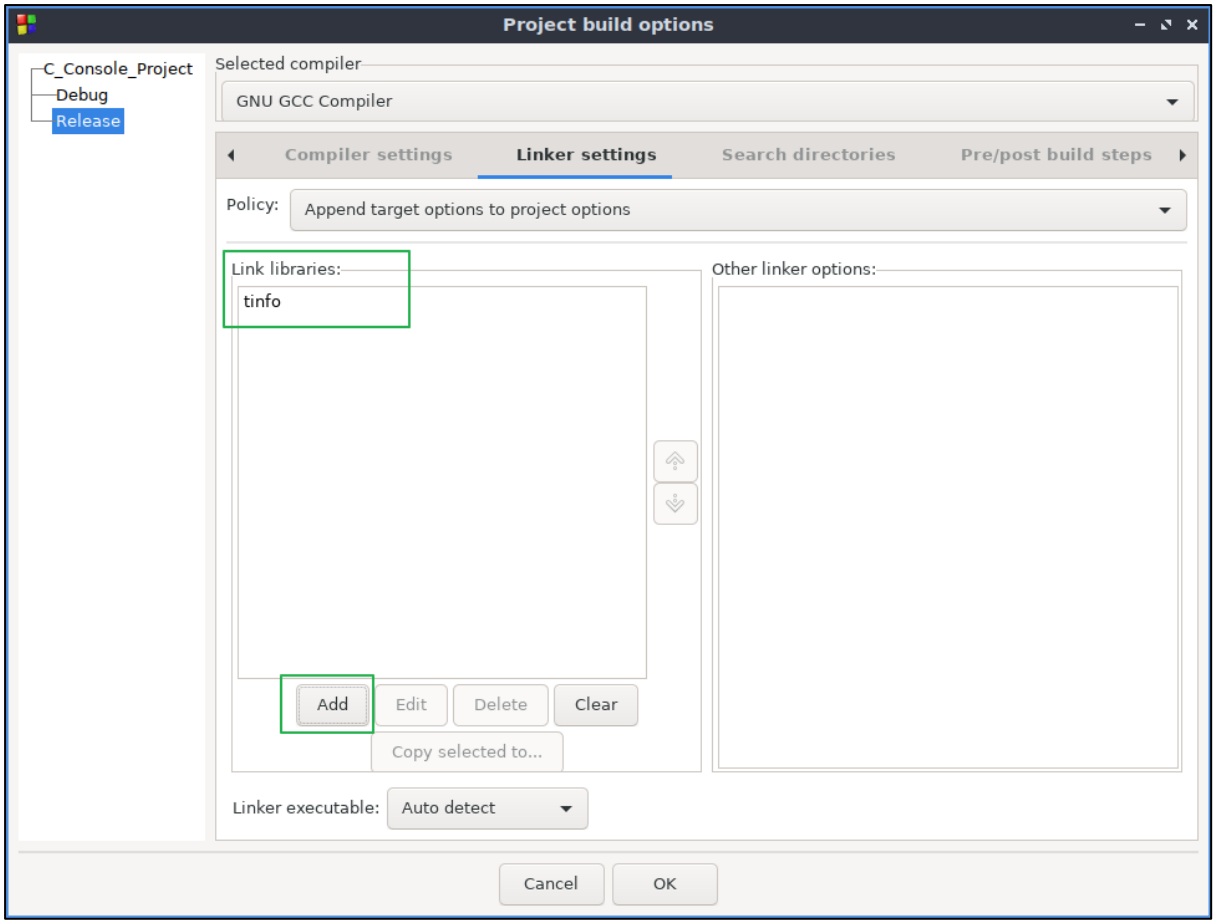
usually 64 bit.

The following 4 settings will give very pedantic warnings about coding syntax mistakes as well as holding the coder to strict coding standards. Not all warnings will mean that your code is broken or not able to run, but are just warnings that sometime undefined behaviour or runtime faults can occur.

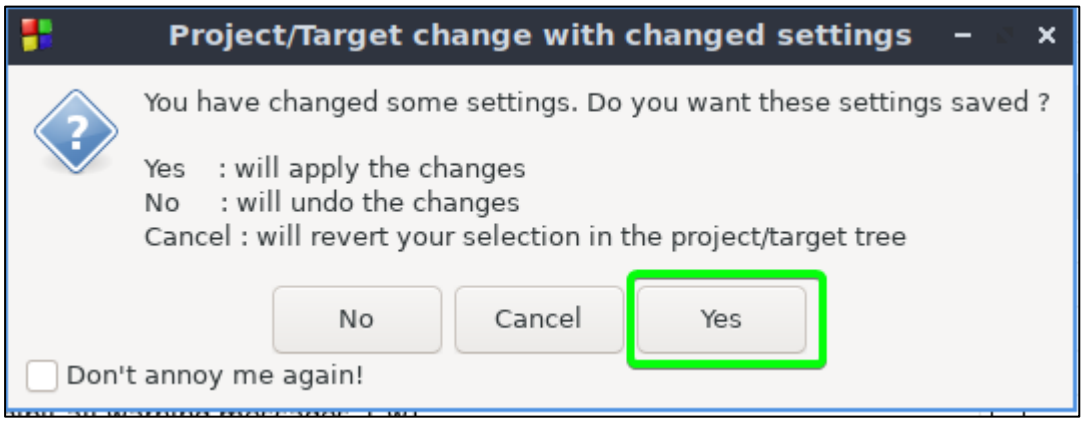


Next switch to the “Linker settings” Tab.

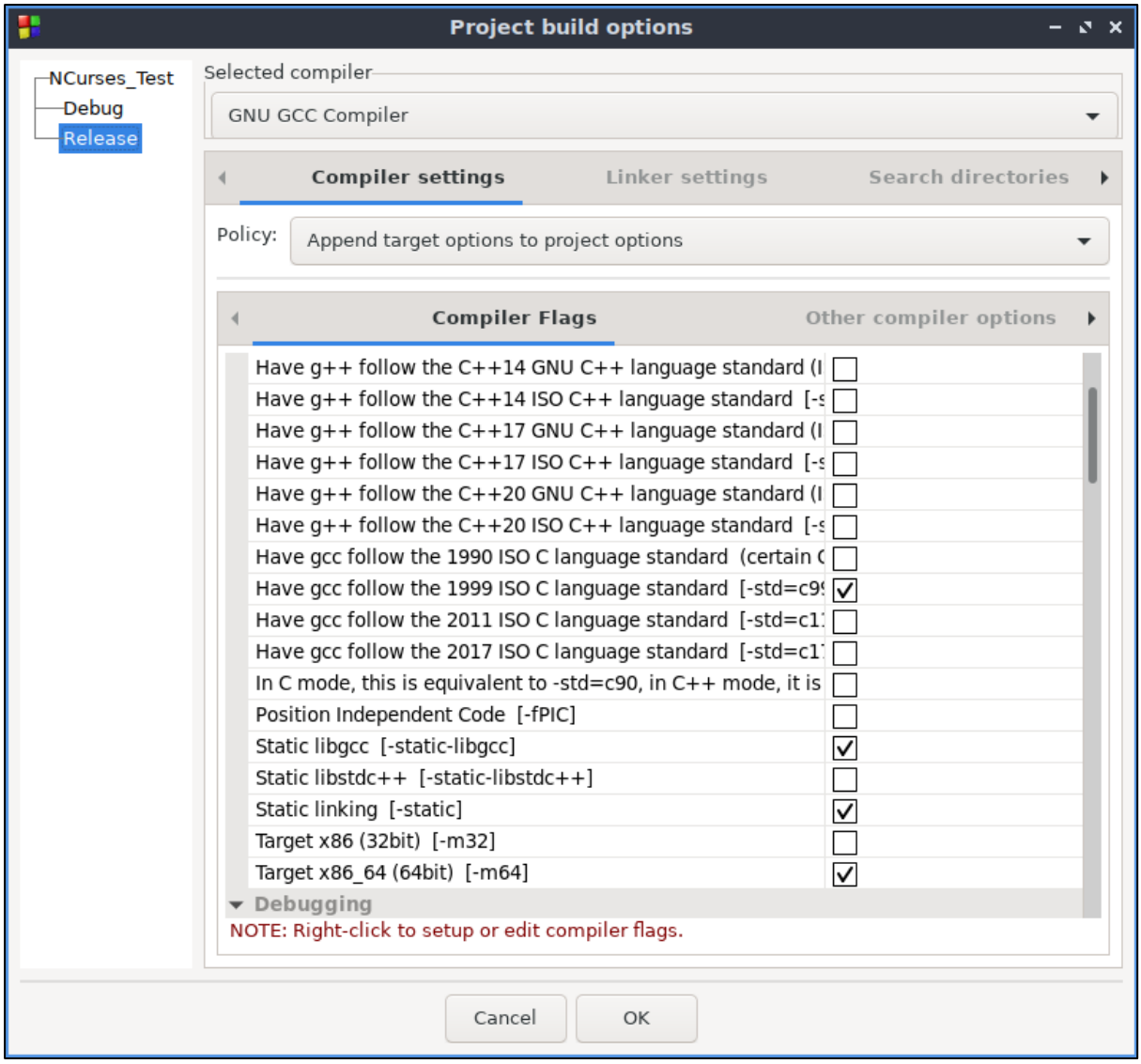
Because we are using -static we must include tinfo under Link libraries. Linux needs this to find the information for static libraries lib.a.



Next select the Release option on the left panel and select [Yes] to save the changes.

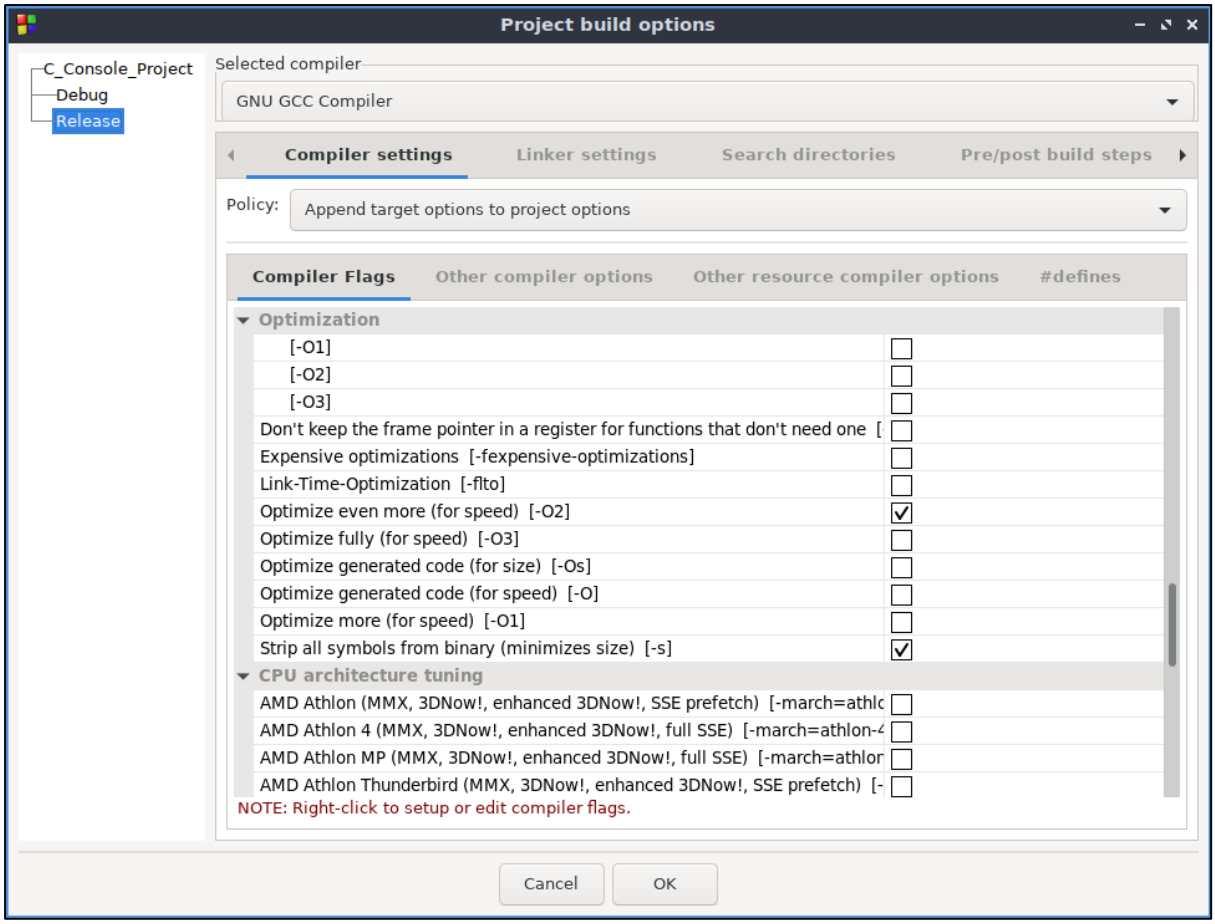


Set the following options for Release.



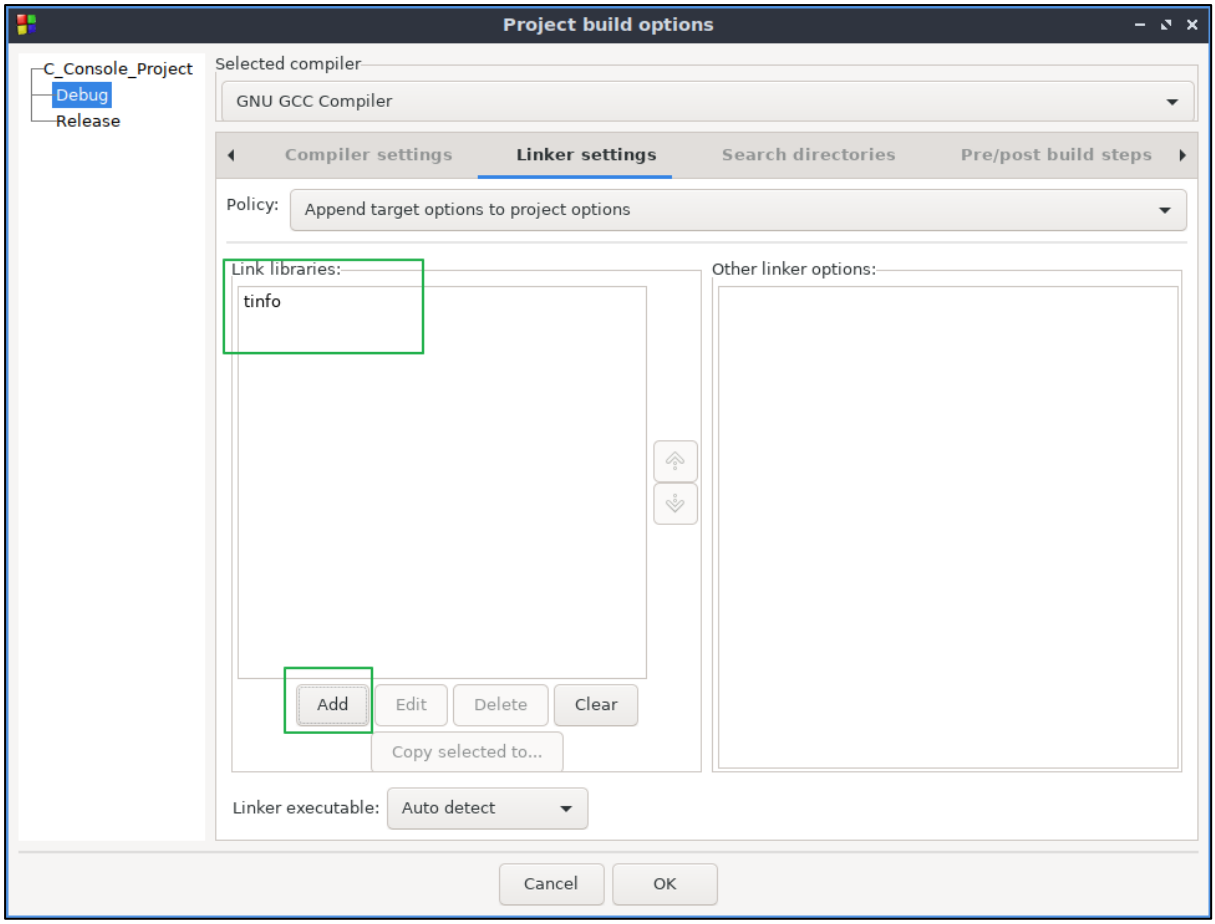
All other options can be left with the default values accept for [-O2, [-s].

You will notice that [-O2, [-s] optimizations have been selected. This offers a good balance of speed, size and stability for your final Release executable.



Next switch to the “Linker settings” Tab.

Because we are using -static we must include tinfo under Link libraries. Linux needs this to find the information for static libraries lib.a.

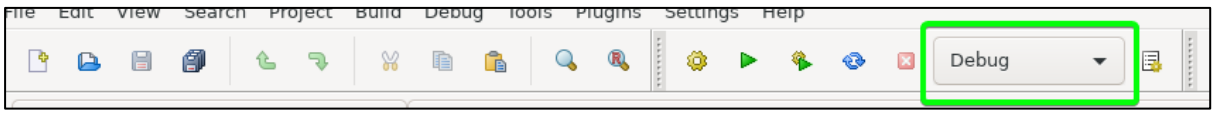


Select [OK] to close and save the project compiler settings.

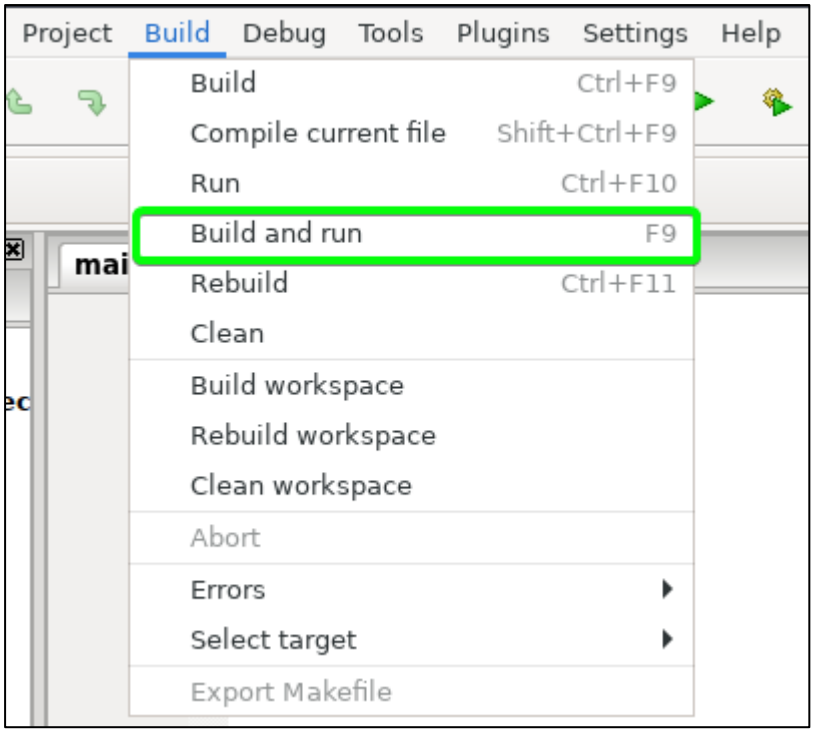
It is important to check and set these settings for each project that you create. If you find that you are not getting an output executable as you would expect, refer back to these settings.

If you read up on the GCC compiler documents you can find custom compiler flags to target specific CPU architectures and hardware, but you will find the settings I have described above are sufficient for most beginner programmers.

In the ToolBar you will see a drop down menu for Debug/Release. Make sure that is set to "**Debug**" as this is where you will spend most of your time learning and coding. The Release option is only for when you are completely sure the application is bug free and wish to offer a copy to other users.

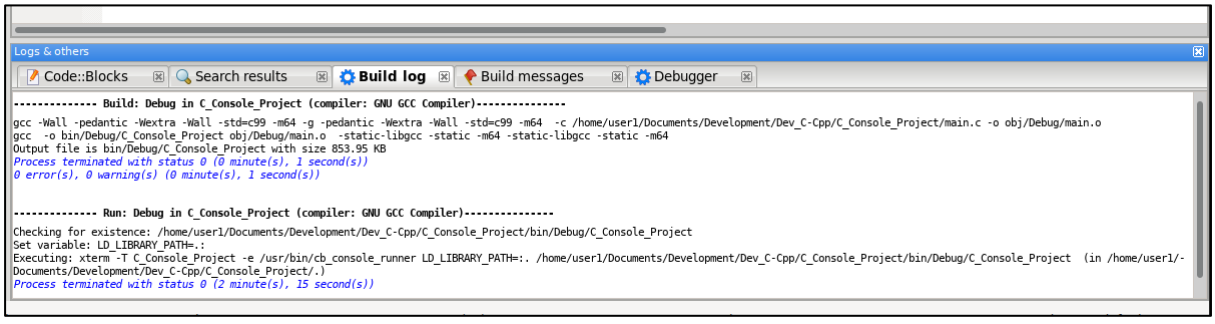


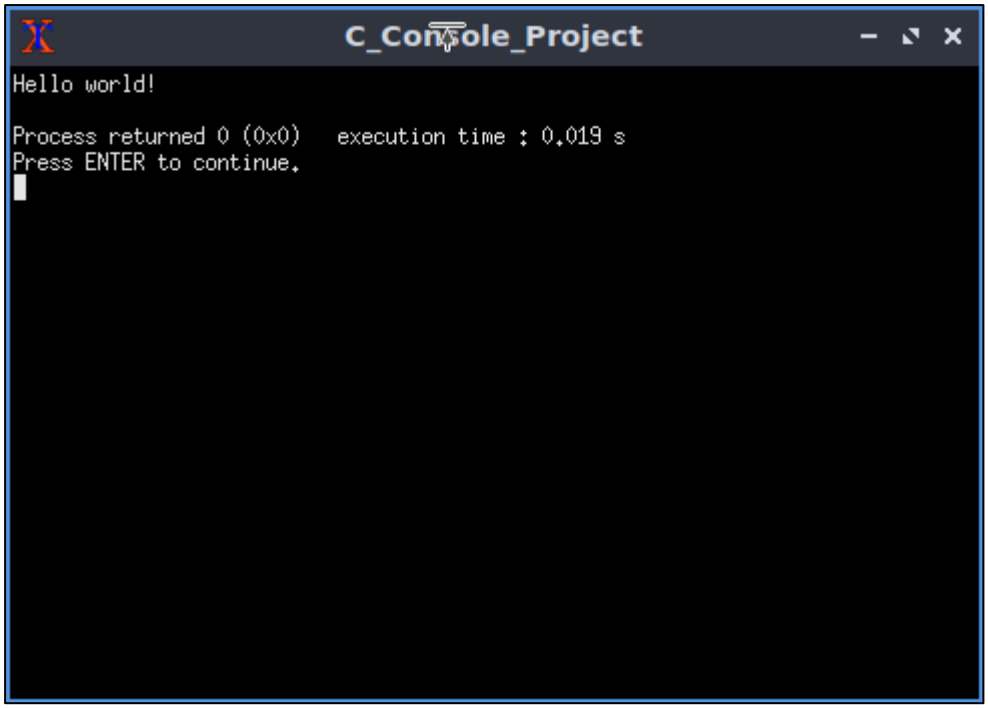
Next select "Build and run".



Or

If everything has worked correctly you should see the build log working through the build stages and then a console window output showing your “Hello world!”





The steps in the build are precompile, compile, Link, create source object (binary) and then assemble to an executable.

You can just select Run to execute the application again without recompiling.

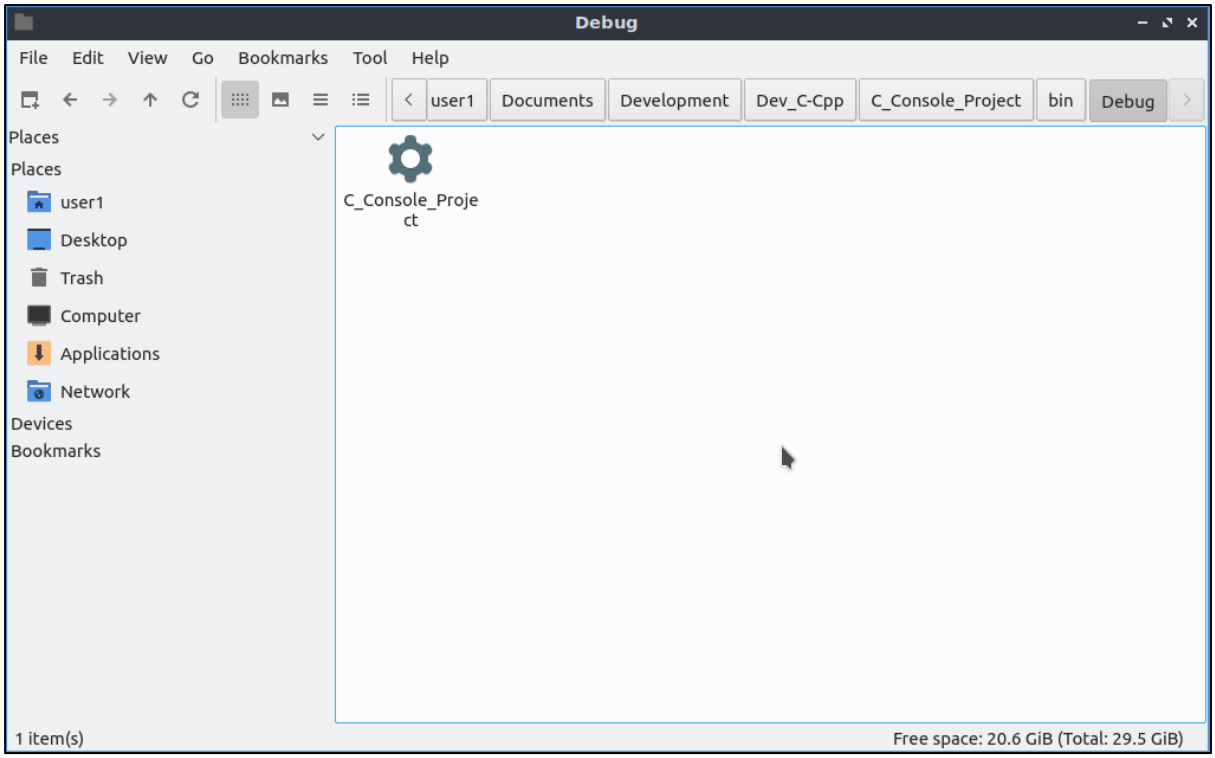
If your application has numerous bugs that you are attempting to find you can just use compile Build to update the debug screen after changes.

Clean and rebuild Workspace may be required to clean up some of the temporary build files if you make significant changes to the project, for example deleting or adding another source document.

Code::blocks does not easily allow you to compile an executable from source outside of a project. Rather than create a new project for each practice exercise you can rename main.c to an alternative name and save it. You can have many alternative source files in the project folder as long as they have unique names and there is only one main.c showing in the project, or one source file of any name with **int main() { …; return 0; }** inside of the IDE project file list.

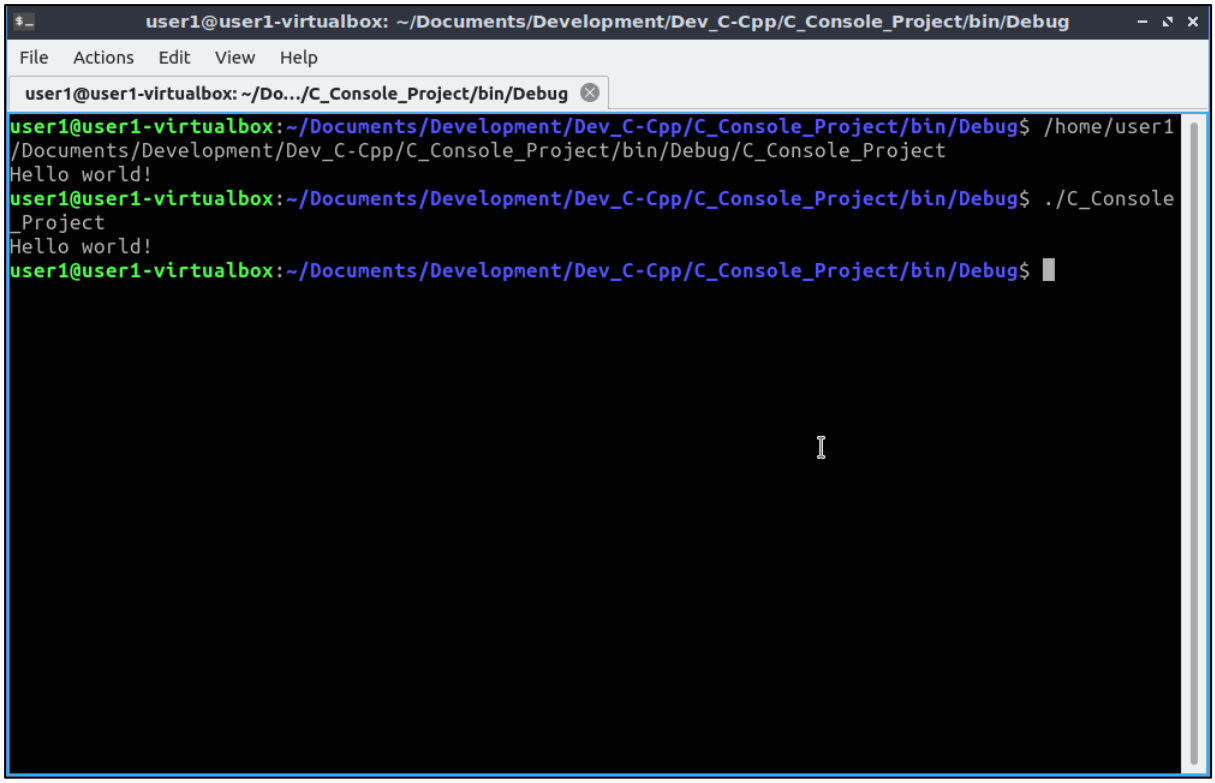
Last, return to your file manager at **/home/user1/Documents/Development/Dev_C-Cpp/C_Console_Project**. You will now see two new folders; /obj and /bin.

Navigate to **/home/user1/Documents/Development/Dev_C-Cpp/C_Console_Project/bin/Debug** and you can find your output executable "C_Console_Project".

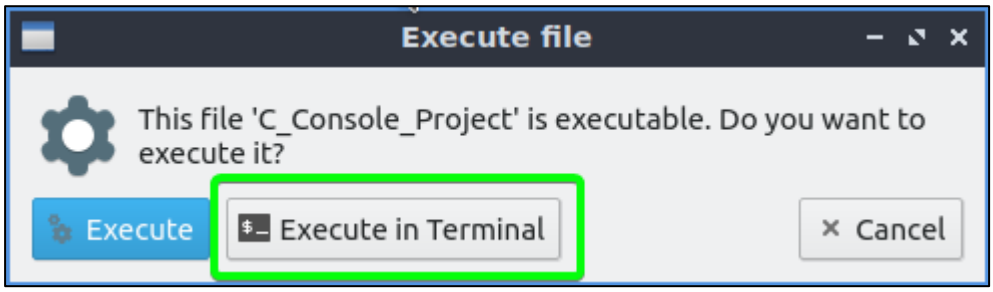


Open a console window at the directory of your source file and enter the name of the executable as **`./C_Console_Project`** followed by [Enter].

You will now see your application print "Hello world!" to the console. Because this is a console application you will either need to place to full pathname to the application to execute the file "**`/home/user1/Documents/Development/Dev_C-Cpp/C_Console_Project/bin/Debug/C_Console_Project`**" or place the **`./`** in front of the file name as "`./C_Console_Project`". If you don't do this Linux cannot find the path to the console executable file.



If you place a pause line at the end of your application you can open our output executable directly from the file manager by double clicking on it and selecting “Execute in Terminal”.



Have a try at some of the examples that I have provided in “A Beginners Guide To Programming” and have fun learning to code.

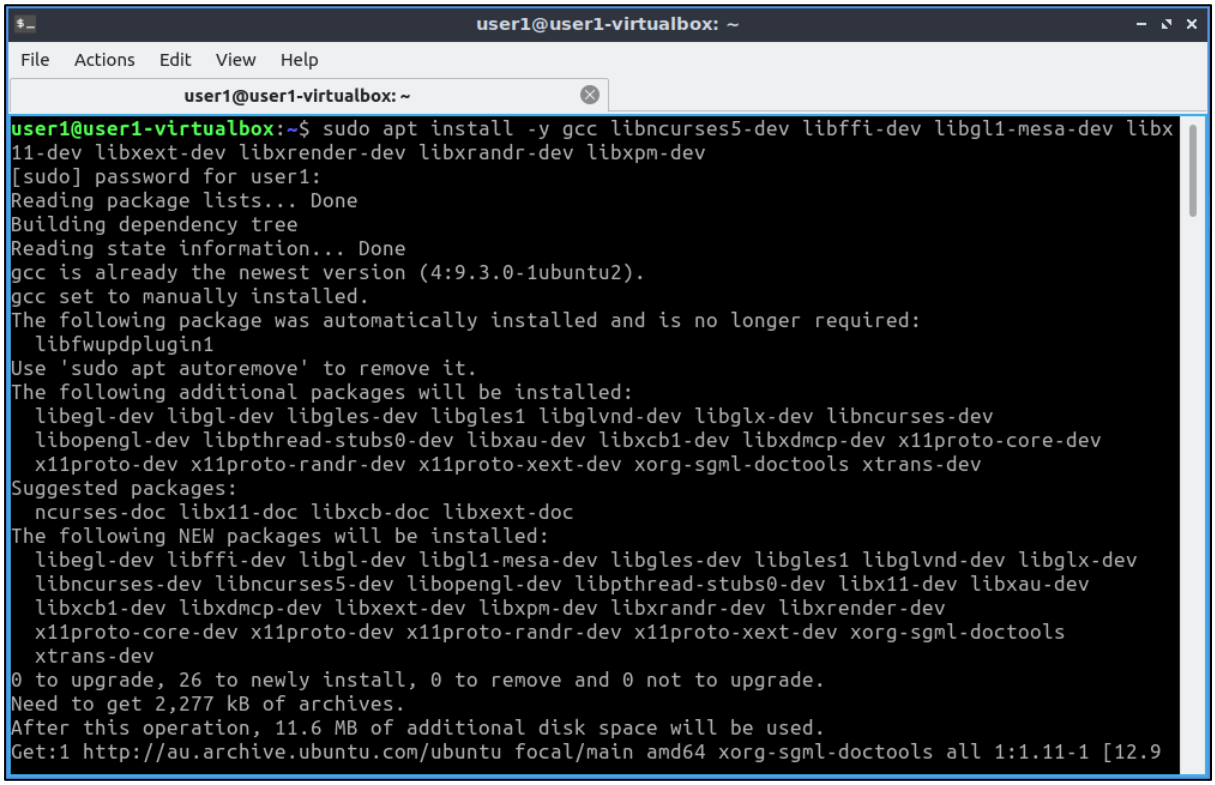
Install FreeBASIC development tools

FreeBASIC Compiler

The following will install the FreeBASIC Compiler (FBC) and the additional support libraries required. FBC will need to be downloaded and installed manually.

Open a command terminal and enter, or copy/paste, the entire install line into the terminal and press [Enter] to continue. This will install the required support library modules.

```
sudo apt install -y gcc libncurses5-dev libffi-dev libgl1-mesa-dev libx11-dev libxext-dev libxrender-dev libxrandr-dev libxpm-dev
```

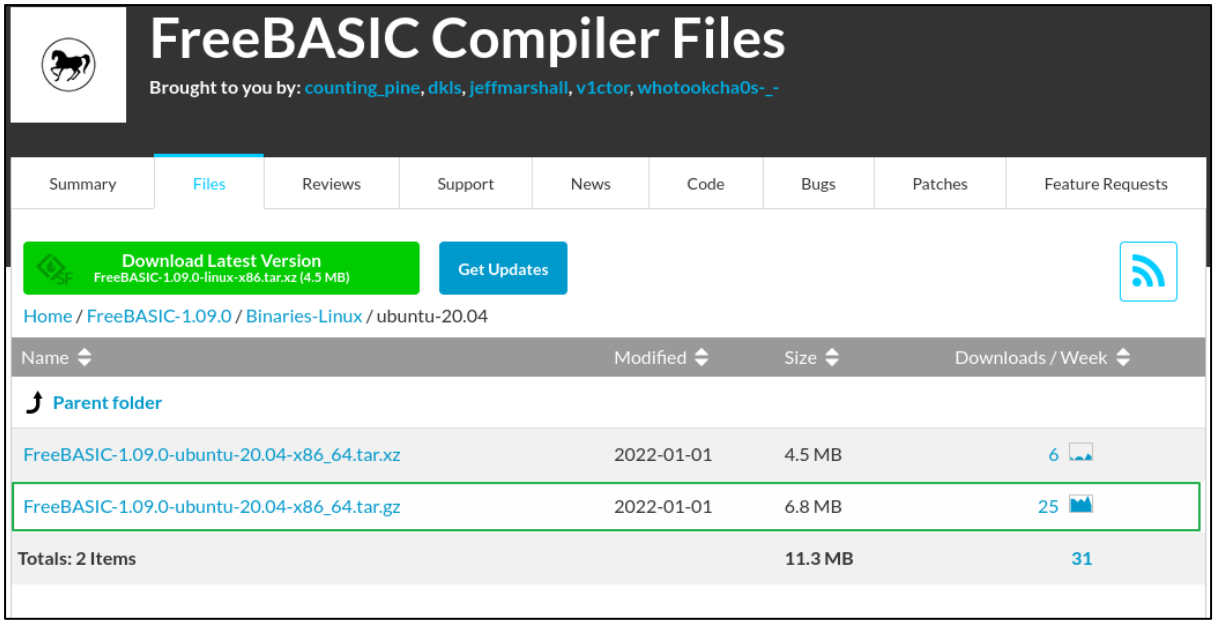


If you have not already downloaded the FBC installer, download it now…

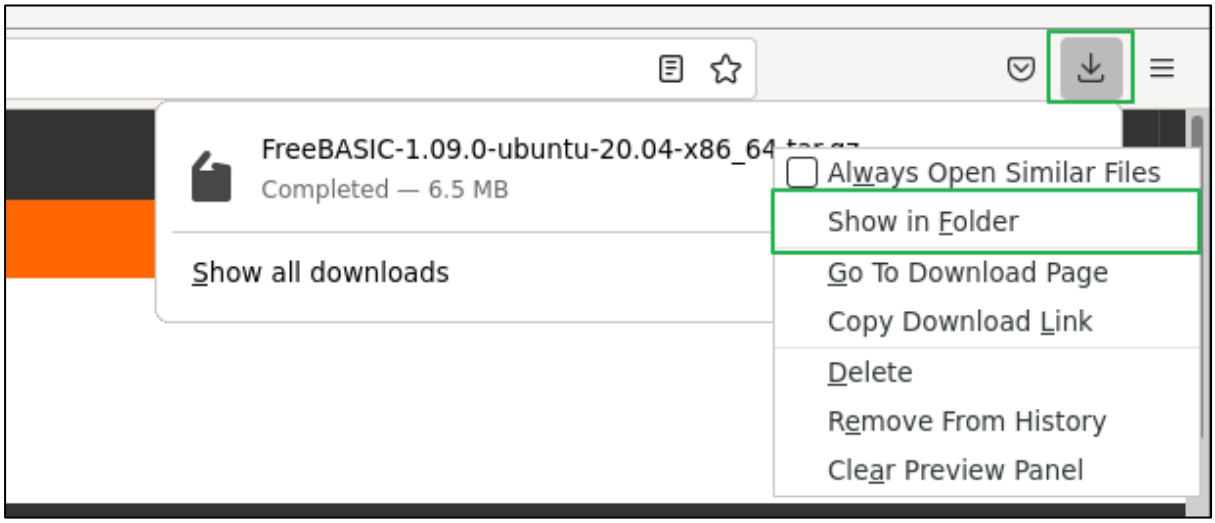
https://sourceforge.net/projects/fbc/files/FreeBASIC-1.09.0/Binaries-Linux/
Download for your Ubuntu version “Ubuntu 20.04.4 LTS (Focal Fossa)”.
“ubuntu-20.04/ FreeBASIC-1.09.0-ubuntu-20.04-x86_64.tar.gz”

Update: A new version of FBC is available for Lubuntu 22.04 LTS

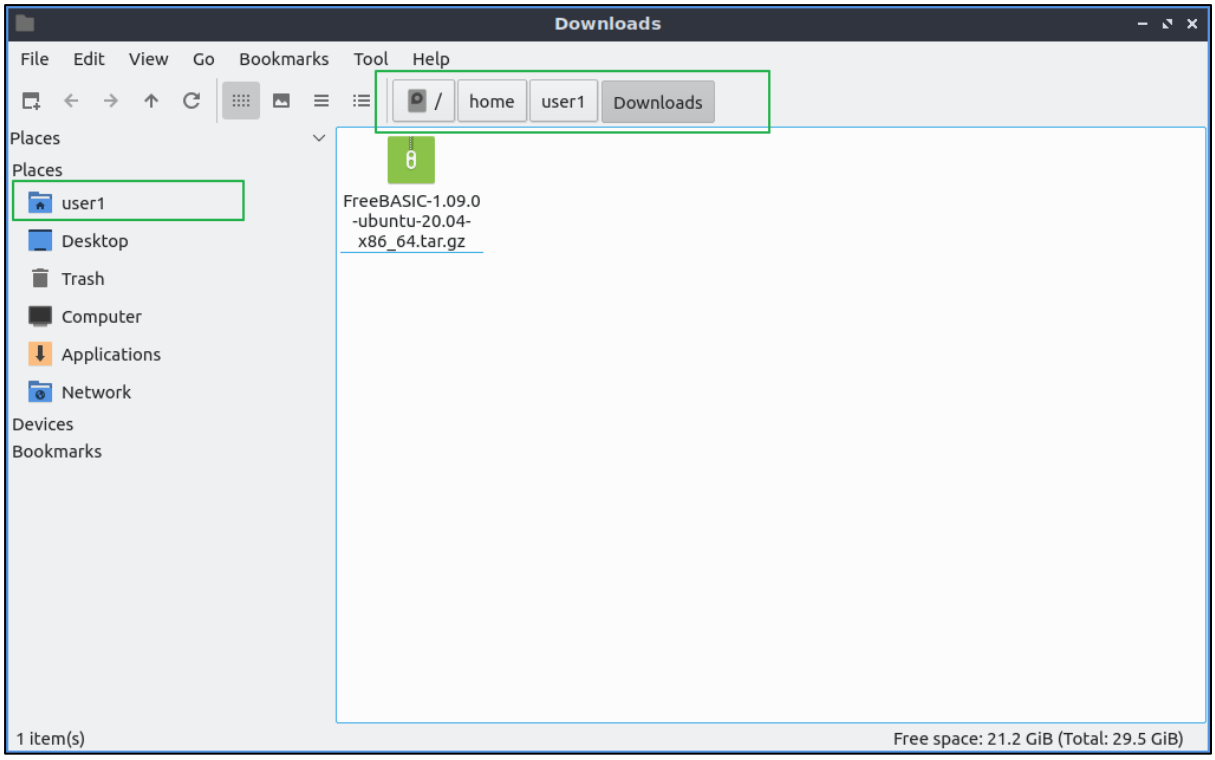
https://sourceforge.net/projects/fbc/files/FreeBASIC-1.10.0/Binaries-Linux/
“FreeBASIC-1.10.0-ubuntu-22.04-x86_64.tar.gz”



Open the download folder location.

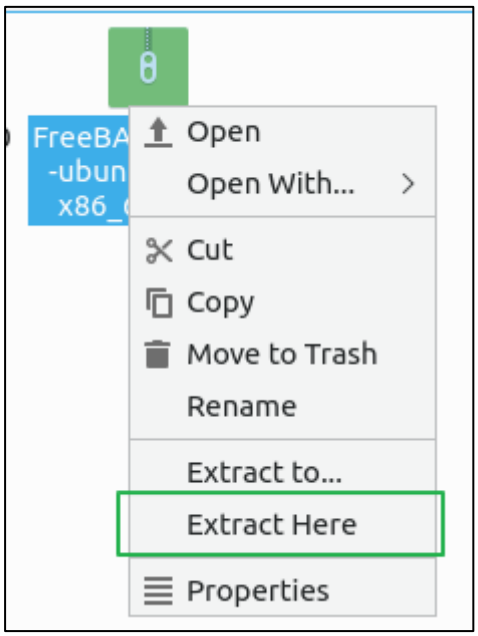


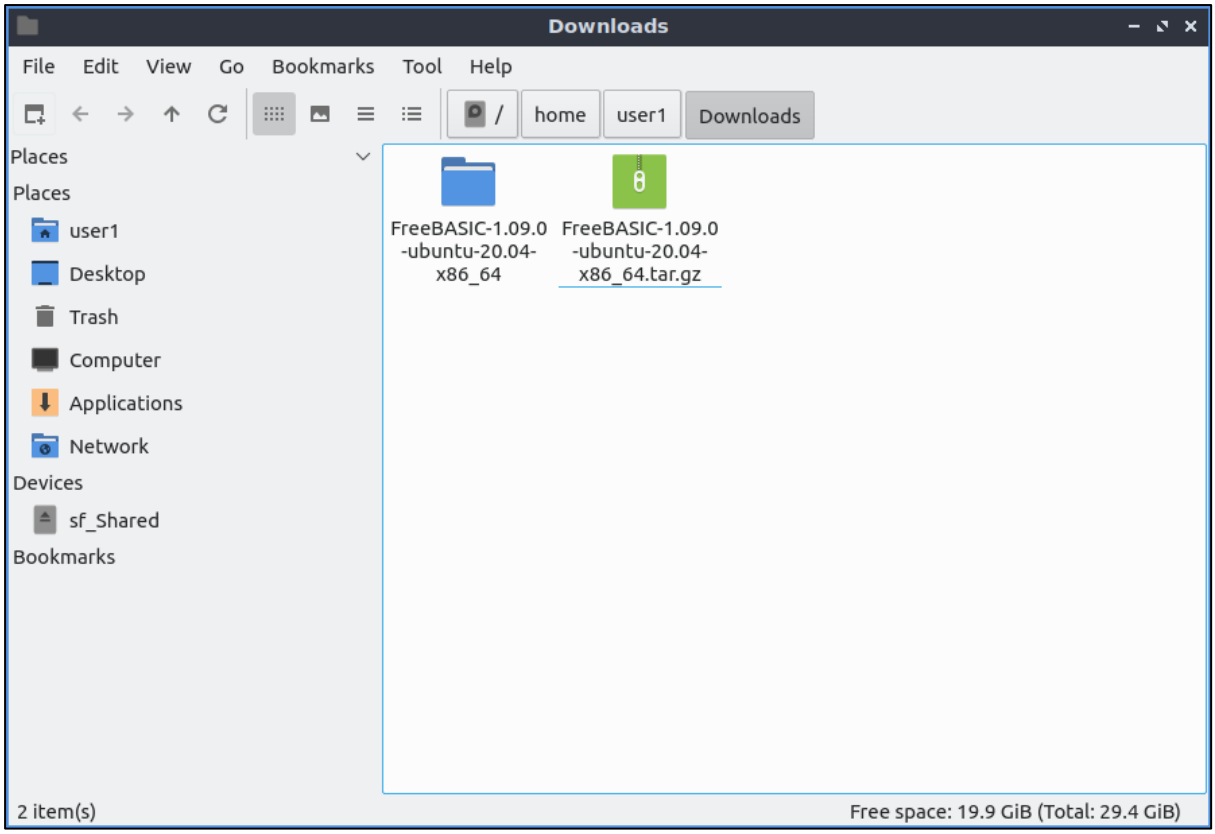
`/home/user1/Downloads`



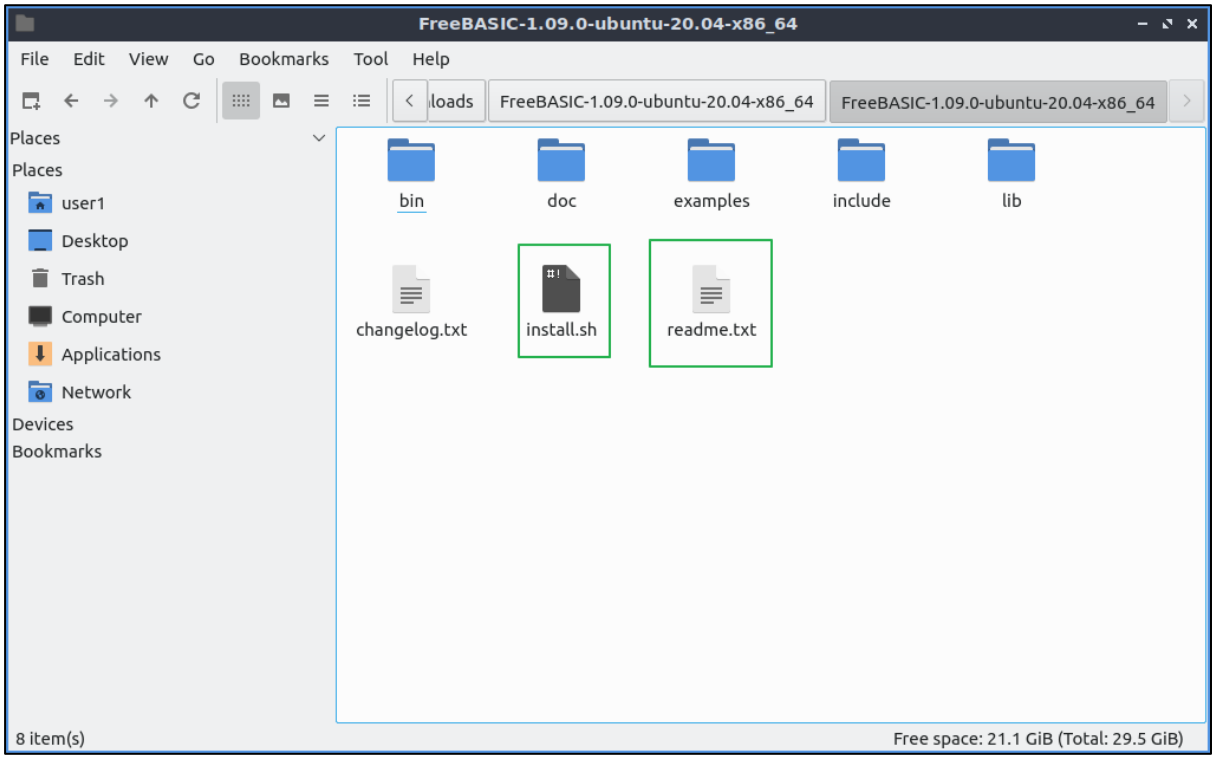
Extract the contents of the compressed file to a subfolder. If you accidently extract to the **`/Downloads`** folder, don't worry, we can delete all of the files after the install ☺

Right click on the downloaded file "FreeBASIC-1.09.0-ubuntu-20.04-x86_64.tar.gz" and select "Extract Here". This may use a different application if not using Lubuntu.

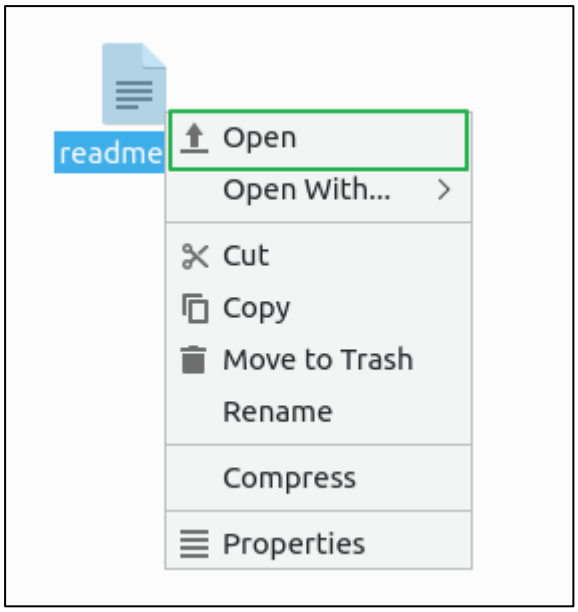




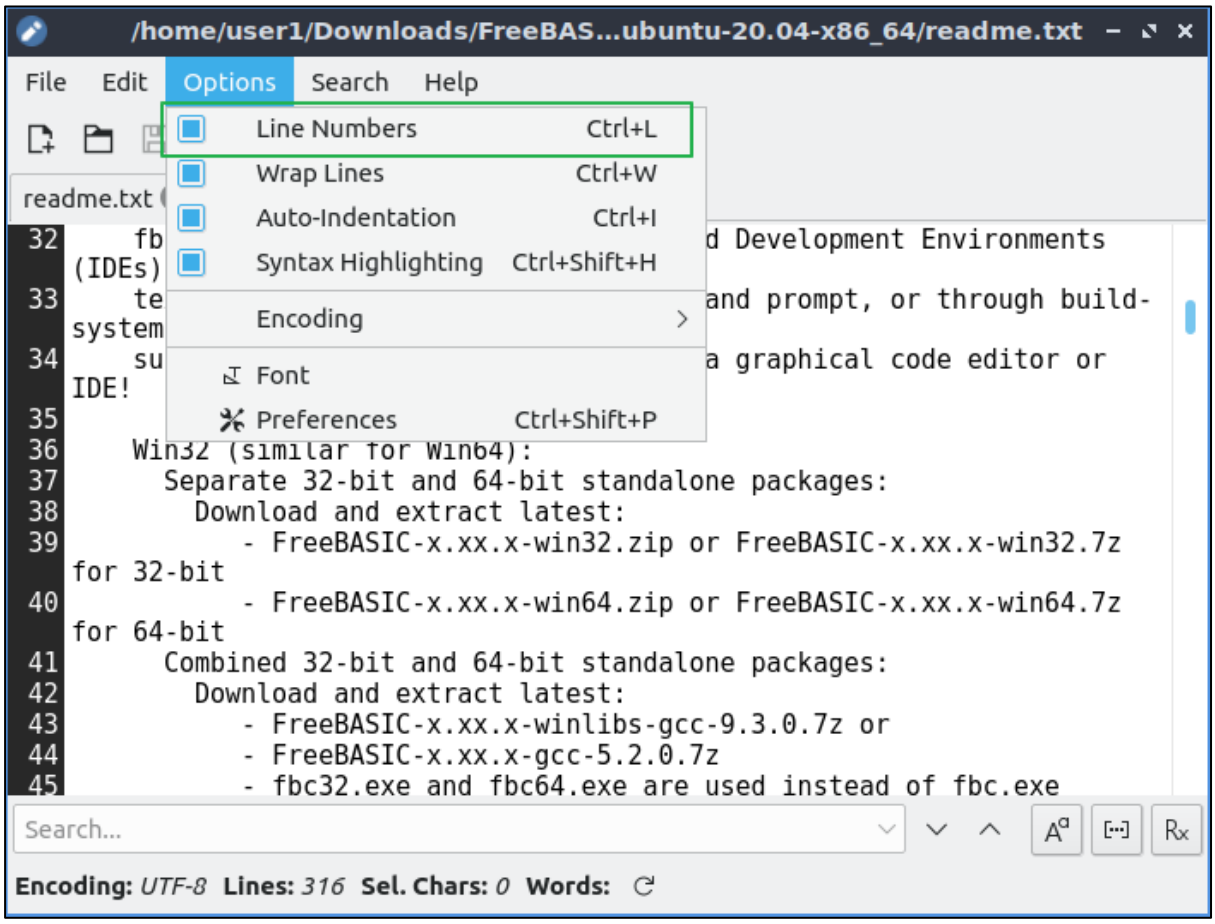
Navigate into the extracted folder **/Downloads/FreeBASIC-1.09.0-ubuntu-20.04-x86_64/FreeBASIC-1.09.0-ubuntu-20.04-x86_64/**



Open readme.txt in a text editor and keep it open.



Select the “Options” drop down and select [/] Line Numbers.



Scroll down to line 76. This is the correct and only way to successfully install the application.

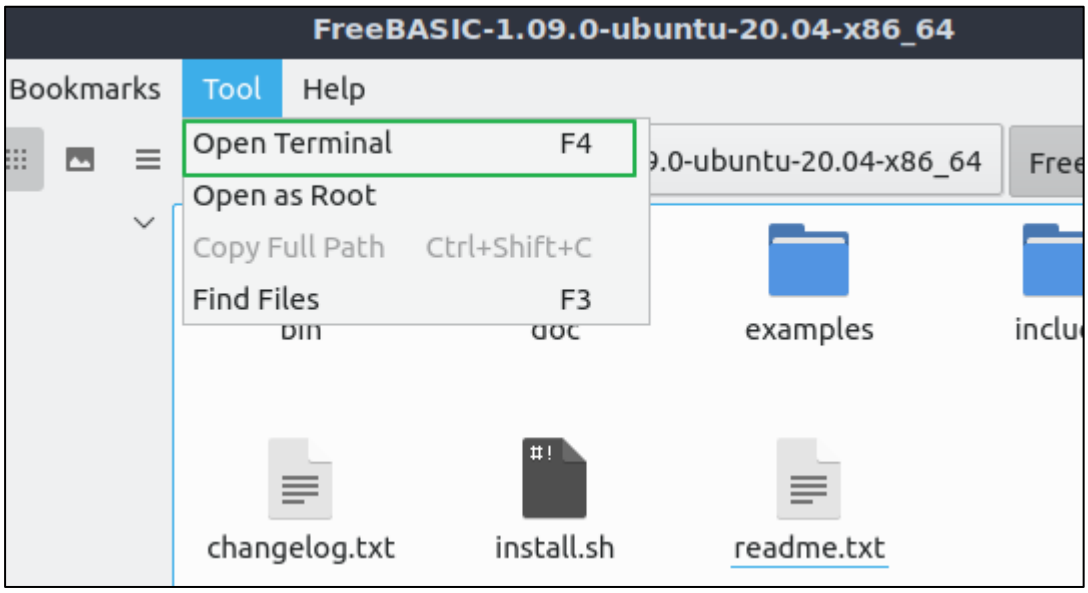
Open a terminal and cd into the extracted FreeBASIC-x.xx.x-linux directory, and run:

"`sudo ./install.sh -i`" to copy the FB setup into `/usr/local`.

Open a terminal window from PCManFM-Qt file manager by selecting “Tool -> Open Terminal F4”.

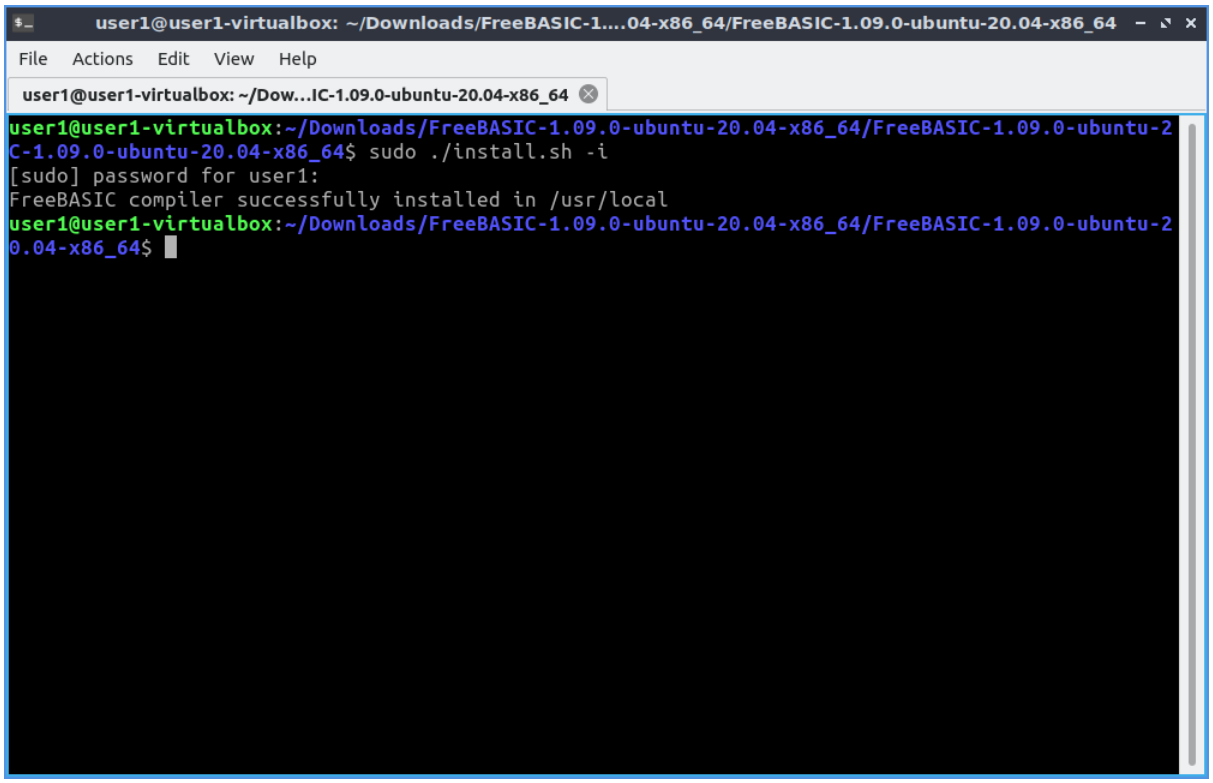
This will open the terminal in the current directory.

“**`home/user1/Downloads/FreeBASIC-1.09.0-ubuntu-20.04-x86_64/FreeBASIC-1.09.0-ubuntu-20.04-x86_64`**”

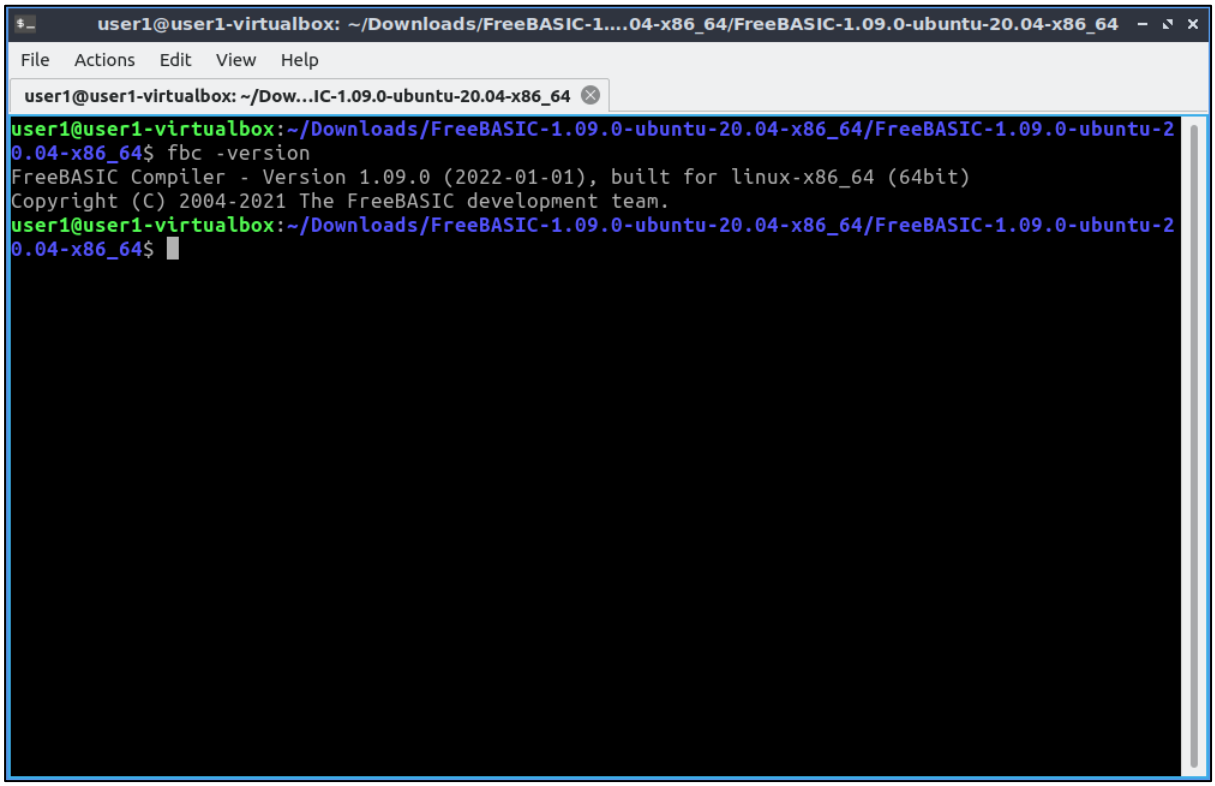


Write or copy/paste the install command into the terminal window and select [Enter].

`sudo ./install.sh -i`



Type **`fbc -version`** followed by [Enter]. If the install worked correctly you will see the FBC version displayed.



If you notice the time stamp you will see that this is a modern programming environment. It is in effect a GCC compiler with a precompiler front end that translates FreeBASIC code into C/C++ source code before compiling to a native executable with GCC. Although beyond the scope of a beginner, Python3 is capable of being compiled into C++ source code and then compiled to a native executable in the same way.

I have included FreeBASIC as many of the constructs of Python have been drawn from BASIC and C/C++ languages. FreeBASIC is like a bit of glue that exemplifies the constructs of Python and sits somewhere between C/C++ and Python scripting ☺

FBC is a compiler the same as GCC is a compiler and library suite and has no IDE. You could write source in a text file and send it via the command line to the compiler the same as we can in C/C++ or Python, but an IDE makes coding far more convenient.

You can now delete all of the unpacked FBC install files in the Download folder.

geany-1.38

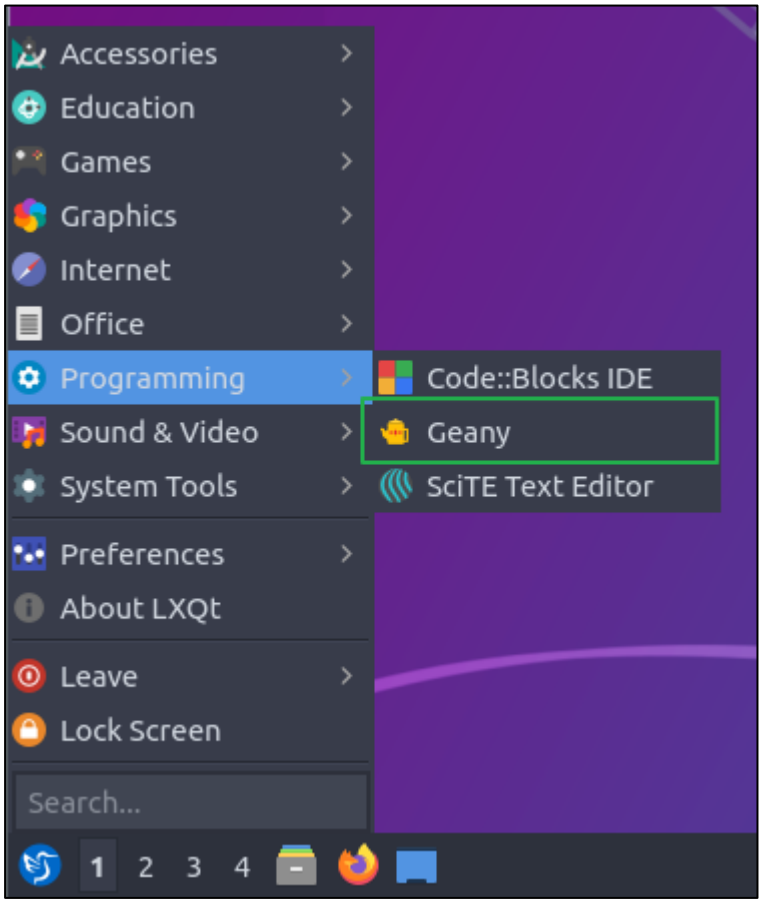
Geany is a Coding editor and lite IDE that works “out of the box” for FreeBASIC. It contains the same essential tool set as any other IDE/source code editor.

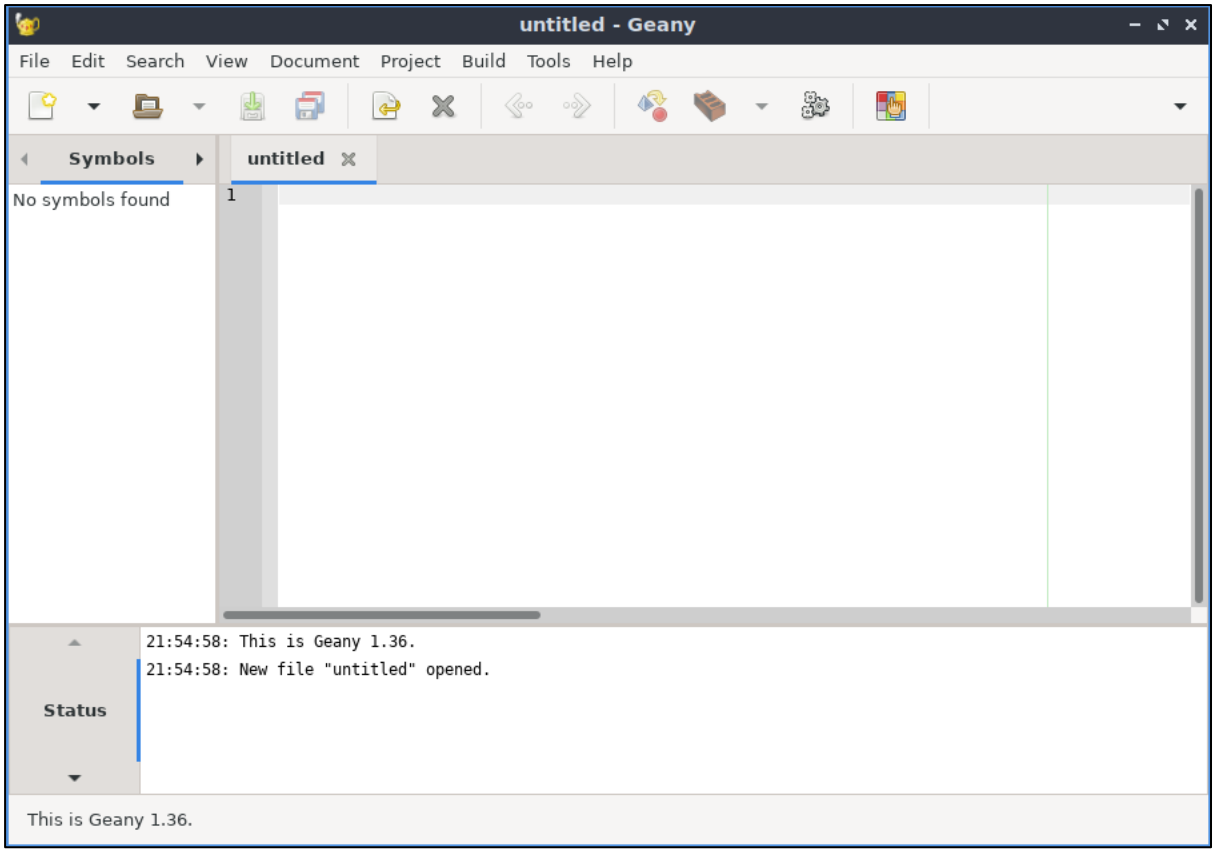
To install Geany write the following install command into a terminal window and press [Enter]

```
sudo apt-get install geany
```

```
user1@user1-virtualbox: ~/Downloads/FreeBASIC-1....04-x86_64/FreeBASIC-1.09.0-ubuntu-20.04-x86_64
File  Actions  Edit  View  Help
user1@user1-virtualbox: ~/Dow...IC-1.09.0-ubuntu-20.04-x86_64
user1@user1-virtualbox:~/Downloads/FreeBASIC-1.09.0-ubuntu-20.04-x86_64/FreeBASIC-1.09.0-ubuntu-2
0.04-x86_64$ sudo apt-get install geany
[sudo] password for user1:
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following package was automatically installed and is no longer required:
  libfwupdplugin1
Use 'sudo apt autoremove' to remove it.
The following additional packages will be installed:
  geany-common
Suggested packages:
  libvte9 doc-base
The following NEW packages will be installed:
  geany geany-common
0 to upgrade, 2 to newly install, 0 to remove and 0 not to upgrade.
Need to get 2,968 kB of archives.
After this operation, 11.8 MB of additional disk space will be used.
Do you want to continue? [Y/n] y
Get:1 http://au.archive.ubuntu.com/ubuntu focal/universe amd64 geany-common all 1.36-1build1 [1,8
43 kB]
Get:2 http://au.archive.ubuntu.com/ubuntu focal/universe amd64 geany amd64 1.36-1build1 [1,124 kB
]
Fetched 2,968 kB in 2s (1,461 kB/s)
Selecting previously unselected package geany-common.
(Reading database ... 274948 files and directories currently installed.)
Preparing to unpack .../geany-common_1.36-1build1_all.deb ...
```

You will now have the menu item under Programming for Geany. Open the Geany IDE.

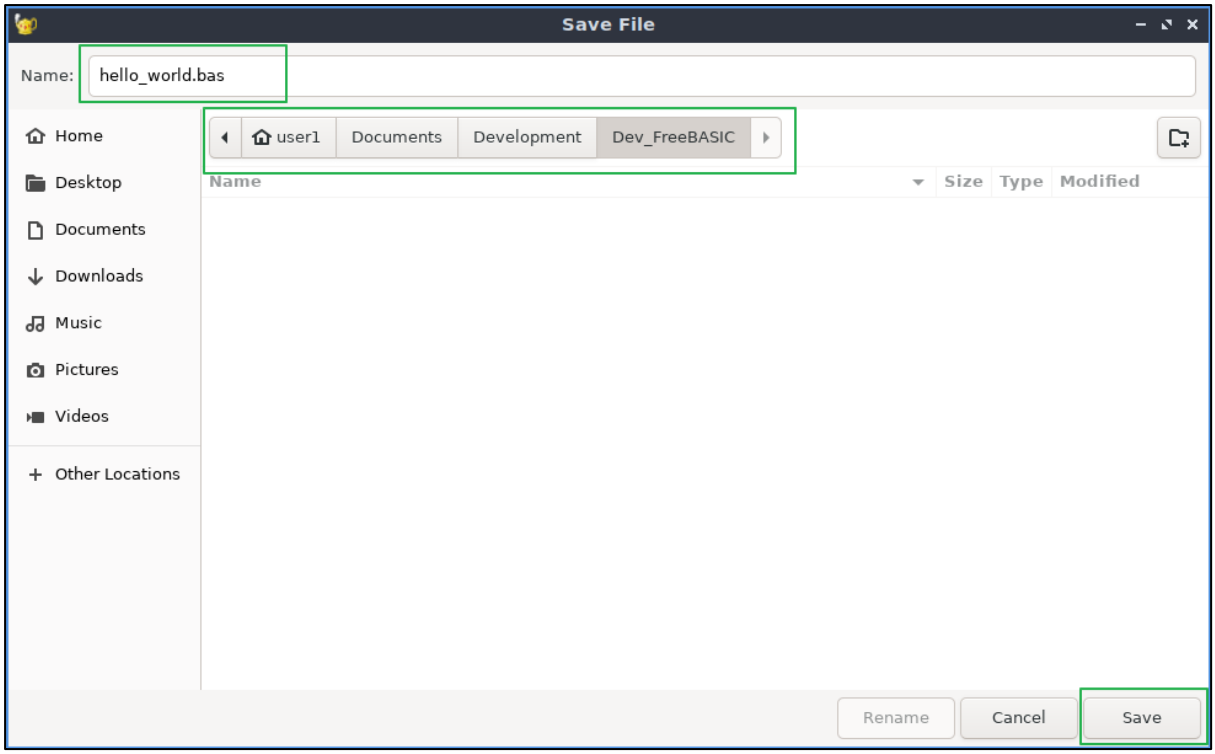




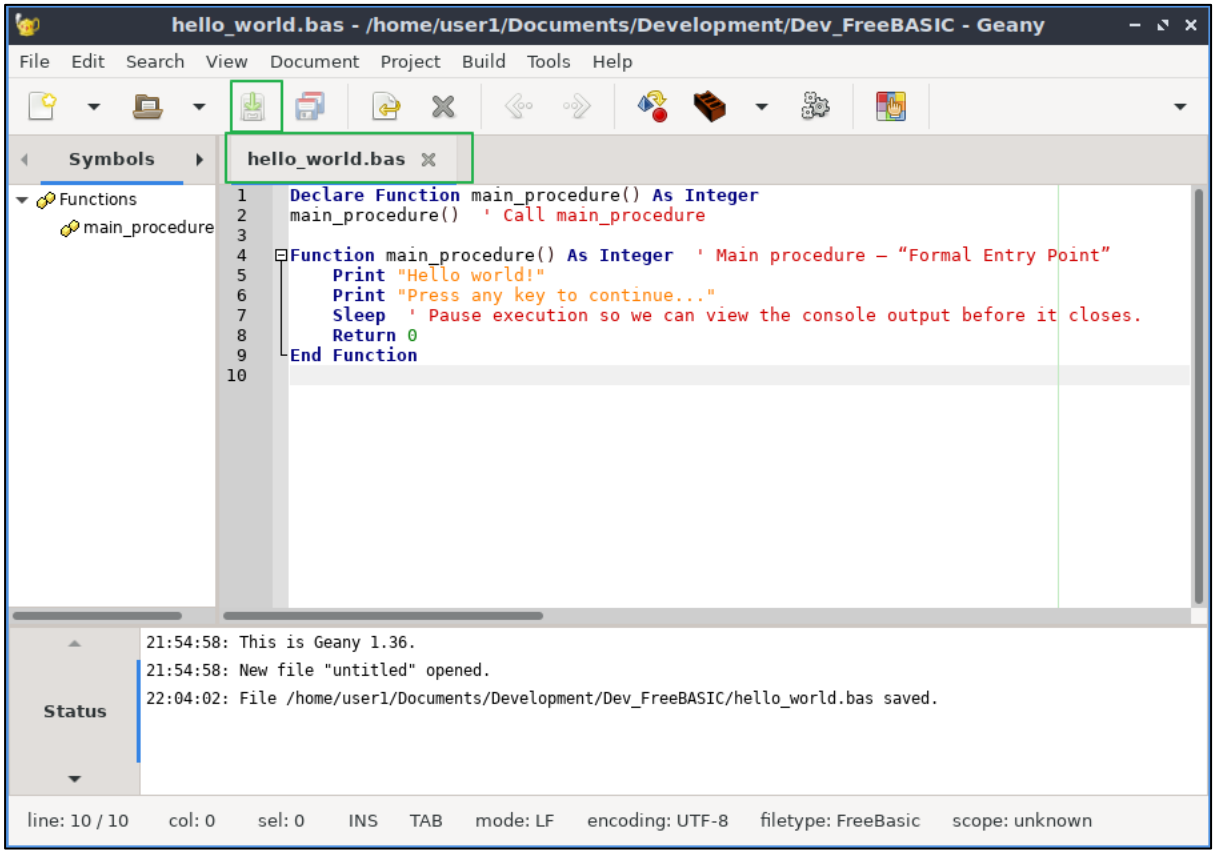
Write or copy/paste the following code lines into the source code TAB “untitled.

```
' Helloworld.bas

Declare Function main_procedure() As Integer
main_procedure()  ' Call main_procedure

function main_procedure() As Integer  ' Main procedure – “Formal Entry point”
    print "Hello world!"
    print "Press any key to continue..."
    sleep  ' Pause execution so we can view the console output before it closes.
    return 0
end Function
```

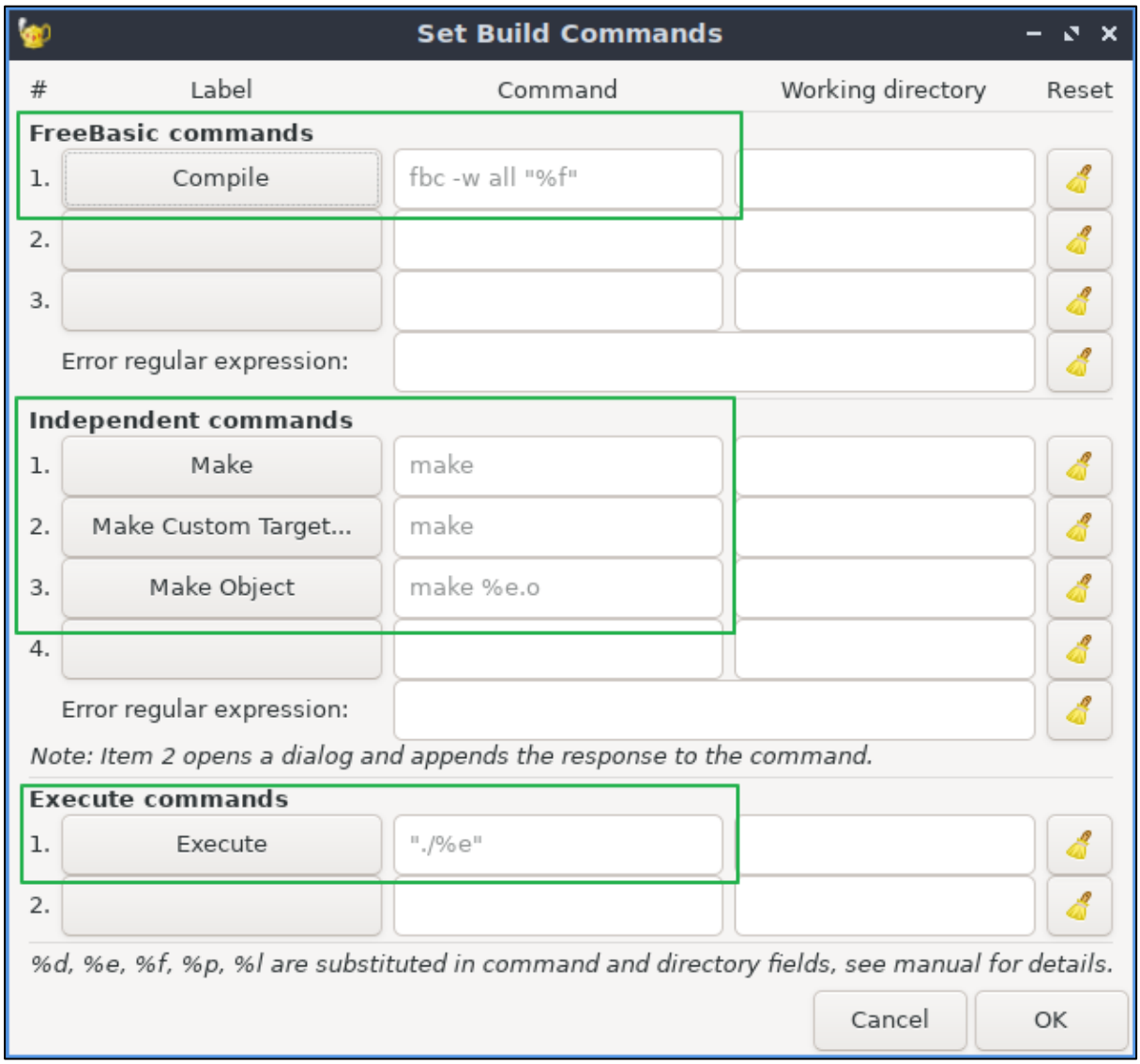
From the File menu select Save As and navigate to your dev folder and save the file as **hello_world.bas**



You will now see that the text highlighting for BASIC. The Filename and save button will be highlighted if the document has not been saved.

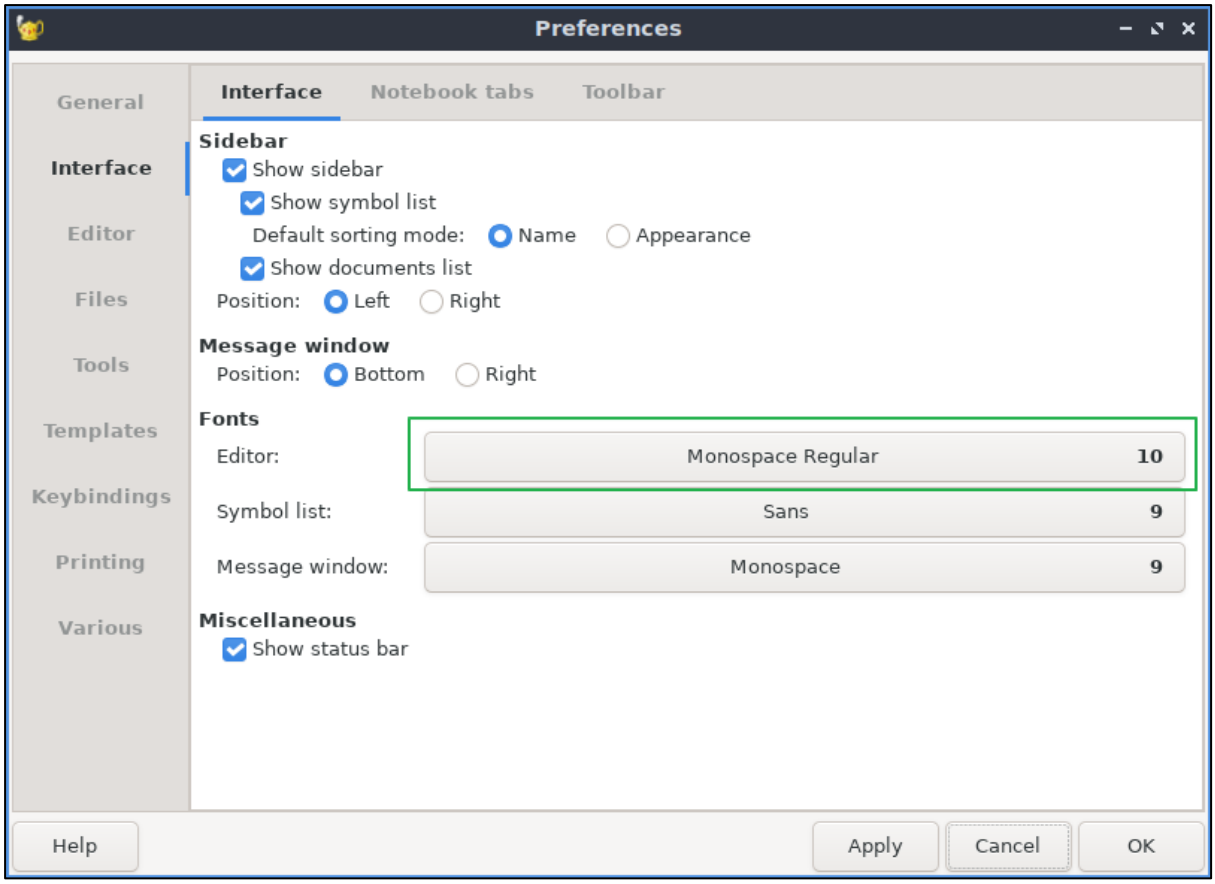


FreeBASIC Compiler is the default for *.bas in Geany, so nothing additional has to be added to compile the application. [Consider adding -v -exx to the compile line]

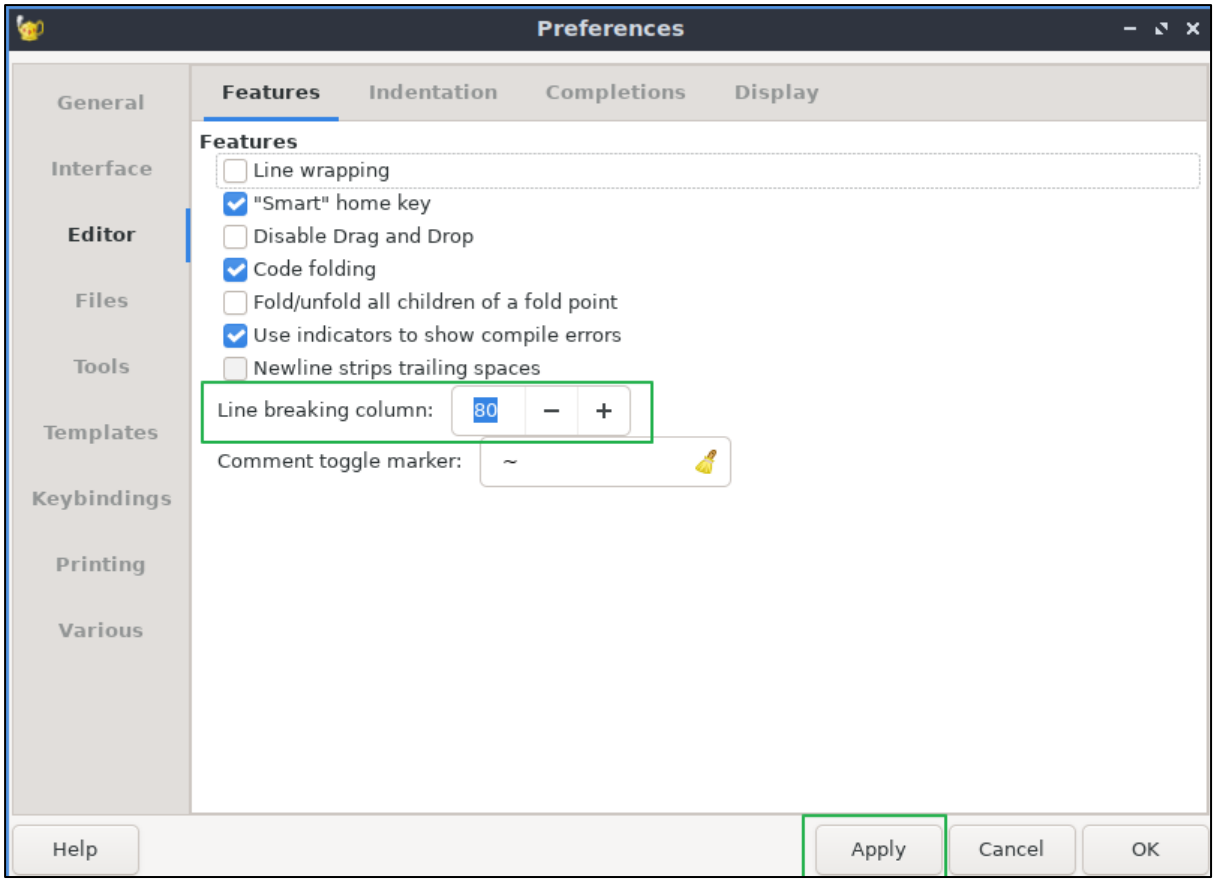


Next open from the menu bar Edit -> Preferences.

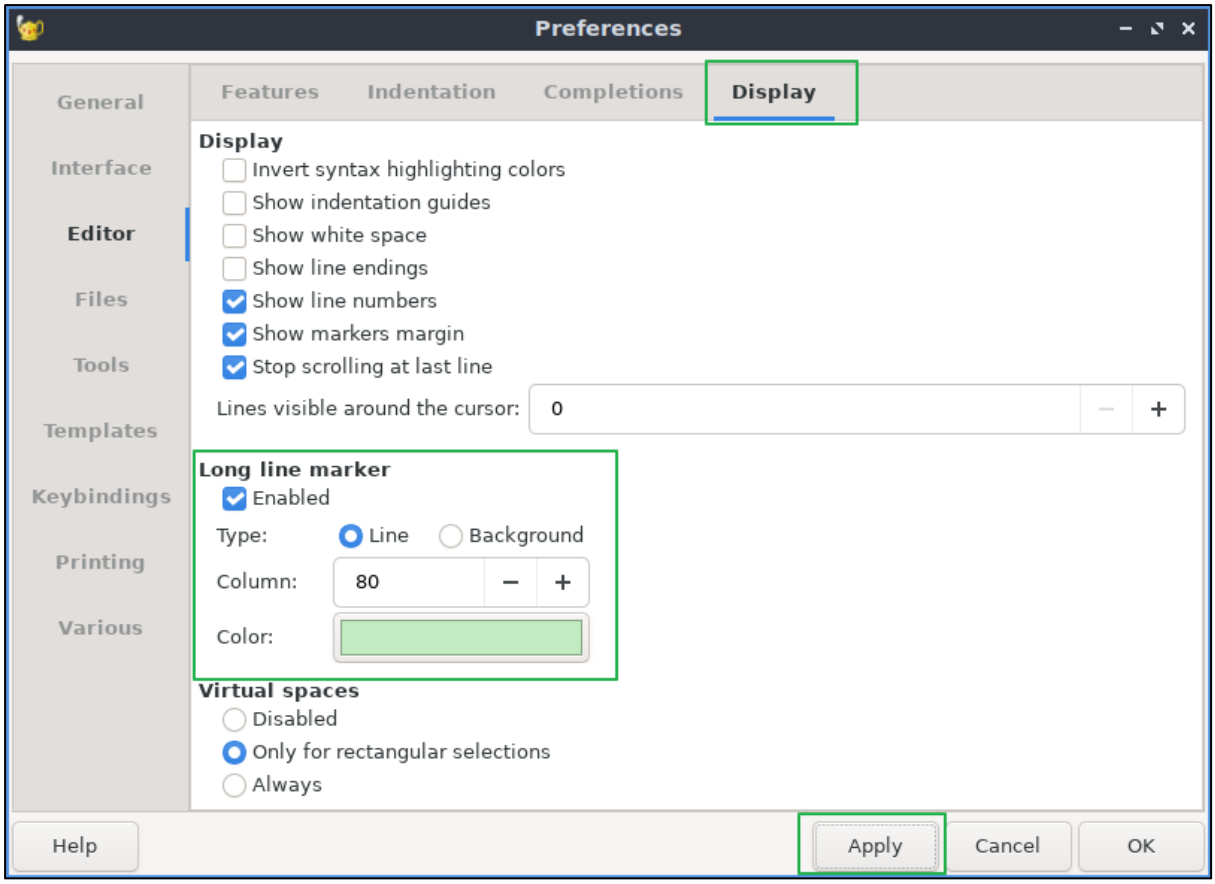
Select the Interface TAB on the left. You can change the default font size to 11 or 12 if you find it too small to read. There is a readability fault with Monospace Regular above 10pt where the lower part of the character is cut off making it impossible to see underscores '_', so if you need a larger font size try some of the other monospaced fonts such as FreeMono regular. Don't forget to click Apply to save the new configurations. See update below "Correcting line spacing margins on some fonts".



Next change to the Editor TAB on the left and change Line breaking columns to 80.



Next change to the Display TAB at the top and change “Long line marker” to 80.



You can close the Preferences dialog after ensuring that you have “Applied” the changes.

Correcting line spacing margins on some fonts.

Open "Tools" > "Configuration Files" > "filetypes.common".

Uncomment the second line to turn on styling

```
#~ [styling]
```

so that it becomes...

```
[styling]
```

Scroll down to line ~90 or search for “line_height”:

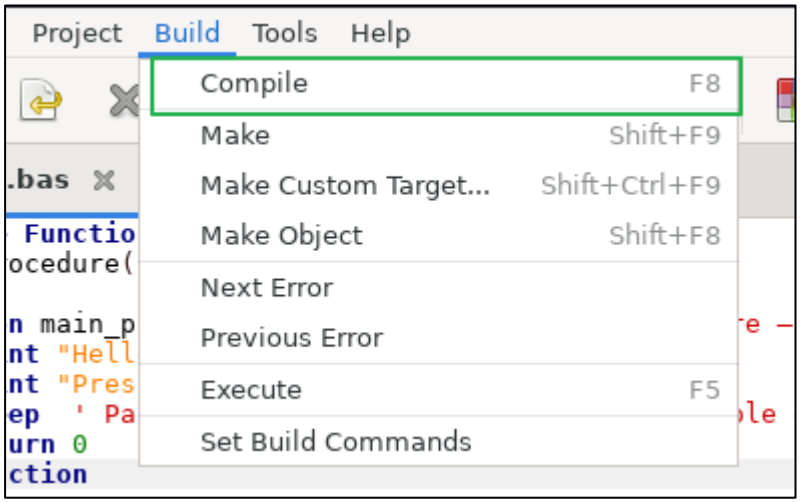
#~ # first argument: amount of space to be drawn **above** the line's baseline
#~ # second argument: amount of **space** to be drawn **below** the line's baseline
#~ line_height=0;0;

Change #~ line_height=0;0;

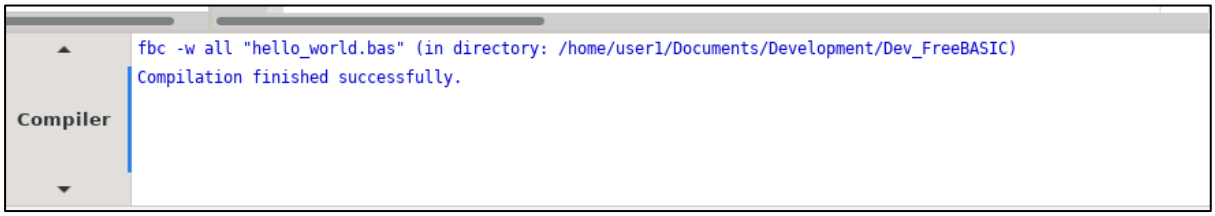
to

```
line_height=0;2;
```

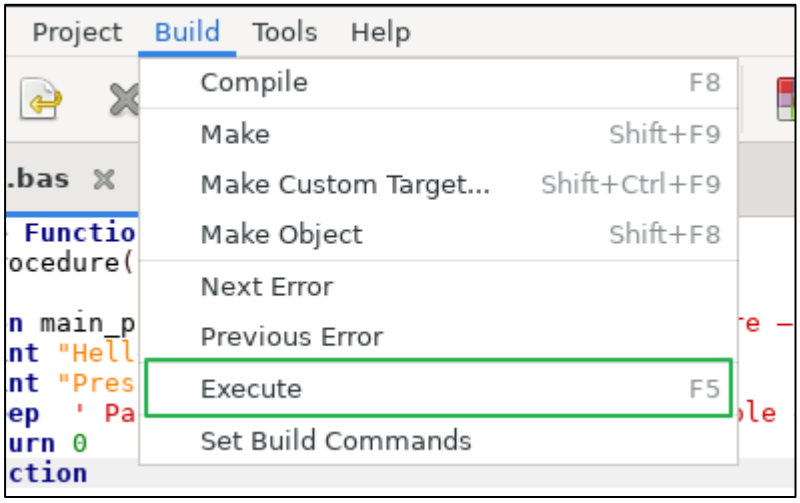
Next from the menu bar select Build -> Compile…



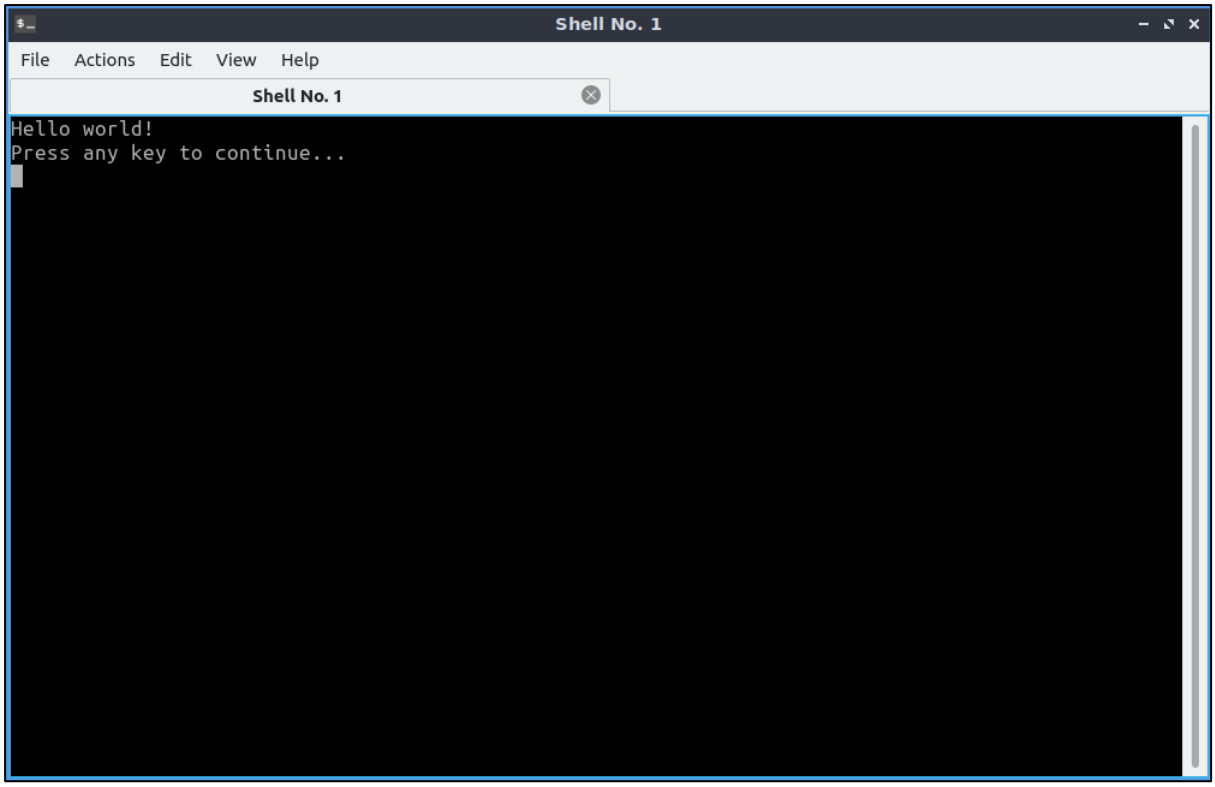
Any warning or errors from the compiler will be displayed in the lower panel. If all has worked well to this point you should see Compilation finished successfully.



Next select the Build -> Execute from the menu bar.

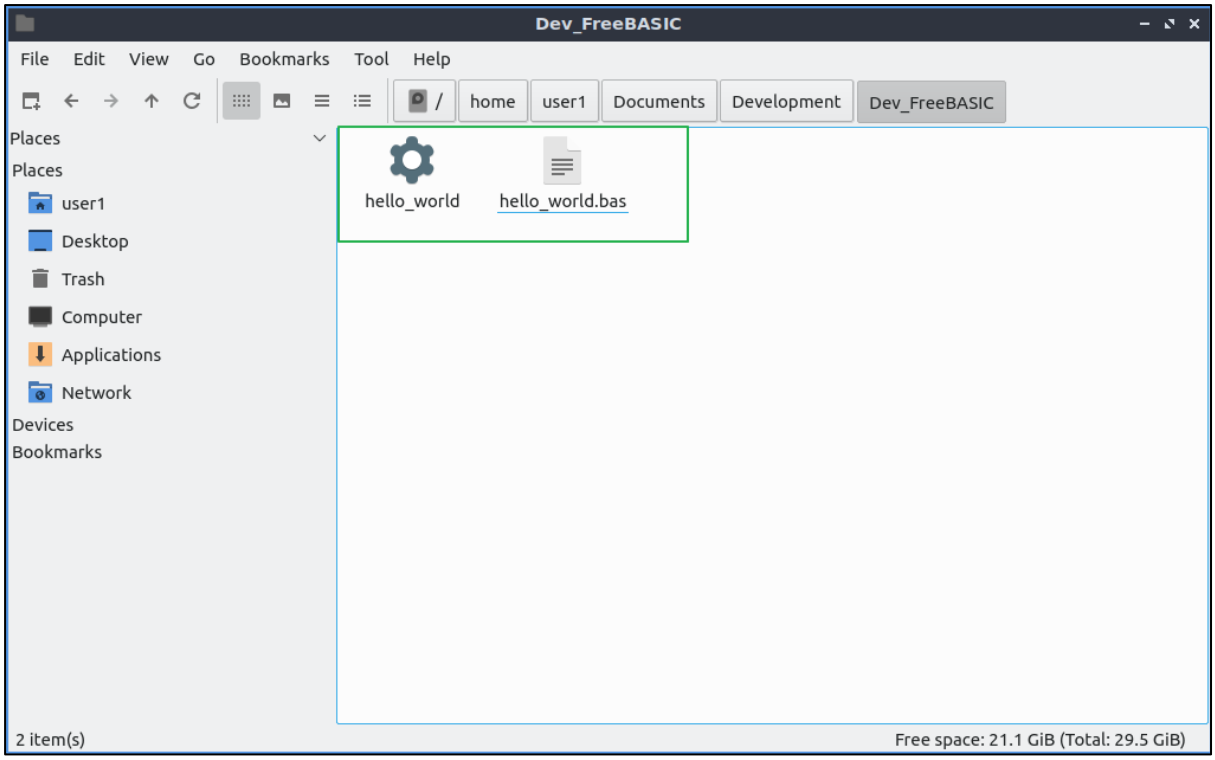


You should see your Hello World application displayed in the terminal window.

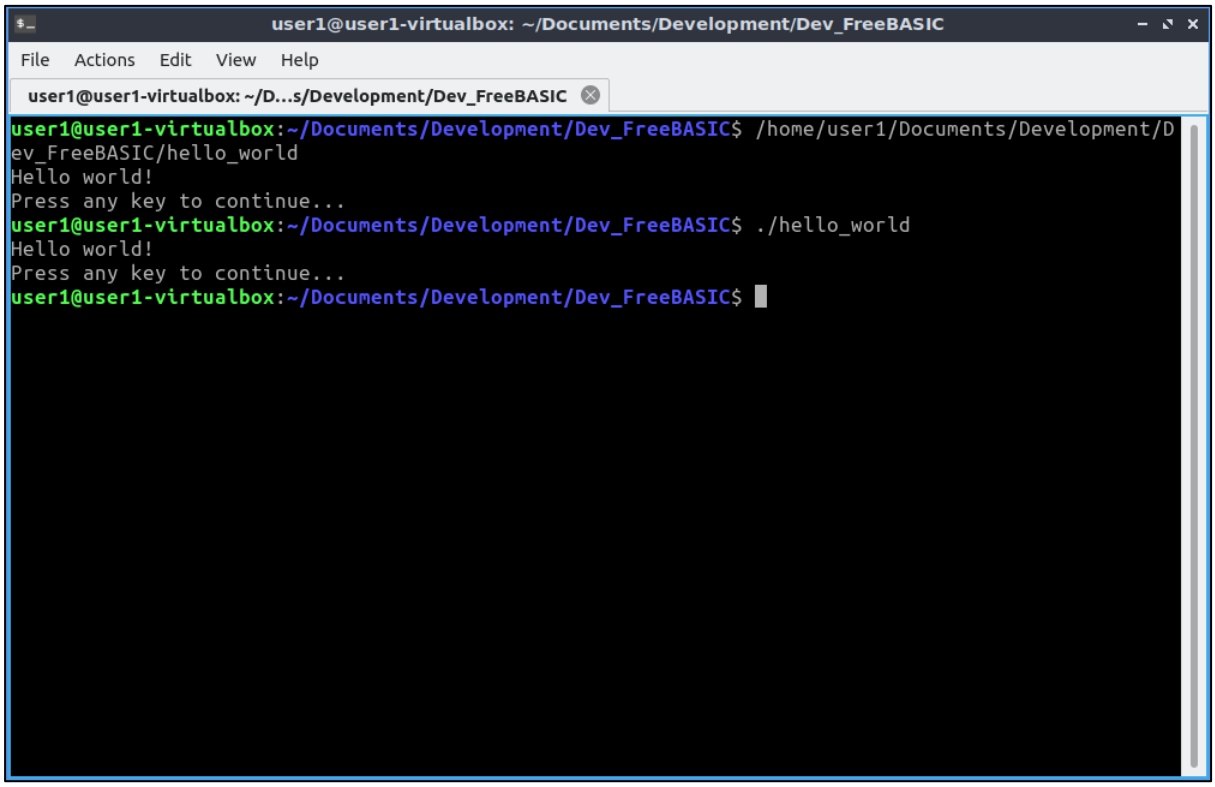


While debugging you will need to make changes to your code and then "Compile" each time until you receive a compilation without errors. You can then "Execute" the application to test for runtime errors.

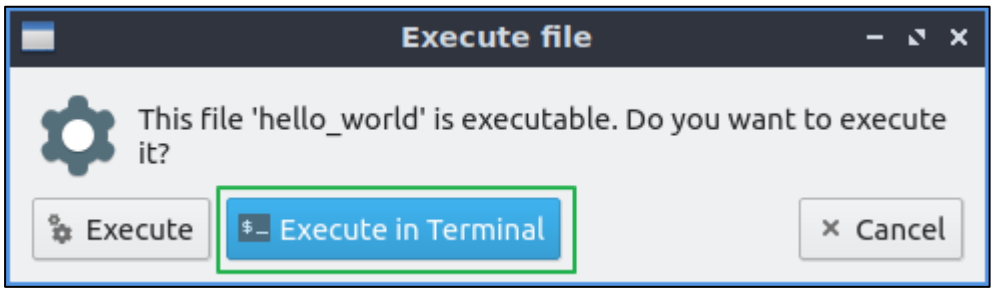
If you open the file manager and navigate to the "**/home/user1/Documents/Development/Dev_FreeBASIC**" directory you will see your source file as well as the output Linux executable file.

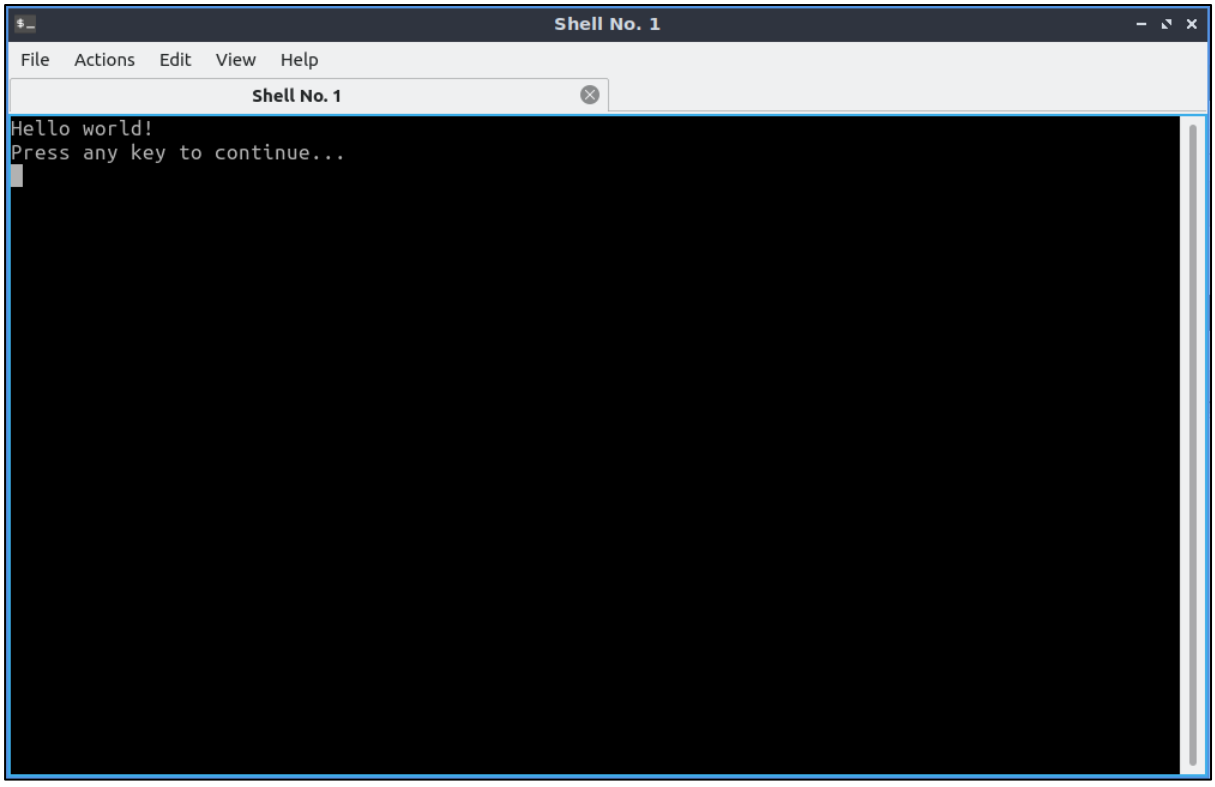


You should be able to execute the file from the command line by entering the full path to the executable file "**`/home/user1/Documents/Development/Dev_FreeBASIC/hello_world`**" **or by using "`./hello_world`**".



Alternatively, by double clicking the file and select “Execute in Terminal”. Remember this is a console application.



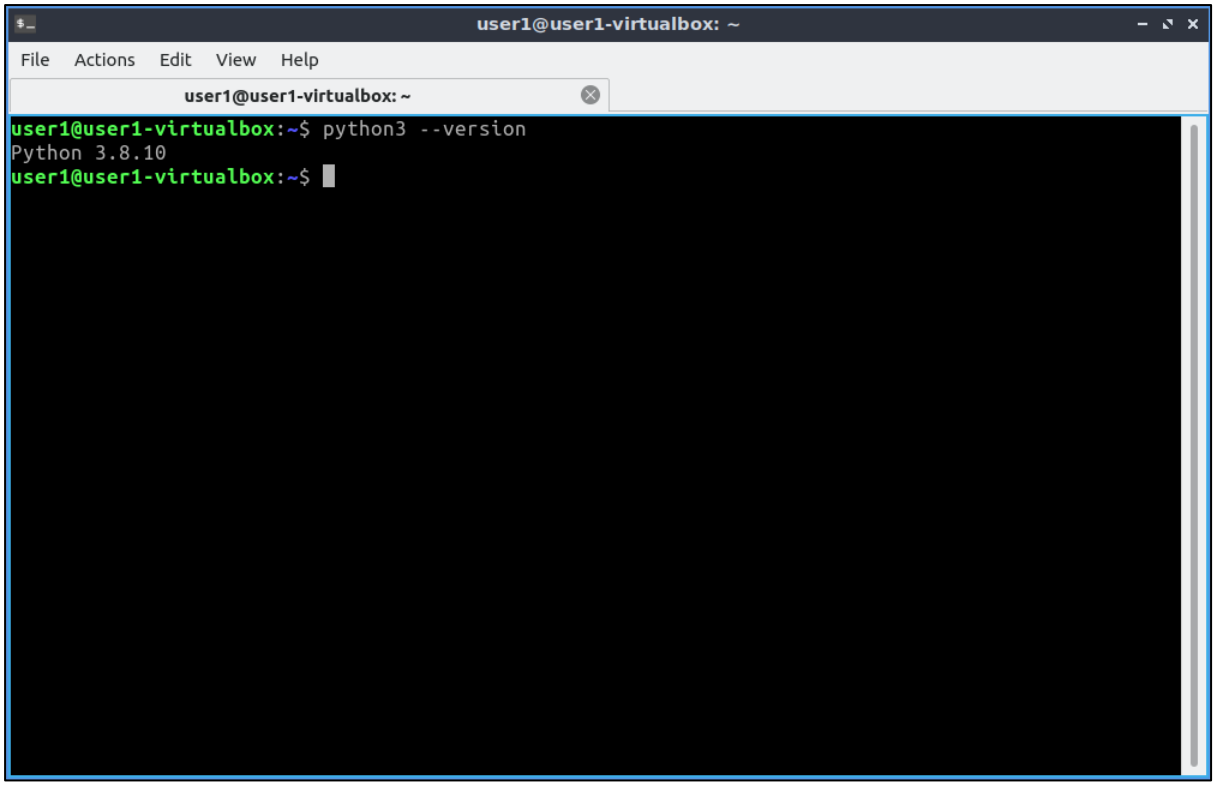


Geany and FBC do not force you to use a single source file in the editor. You can open as many source TABs as you wish each with an individual file name and compile each separately. Each executable will reflect the name of the source.bas file name.

Have a try at some of the examples that I have provided in “A Beginners Guide To Programming” and have fun learning to code.

Install Python 3 development tools

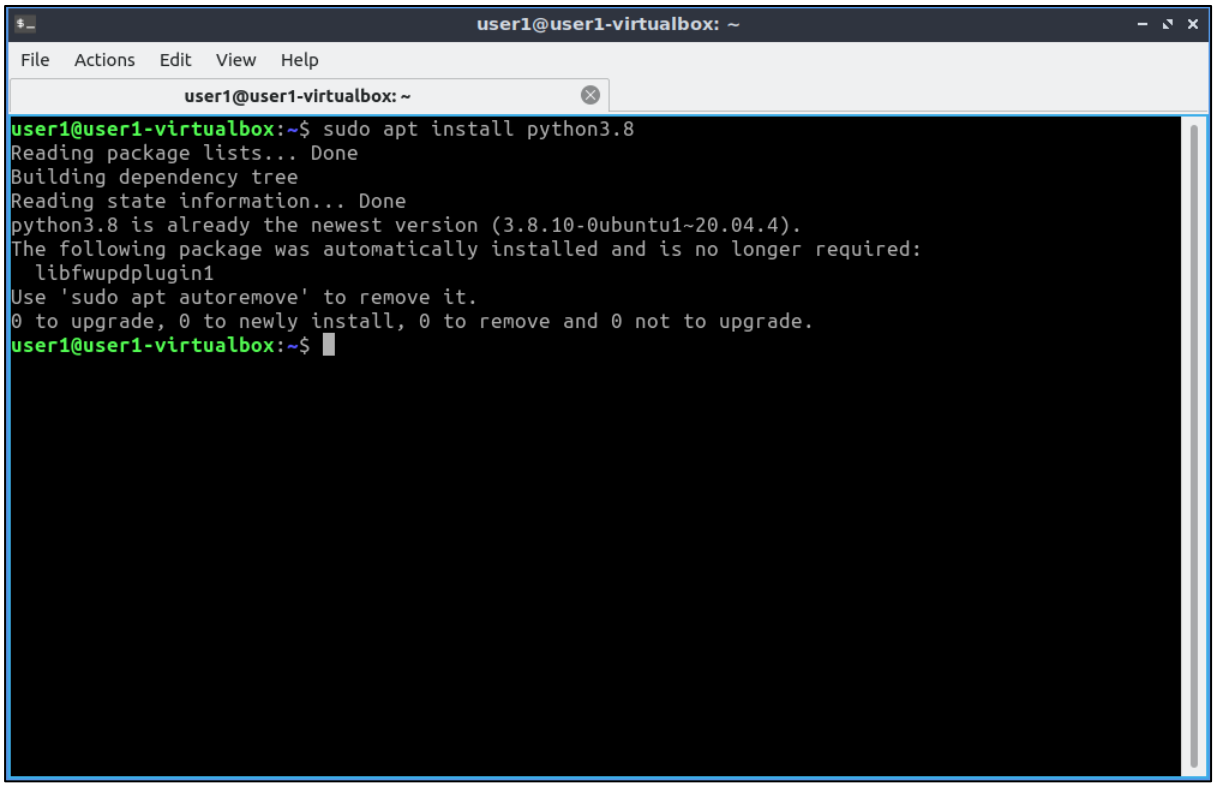
Python 3.8 is likely to be installed by default in Ubuntu distros. You can check the current version by entering **`python3 --version`** into a terminal. (Python 3.8.10)



If for some reason Python3.8.x is not installed, enter the following into the terminal.

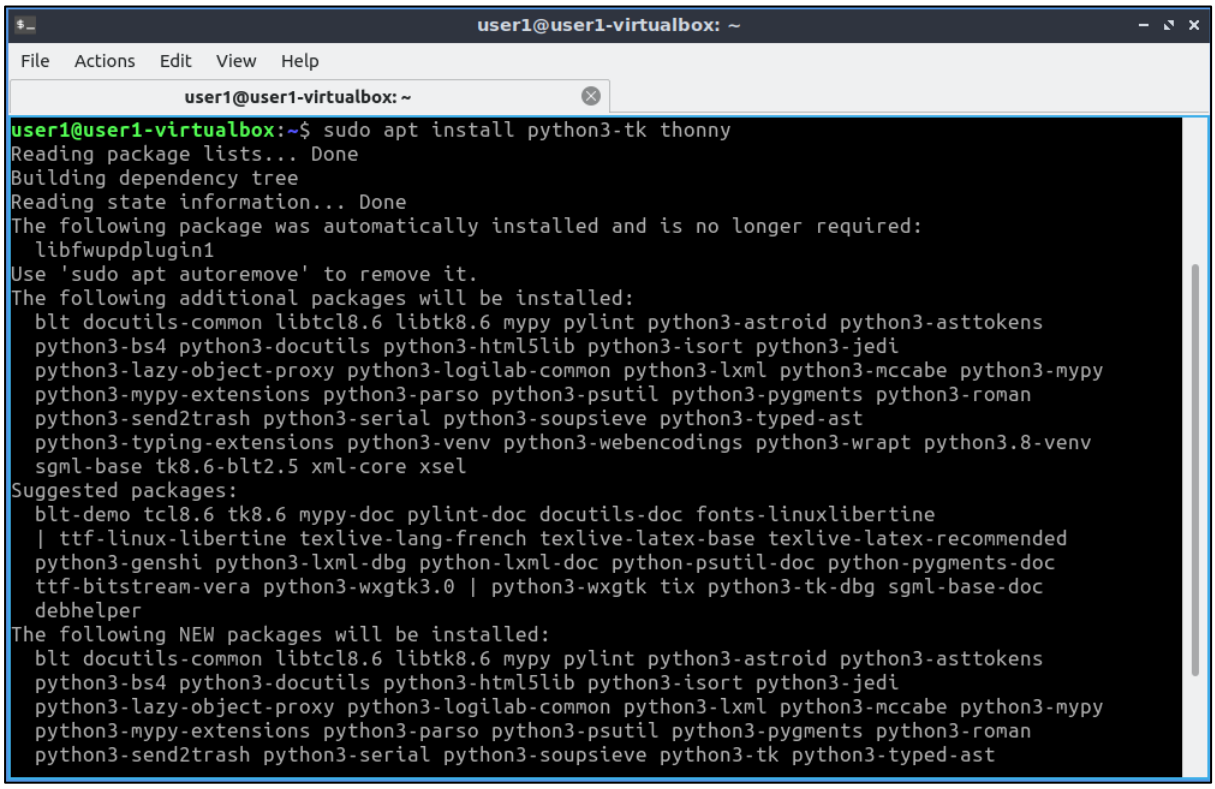
`sudo apt install python3.8` `(Note Python is usually installed by default)`

If it is already installed it will report as such in the terminal.

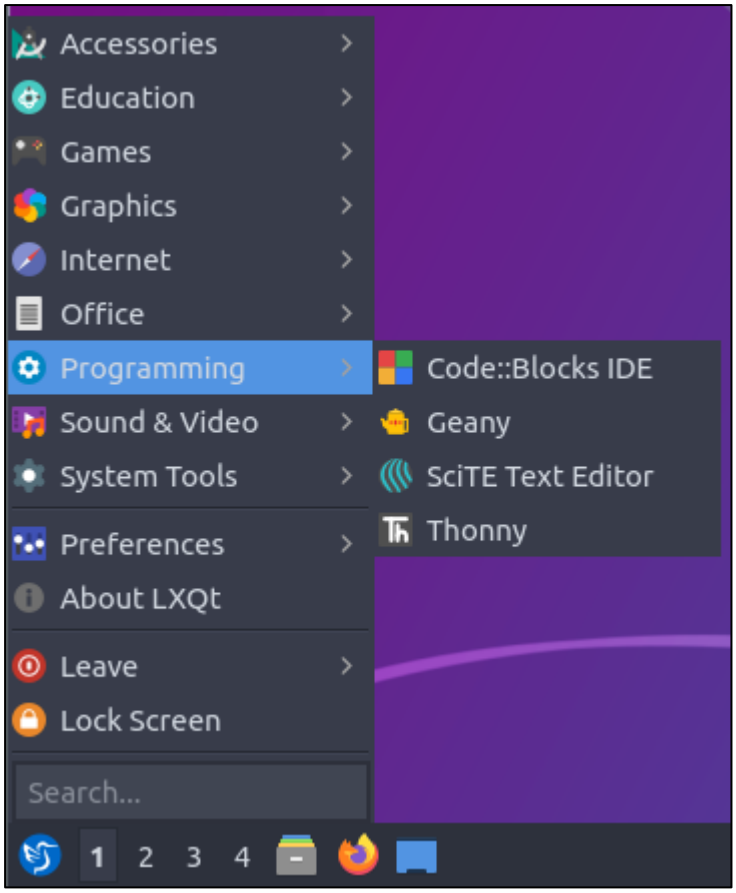


Thonny (3.2.7.1)

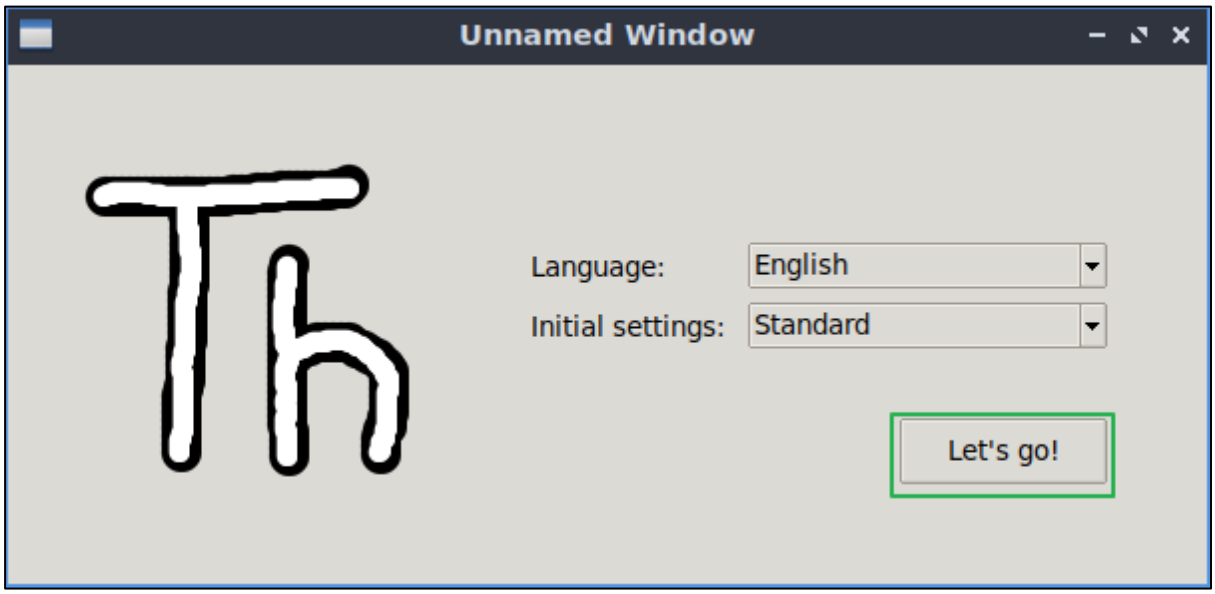
```
sudo apt install thonny
```

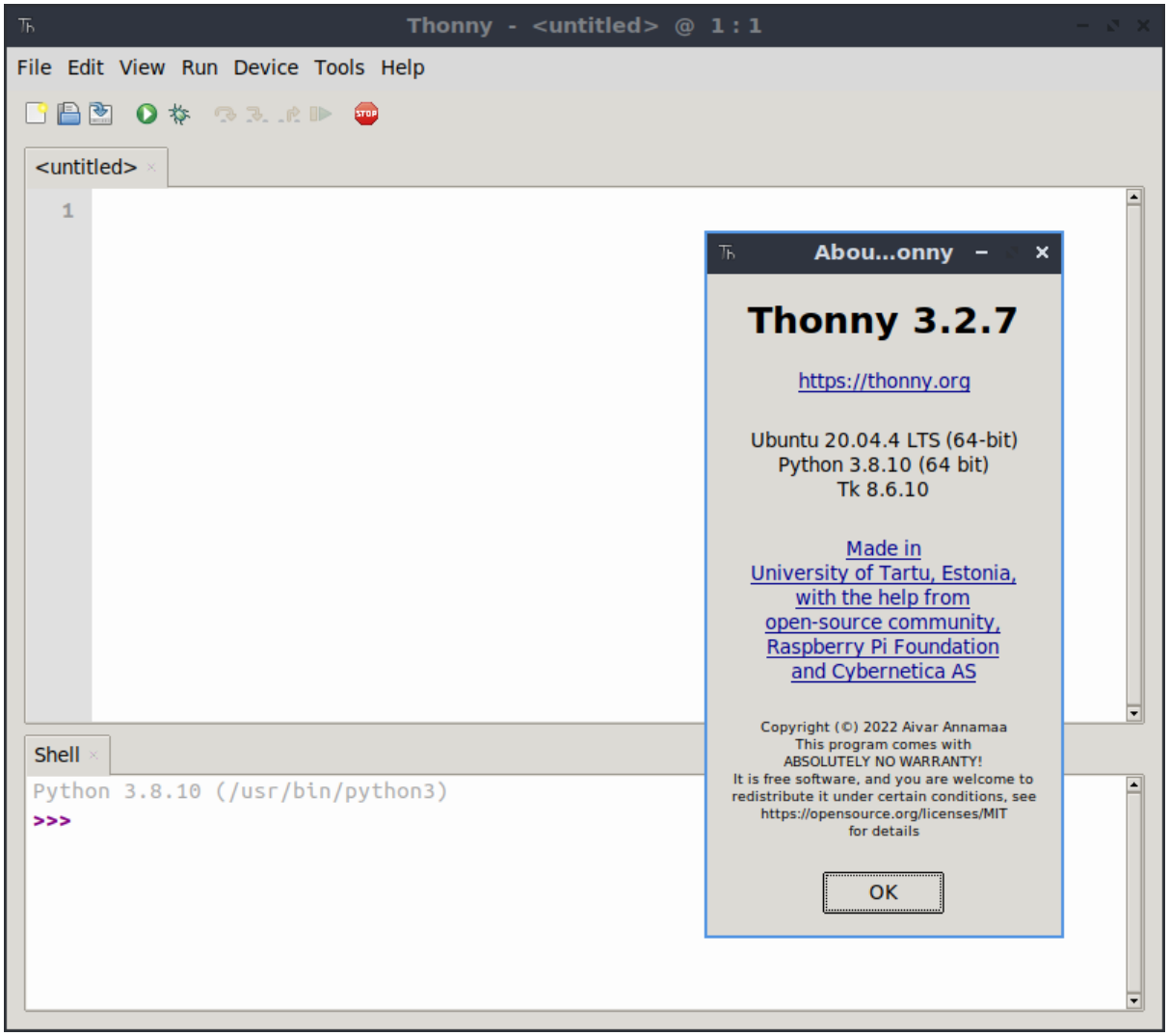


Launch the Thonny IDE from the start menu.



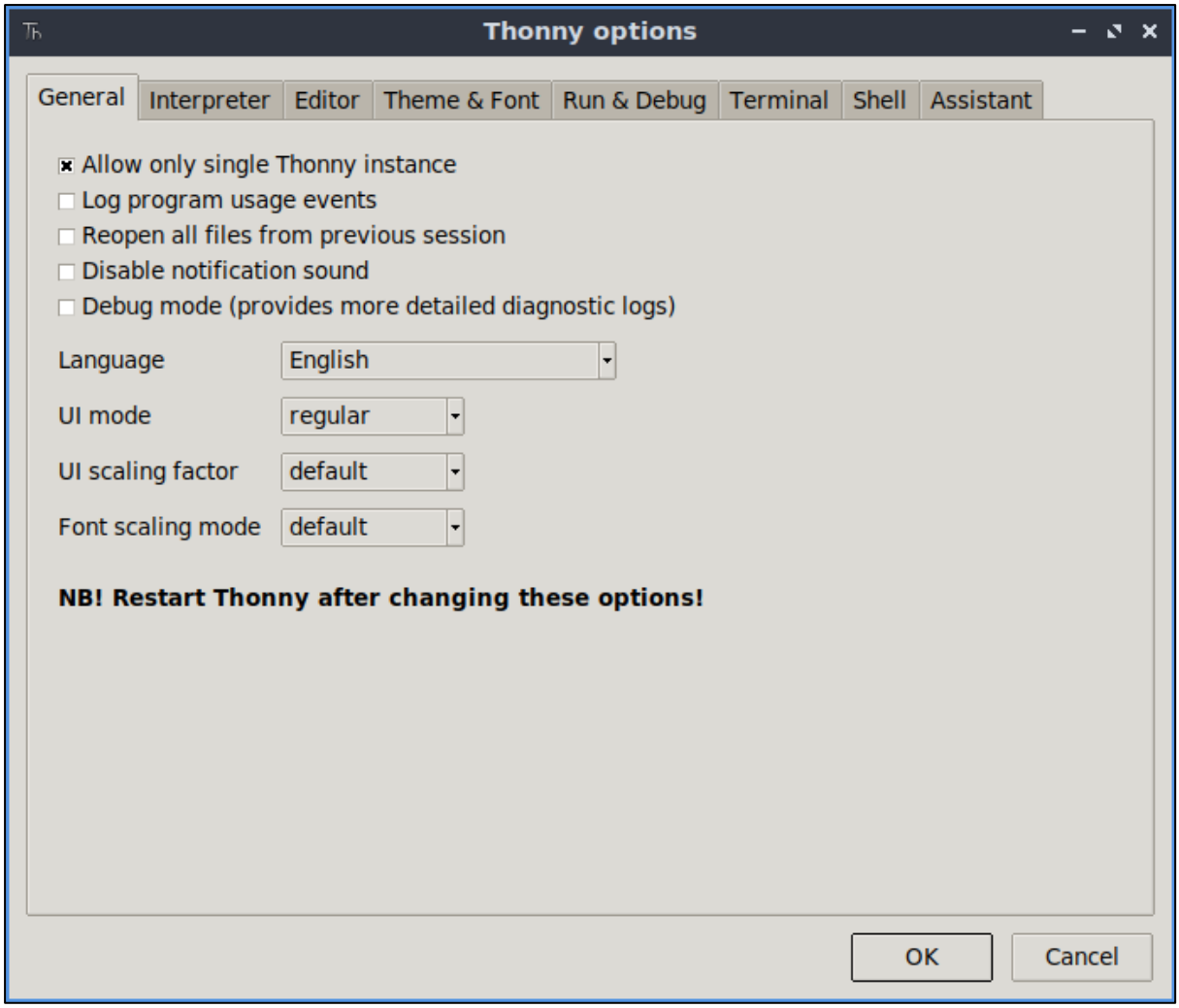
Select the defaults at start up…



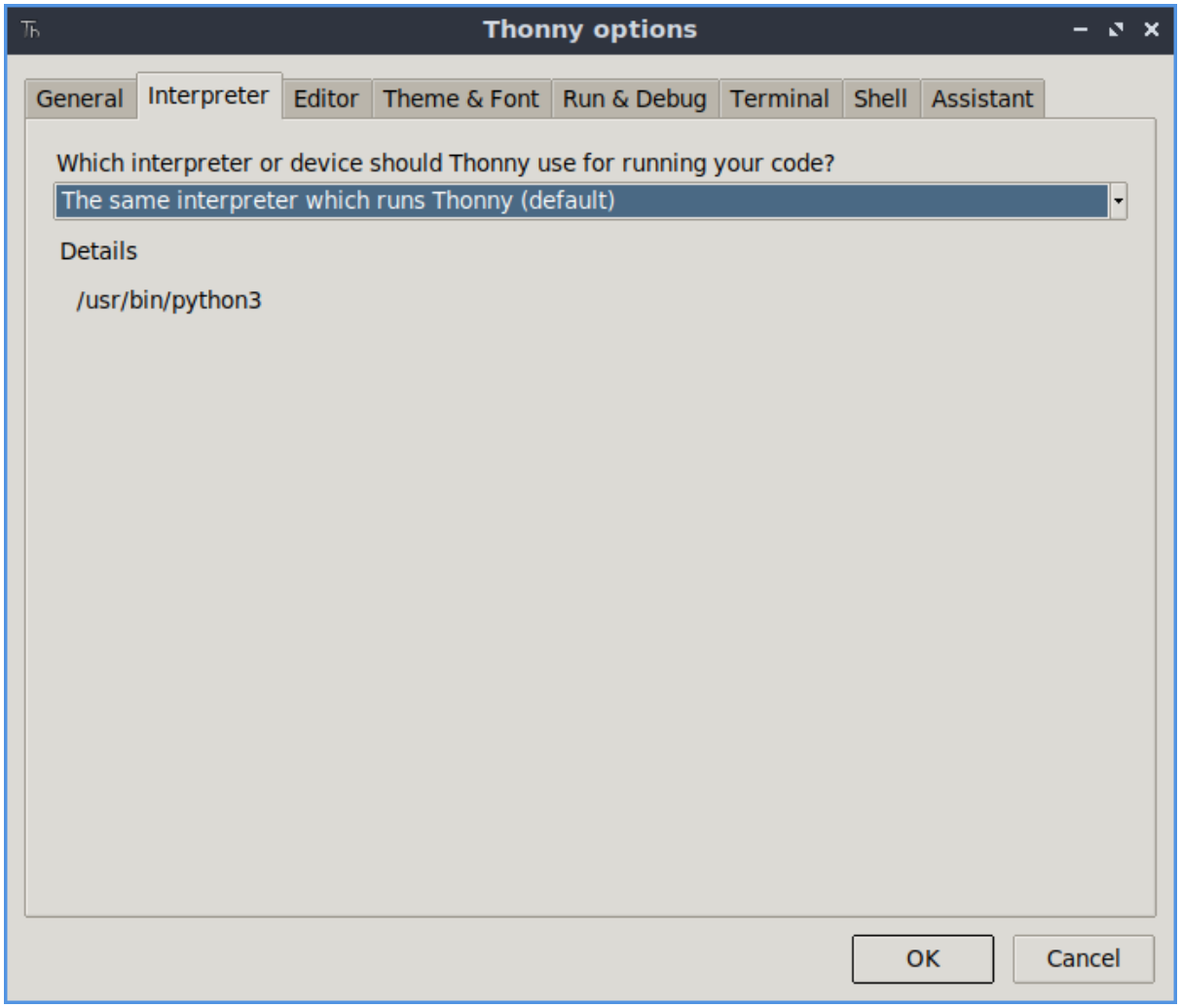


You can begin writing Python-3 scripts and testing them from this point if you wish, but would suggest the following changes to the application Options to make your learning process a little easier.

From the Menu bar select Tools -> Options…

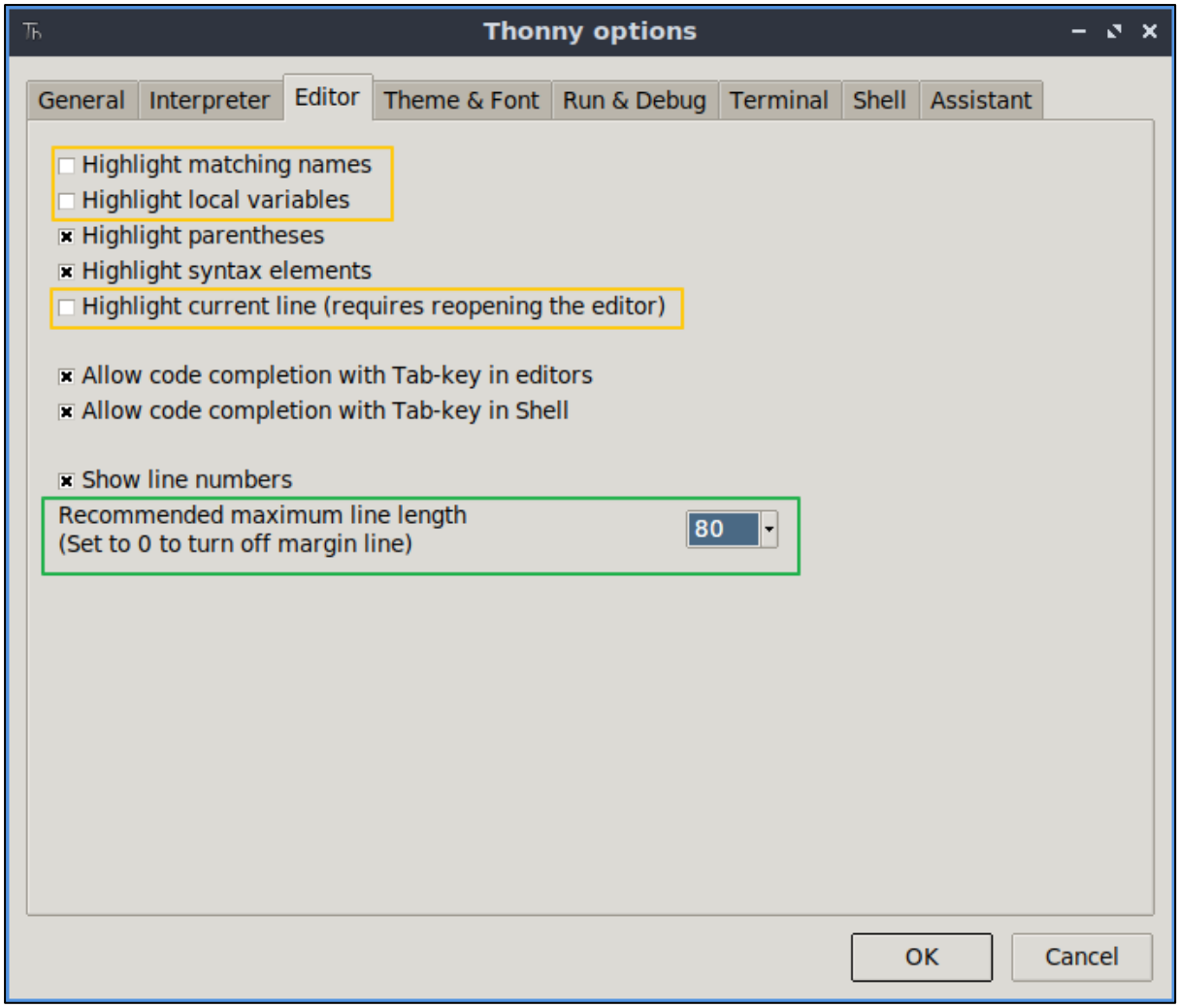


Change to the Interpreter TAB. We won't make any changes here at this time. If in the future you wish to use a separate Python 3 install rather than the default Python interpreter used by Thonny, select "Alternative Python 3 interpreter or virtual environment" and enter the path to the Python 3 executable you wish to use. Thonny uses the current Python3 install in Ubuntu by default `/usr/bin/python3`.



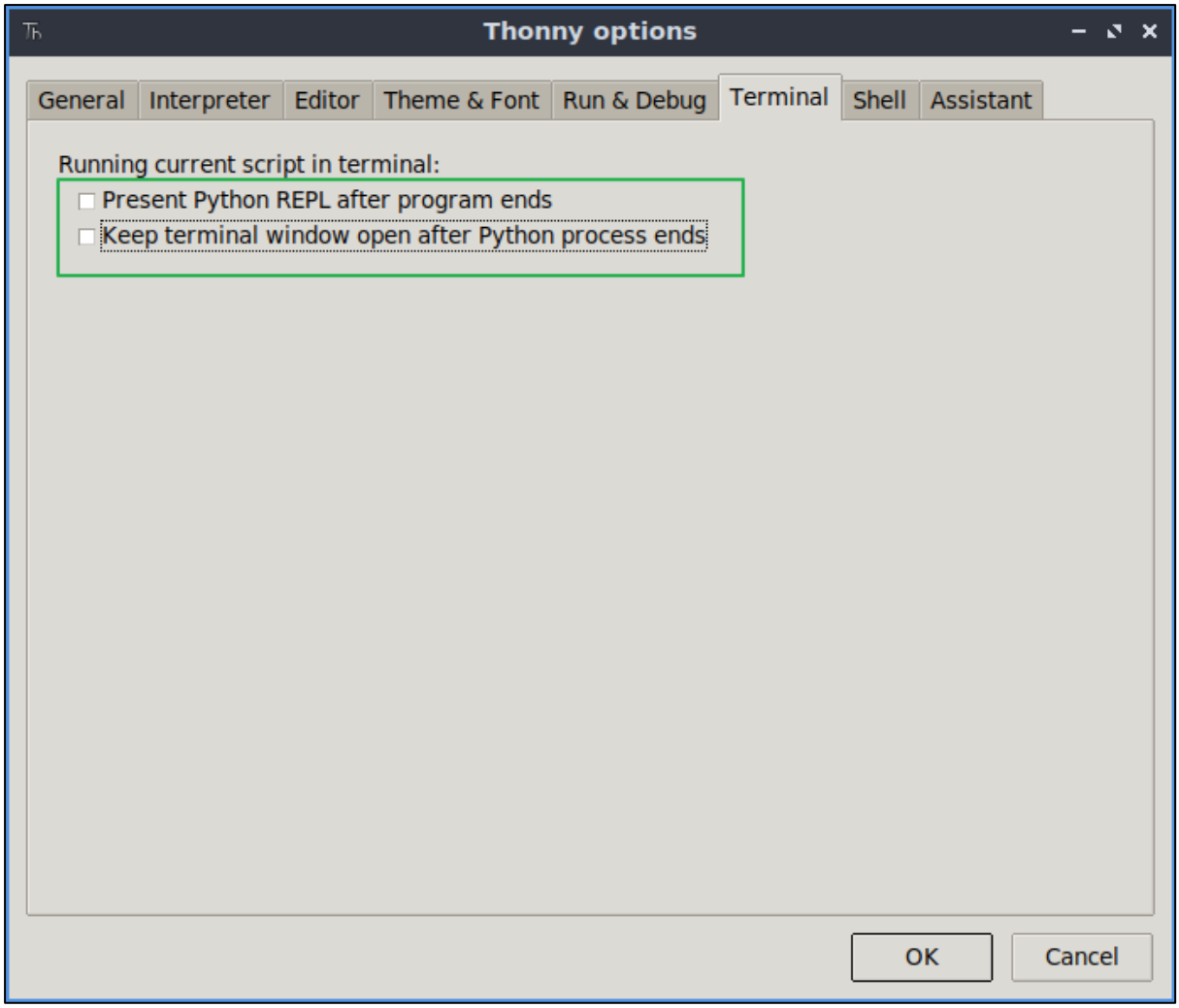
Change to the Editor TAB. The first 2 options can be helpful. I suggest coming back to this TAB once you have some lines of code to work with and try toggling the Highlight matching names and local variables to see which you are most comfortable with.

Set the last item “Recommend maximum line length to 80. This will create a right margin guide to help avoid writing excessively long code lines.



The “Theme and Font” TAB is set to a very usable code highlighting colour for beginners. You can try different code highlighting colours as you become more confident with coding.

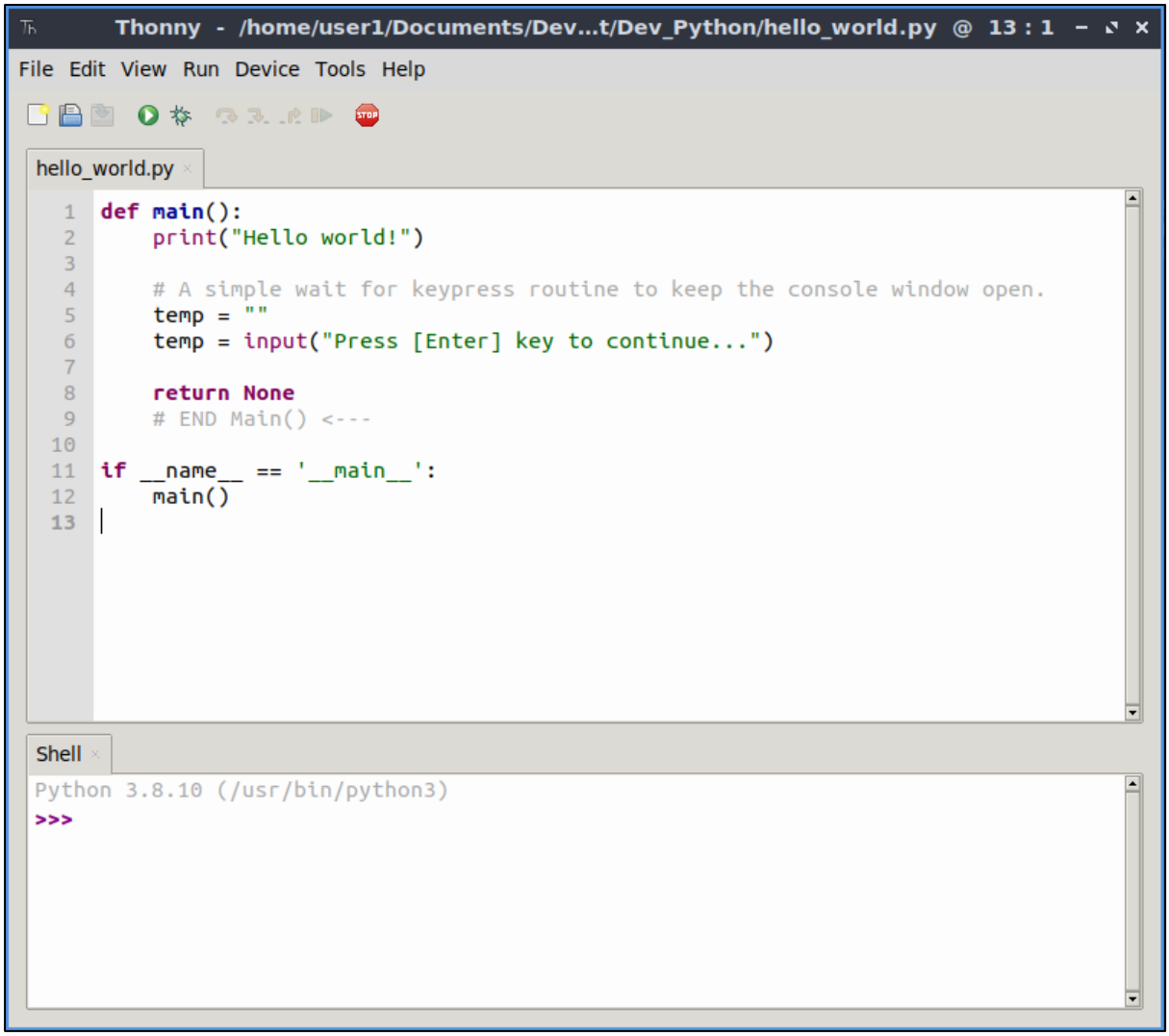
Next select the Terminal TAB. Deselect [ ] both options here. If you leave the “[ ] Keep terminal window open after Python process ends” you will need to close the terminal (Console) window by Typing quit() then [Enter] every time you run an application in a console window. I find it more convenient to place a “wait for keypress” at the end of my scripts as it only requires a single keypress of the spacebar or enter to exit the console. You can see examples of this in “A Beginners Guide To Programming”. Adding a “wait for keypress” also allows you to test your script directly from the command line while allowing the window to stay open so you can see the results.



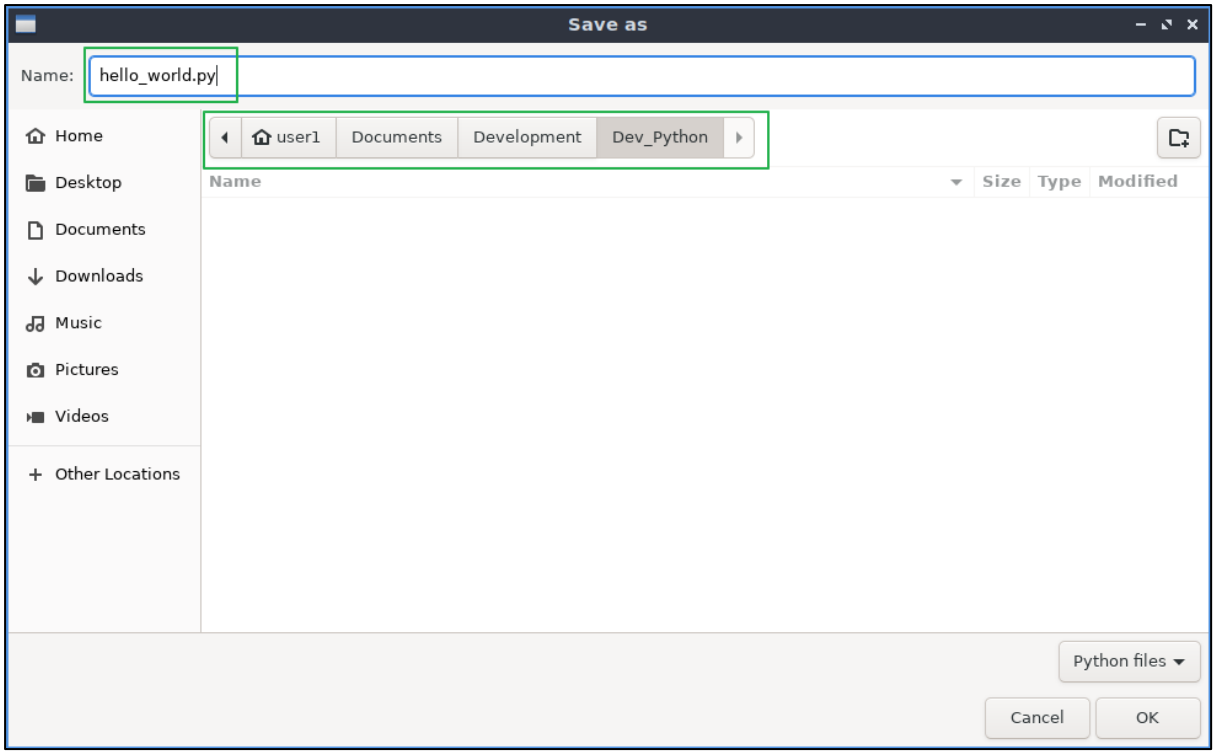
All other settings are fine at the defaults. Press OK to save the changes and exit the Options dialog. You are now ready to start coding.

A new source code document should already be open in Thonny. If not, use File -> New to create a new source document. Enter the following Hello world into the code area.

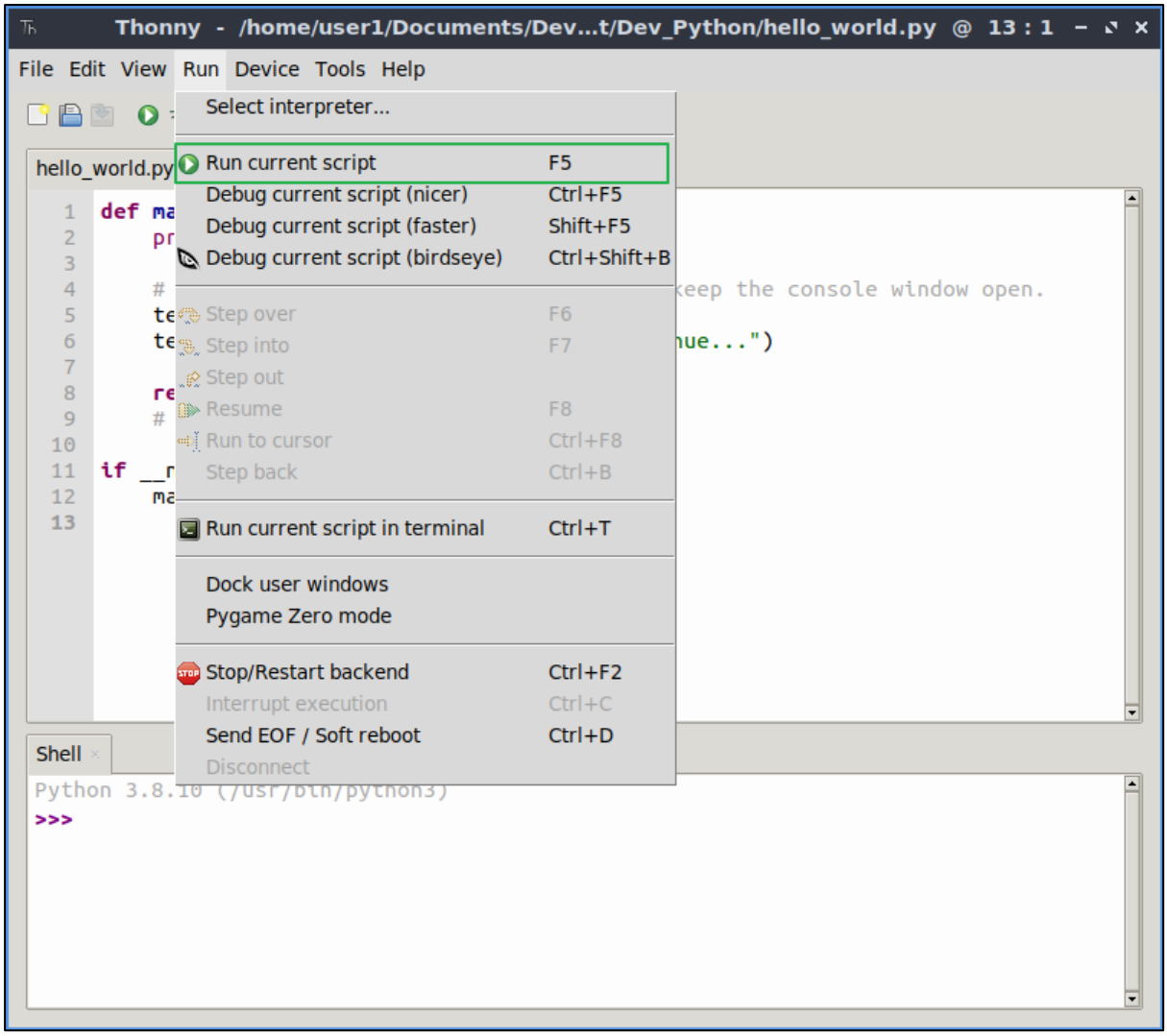
```
' Helloworld.bas

Declare Function main_procedure() As Integer
main_procedure()  ' Call main_procedure

function main_procedure() As Integer  ' Main procedure – “Formal Entry point”
    print "Hello world!"
    print "Press any key to continue..."
    sleep  ' Pause execution so we can view the console output before it closes.
    return 0
end Function
```



Click on save in the toolbar and navigate to **/home/user1/Documents/Development/Dev_Python** and save as “hello_world.py”. An unsaved document will have an asterisks * next to the file name.

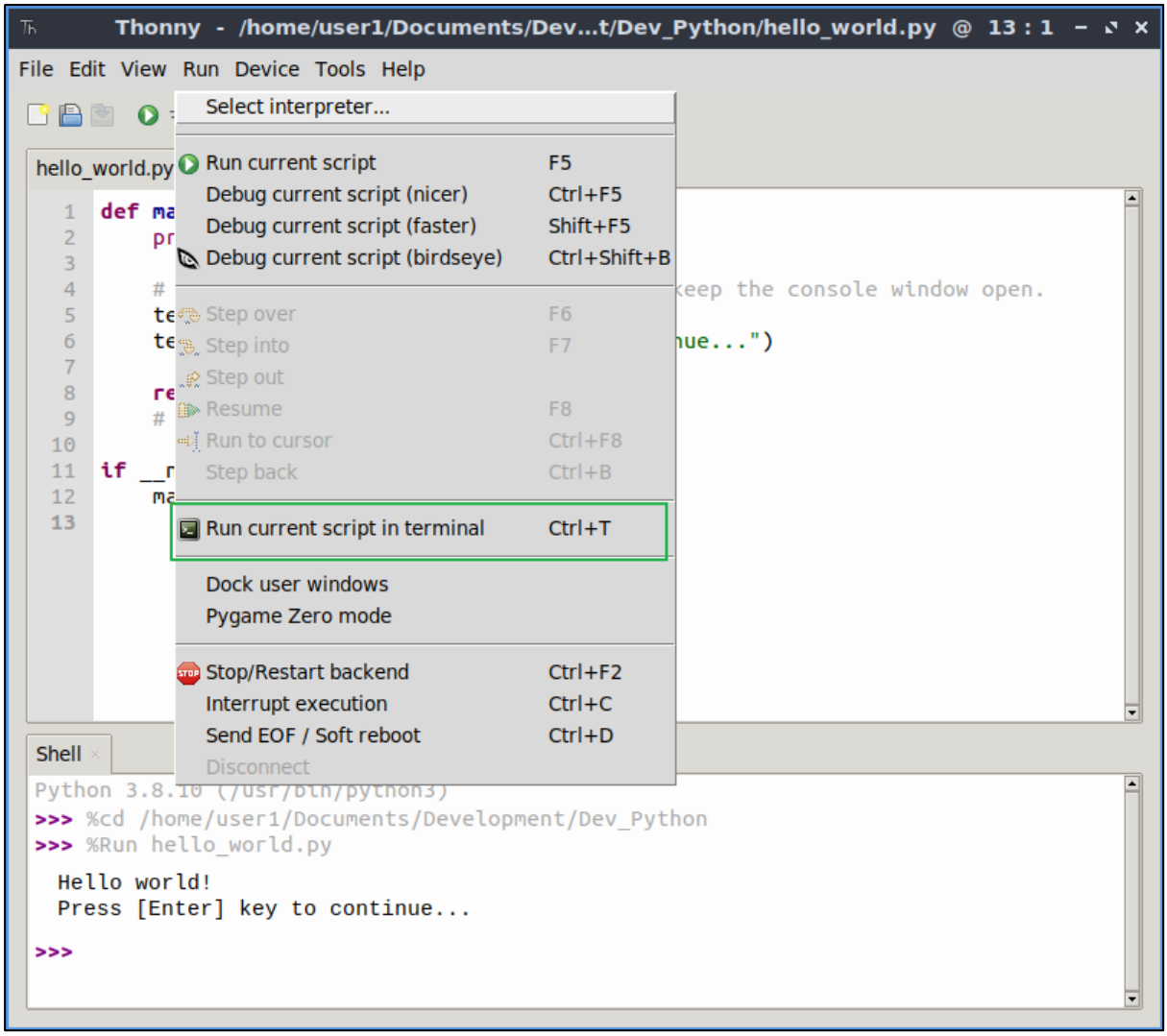


Next select the green run button to test the hello world script in Thonnys' REPL output window under "Shell" in the lower part of the IDE. Or from the Menu bar select Run – Run current script.

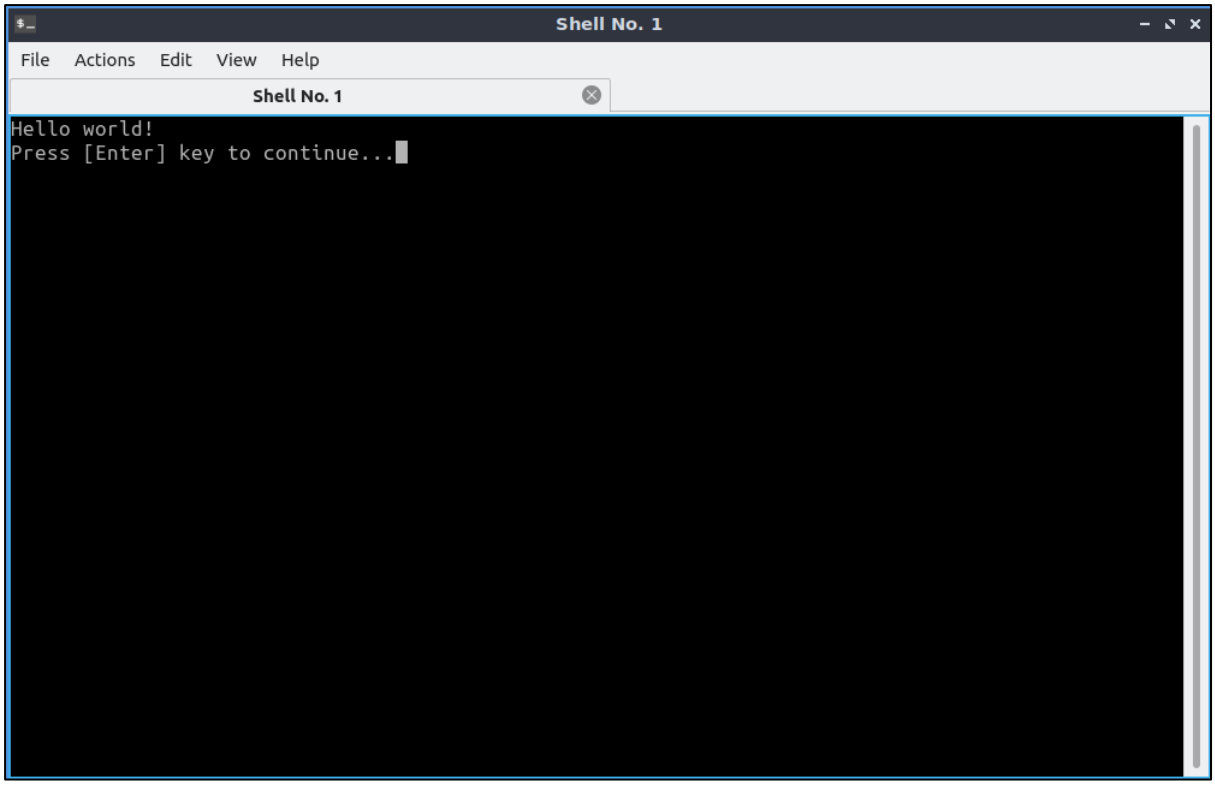


You should see the Hello world! Followed by the Press [Enter] key to continue... in the lower screen.

To test your application in the native console environment, select Run -> Run current script in terminal.

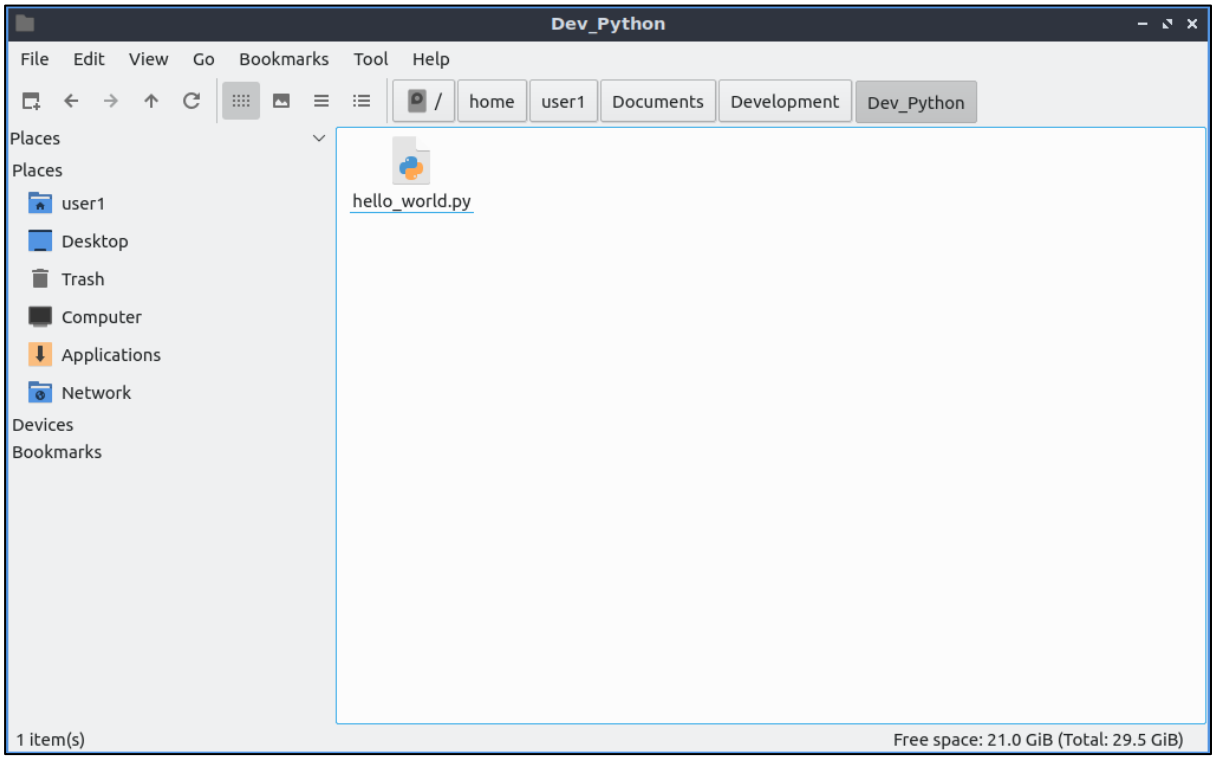


If all has worked well you should see the console window with Hello world! Followed by the Press [Enter] key to continue...



Press enter to close the console window.

If you use the file manager you can navigate to your Python script file.



To run the script file from the command line open a terminal and enter the following.

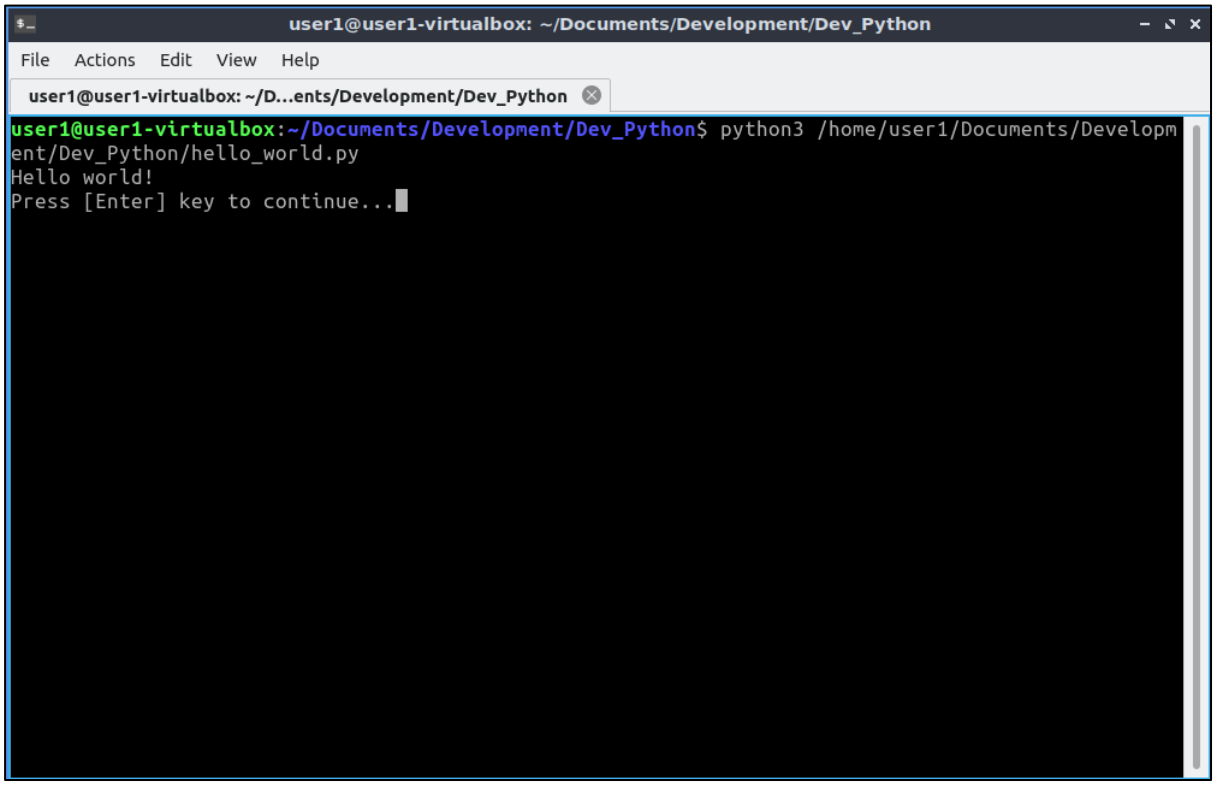
`python3 /home/user1/Documents/Development/Dev_Python/hello_world.py`

Alternatively you can navigate to the script location enter the following into the terminal.

`./hello_world.py`

The "**`./`**" prefix is MACRO required to set the Current Directory of the file and is interpreted the same as the full path above..

You will need to select the Python interpreter followed by the full path to the script.



From here you can create your own source examples to experiment with and test, or try some of the examples from "A Beginners Guide To Programming".

Supplemental

VirtualBox Manual Pages:

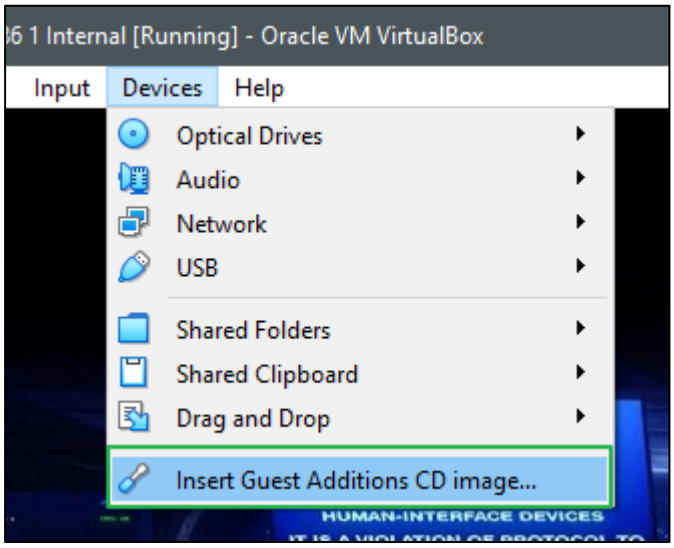
- https://www.virtualbox.org/manual/ch04.html
- https://www.virtualbox.org/manual/ch04.html#additions-windows
- https://www.virtualbox.org/manual/ch04.html#additions-linux
- https://www.virtualbox.org/manual/ch04.html#sharedfolders

Creating VirtualBox “Shared folders”

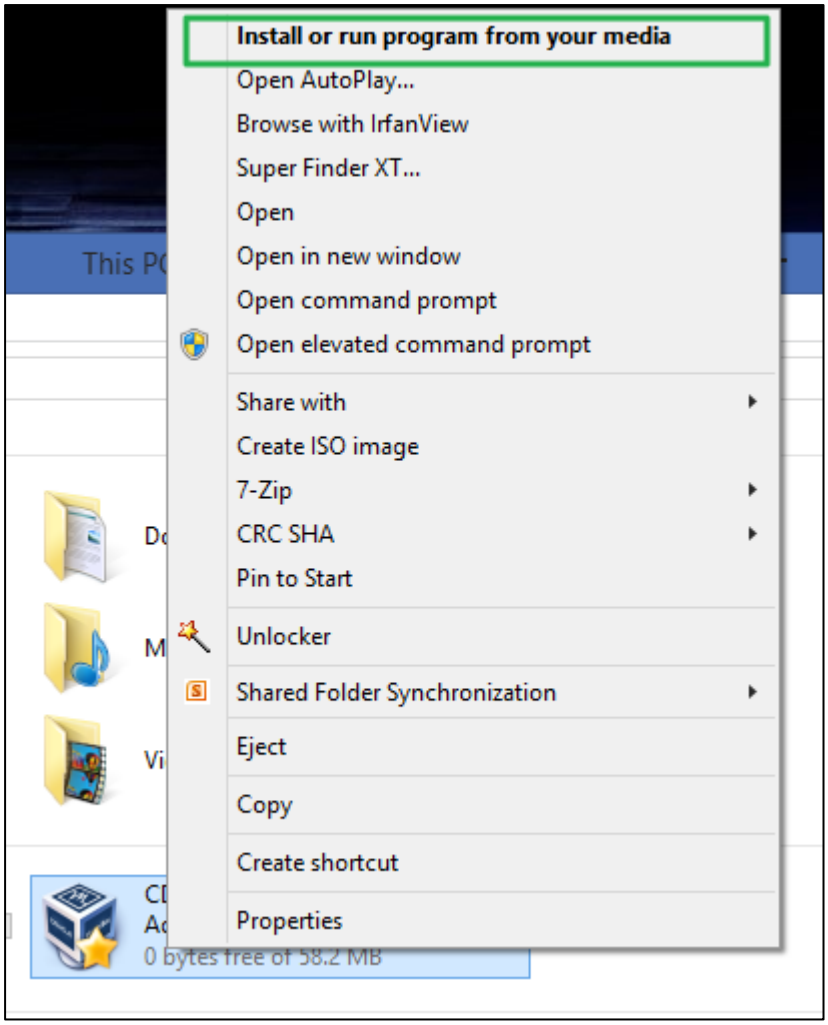
Windows Guest

Add VirtualBox shared folder.

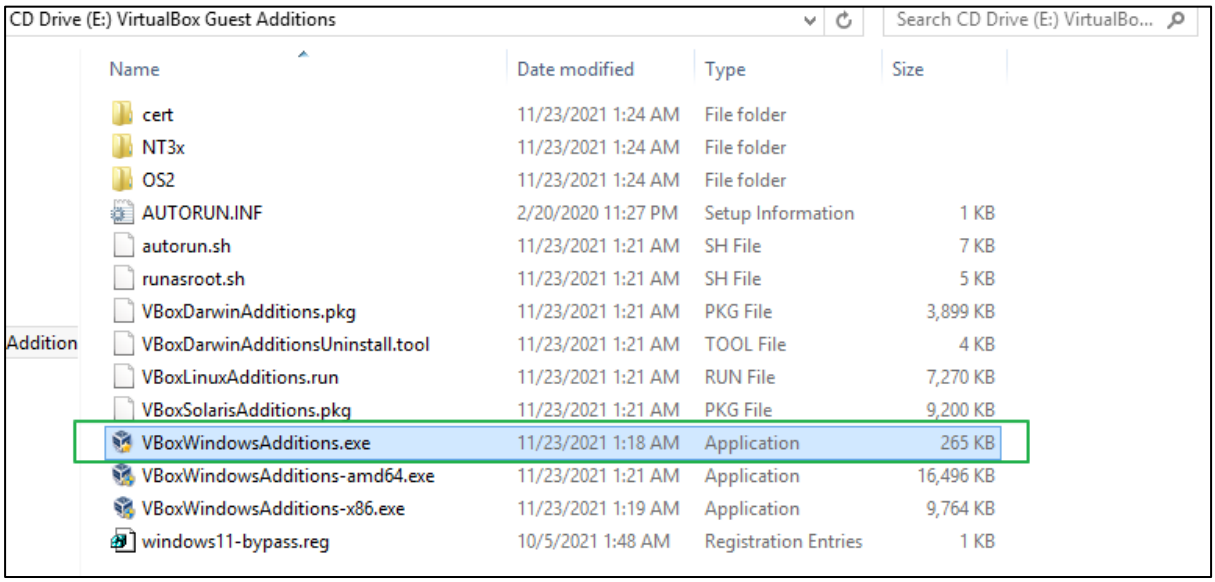
Ensure that you have installed the VirtualBox guest additions from the built in ISO.



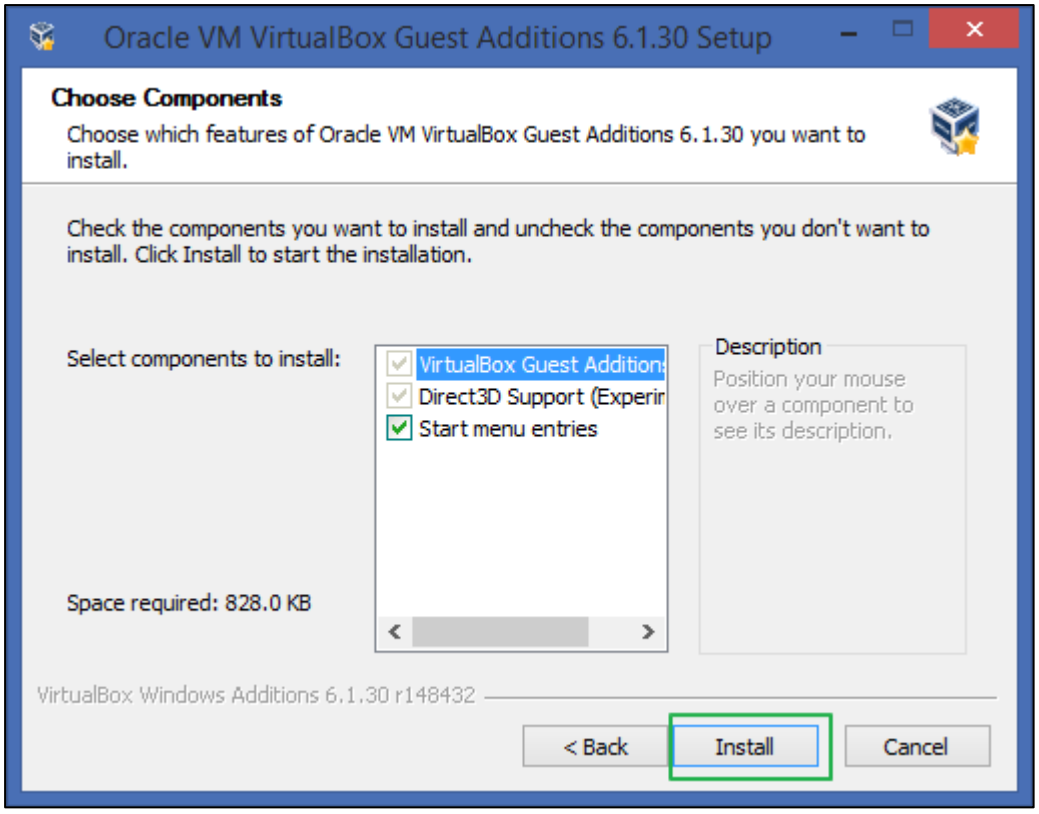
Open Windows Explorer and right click on the CD/DVD drive and select “Install or run program from your media”.



Alternatively open the media drive and run the “VBoxWindowsAdditions.exe”.

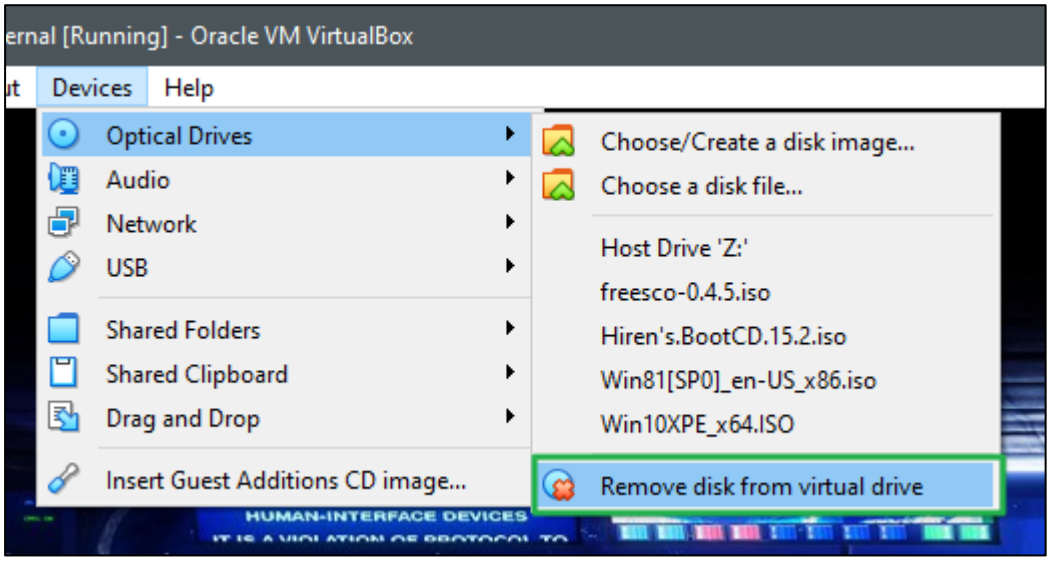


Follow through the setup dialogs and select next for each screen. Leave the default entries as they are.



When finished, restart the VM Client.

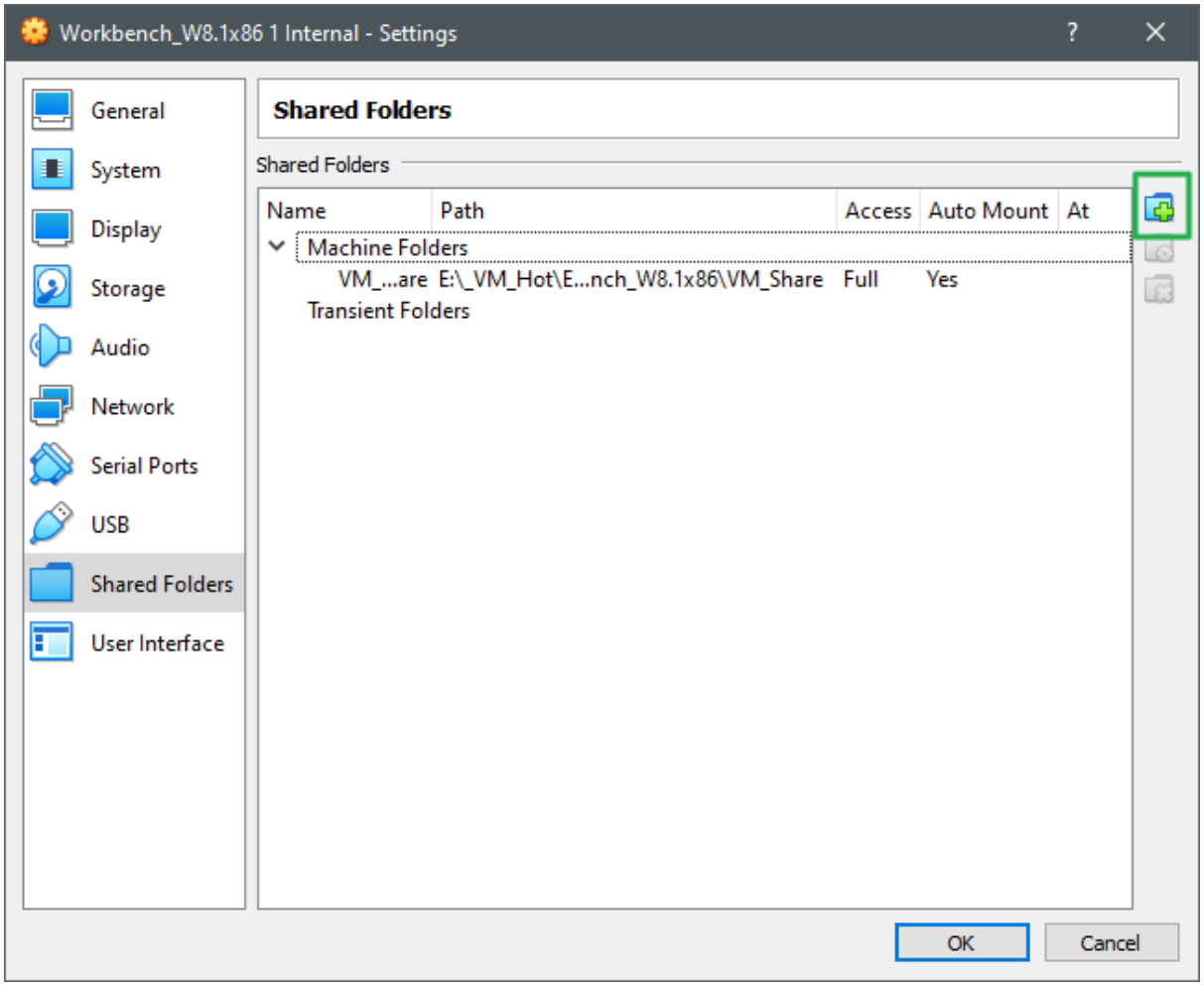
When you have restarted and logged in to Windows desktop remove the VirtualBox guest additions ISO.



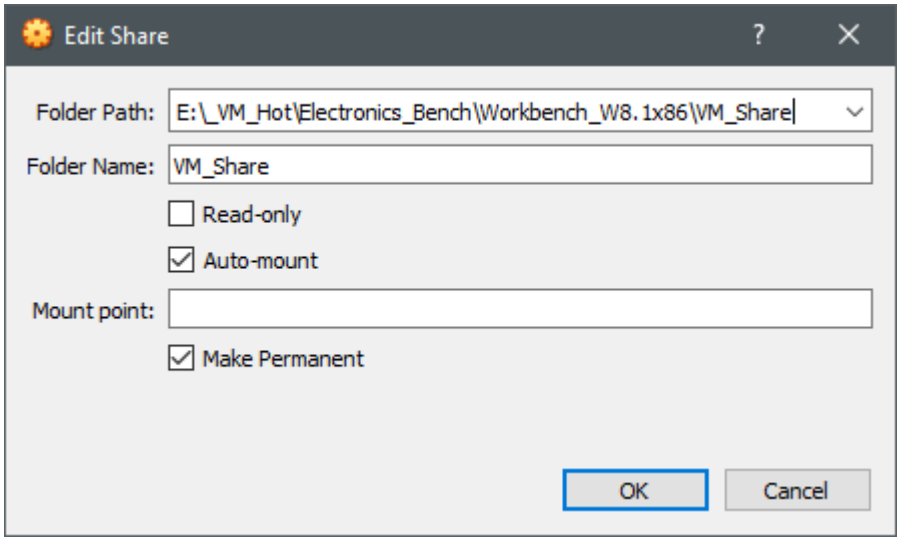
Create the shared folder in VirtualBox.

Open Shared Folder Settings dialog.

Select “Adds new shared folder.”

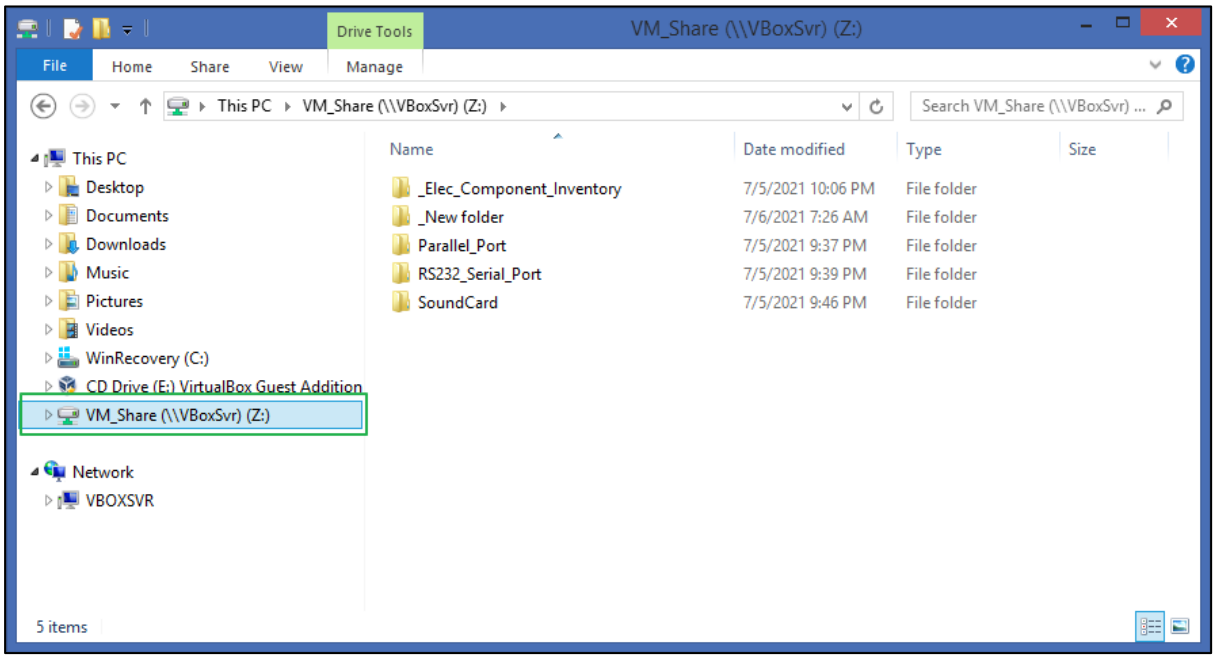


Select the shared folder location on your host machine. I usually create a folder named “Shared” in the directory with the virtual machine files.



Restart the Virtual machine client OS.

Windows will Automatically mount the Shared Folder that you have selected above.



Unbuntu Guest

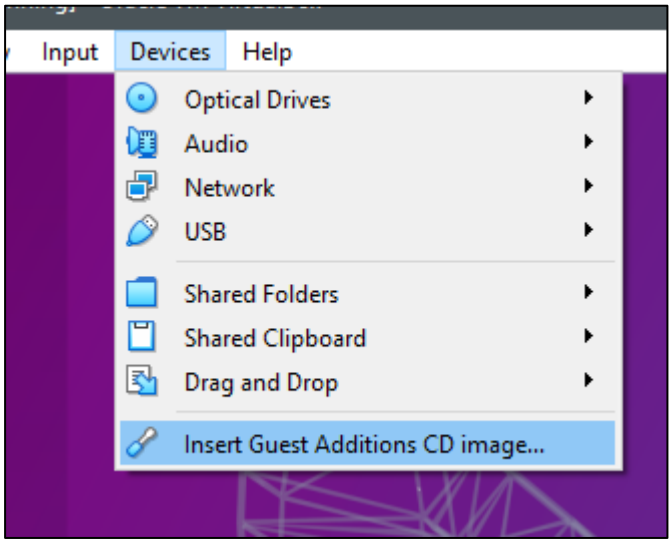
[Fix sf_shared user account permissions issues

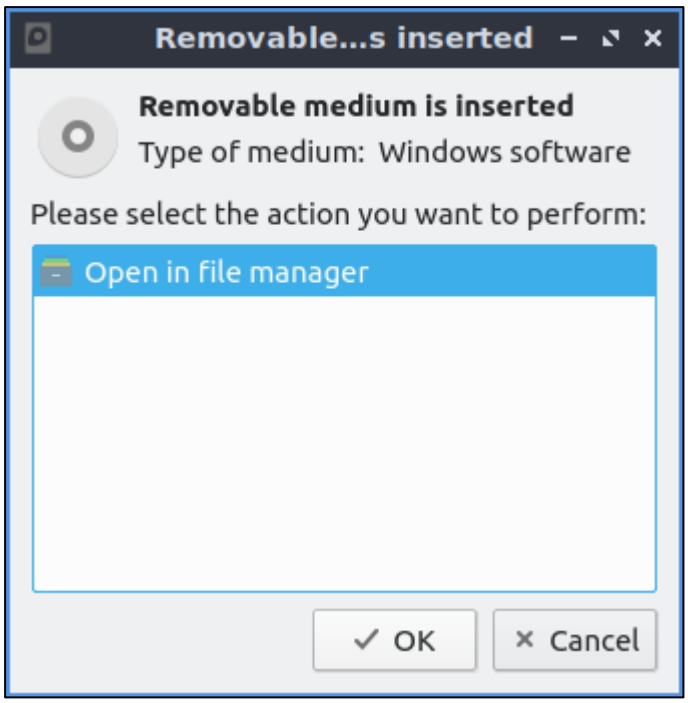
sudo usermod -G vboxsf -a user2 (whoami?)]

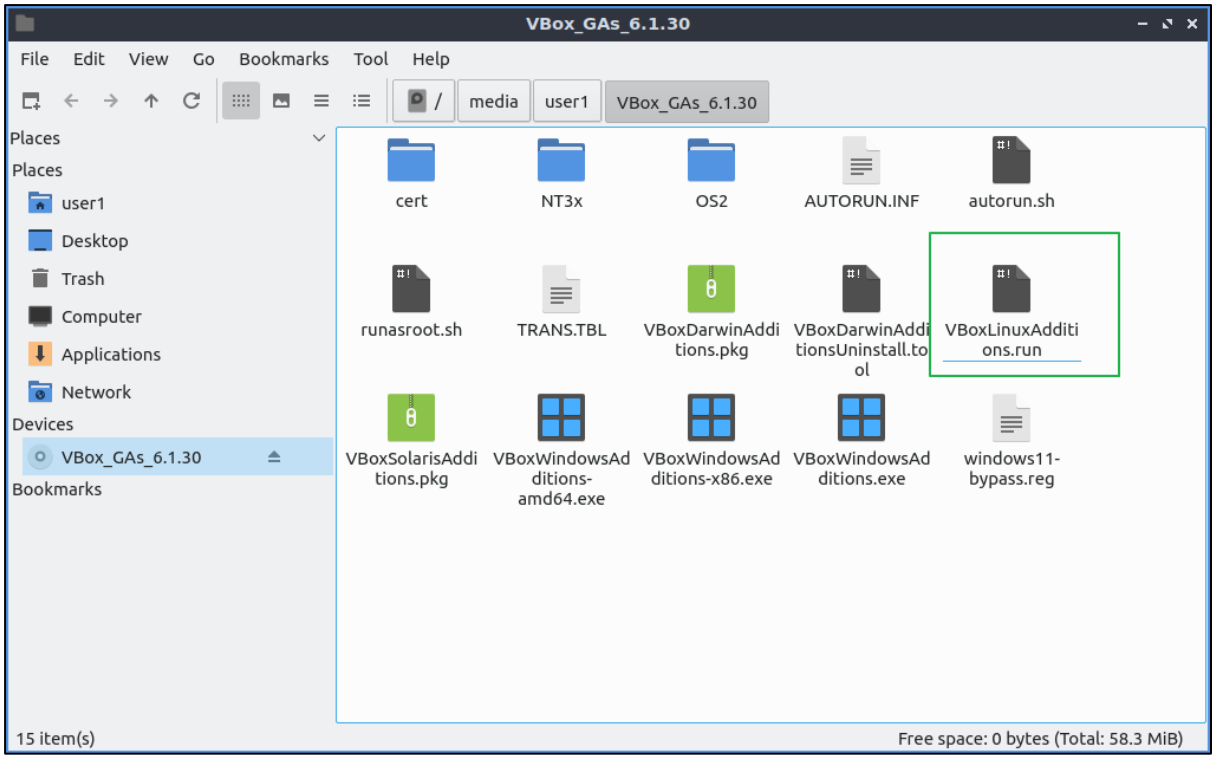
Add VirtualBox shared folder.

Note: Installing Virtuabox Guest Additions can be a little complicated depending upon the underlying Host OS version, the Guest OS version and the

Ensure that you have installed the VirtualBox guest additions from the built in ISO.







Open a terminal from the file manager or navigate to **`/media/user1/VBox_GAs_6.1.30`**.

In Linux some additional steps are required to install the guest additions from the built in ISO.

Type **sudo apt update** [Enter]

Type **sudo apt install build-essential** [Enter]

The following may not be required.

Type **`sudo apt install virtualbox-guest-utils virtualbox-guest-dkms`** [Enter]

```
user1@user1-virtualbox:/media/user1/VBox_GAs_6.1.34$ sudo apt install virtualbox-guest-utils
virtualbox-guest-dkms
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are no longer required:
  libfwupdplugin1 libllvm10 libqt5opengl5 qtgstreamer-plugins-qt5
Use 'sudo apt autoremove' to remove them.
The following additional packages will be installed:
  binutils binutils-common binutils-x86-64-linux-gnu build-essential
  dctrl-tools dkms dpkg-dev fakeroot g++ g++-9 gcc gcc-9
  libalgorithm-diff-perl libalgorithm-diff-xs-perl libalgorithm-merge-perl
  libasan5 libatomic1 libbinutils libc-dev-bin libc6-dev libcc1-0
  libcrypt-dev libctf-nobfd0 libctf0 libdpkg-perl libfakeroot
  libfile-fcntllock-perl libgcc-9-dev libitm1 liblsan0 libquadmath0
  libstdc++-9-dev libtsan0 libubsan1 linux-libc-dev make manpages-dev
Suggested packages:
  binutils-doc debtags menu debian-keyring g++-multilib g++-9-multilib
```

```
  gcc-9-doc gcc-multilib autoconf automake libtool flex bison gdb gcc-doc
  gcc-9-multilib gcc-9-locales glibc-doc bzr libstdc++-9-doc make-doc
  virtualbox-guest-x11
The following NEW packages will be installed:
  binutils binutils-common binutils-x86-64-linux-gnu build-essential
  dctrl-tools dkms dpkg-dev fakeroot g++ g++-9 gcc gcc-9
  libalgorithm-diff-perl libalgorithm-diff-xs-perl libalgorithm-merge-perl
  libasan5 libatomic1 libbinutils libc-dev-bin libc6-dev libcc1-0
  libcrypt-dev libctf-nobfd0 libctf0 libdpkg-perl libfakeroot
  libfile-fcntllock-perl libgcc-9-dev libitm1 liblsan0 libquadmath0
  libstdc++-9-dev libtsan0 libubsan1 linux-libc-dev make manpages-dev
  virtualbox-guest-dkms virtualbox-guest-utils
0 to upgrade, 39 to newly install, 0 to remove and 0 not to upgrade.
Need to get 38.8 MB of archives.
After this operation, 187 MB of additional disk space will be used.
Do you want to continue? [Y/n] y
Get:1 http://au.archive.ubuntu.com/ubuntu focal-updates/main amd64 libcc1-0 amd64 10.3.0-
1ubuntu1~20.04 [48.8 kB]
Get:2 http://au.archive.ubuntu.com/ubuntu focal-updates/main amd64 binutils-common amd64
2.34-6ubuntu1.3 [207 kB]
Get:3 http://au.archive.ubuntu.com/ubuntu focal-updates/main amd64 libbinutils amd64 2.34-
6ubuntu1.3 [474 kB]
Get:4 http://au.archive.ubuntu.com/ubuntu focal-updates/main amd64 libctf-nobfd0 amd64 2.34-
6ubuntu1.3 [47.4 kB]
Get:5 http://au.archive.ubuntu.com/ubuntu focal-updates/main amd64 libctf0 amd64 2.34-
6ubuntu1.3 [46.6 kB]
Get:6 http://au.archive.ubuntu.com/ubuntu focal-updates/main amd64 binutils-x86-64-linux-gnu
amd64 2.34-6ubuntu1.3 [1,613 kB]
Get:7 http://au.archive.ubuntu.com/ubuntu focal-updates/main amd64 binutils amd64 2.34-
6ubuntu1.3 [3,380 B]
Get:8 http://au.archive.ubuntu.com/ubuntu focal-updates/main amd64 libitm1 amd64 10.3.0-
1ubuntu1~20.04 [26.2 kB]
Get:9 http://au.archive.ubuntu.com/ubuntu focal-updates/main amd64 libatomic1 amd64 10.3.0-
1ubuntu1~20.04 [9,284 B]
Get:10 http://au.archive.ubuntu.com/ubuntu focal-updates/main amd64 libasan5 amd64 9.4.0-
1ubuntu1~20.04.1 [2,751 kB]
Get:11 http://au.archive.ubuntu.com/ubuntu focal-updates/main amd64 liblsan0 amd64 10.3.0-
1ubuntu1~20.04 [835 kB]
Get:12 http://au.archive.ubuntu.com/ubuntu focal-updates/main amd64 libtsan0 amd64 10.3.0-
1ubuntu1~20.04 [2,009 kB]
Get:13 http://au.archive.ubuntu.com/ubuntu focal-updates/main amd64 libubsan1 amd64 10.3.0-
1ubuntu1~20.04 [784 kB]
Get:14 http://au.archive.ubuntu.com/ubuntu focal-updates/main amd64 libquadmath0 amd64
10.3.0-1ubuntu1~20.04 [146 kB]
Get:15 http://au.archive.ubuntu.com/ubuntu focal-updates/main amd64 libgcc-9-dev amd64 9.4.0-
1ubuntu1~20.04.1 [2,359 kB]
Get:16 http://au.archive.ubuntu.com/ubuntu focal-updates/main amd64 gcc-9 amd64 9.4.0-
1ubuntu1~20.04.1 [8,274 kB]
Get:17 http://au.archive.ubuntu.com/ubuntu focal/main amd64 gcc amd64 4:9.3.0-1ubuntu2 [5,208
B]
Get:18 http://au.archive.ubuntu.com/ubuntu focal-updates/main amd64 libdpkg-perl all
```

```
1.19.7ubuntu3.2 [231 kB]
Get:19 http://au.archive.ubuntu.com/ubuntu focal/main amd64 make amd64 4.2.1-1.2 [162 kB]
Get:20 http://au.archive.ubuntu.com/ubuntu focal-updates/main amd64 dpkg-dev all
1.19.7ubuntu3.2 [679 kB]
Get:21 http://au.archive.ubuntu.com/ubuntu focal-updates/main amd64 libc-dev-bin amd64 2.31-
0ubuntu9.9 [71.8 kB]
Get:22 http://au.archive.ubuntu.com/ubuntu focal-updates/main amd64 linux-libc-dev amd64 5.4.0-
126.142 [1,118 kB]
Get:23 http://au.archive.ubuntu.com/ubuntu focal/main amd64 libcrypt-dev amd64 1:4.4.10-
10ubuntu4 [104 kB]
Get:24 http://au.archive.ubuntu.com/ubuntu focal-updates/main amd64 libc6-dev amd64 2.31-
0ubuntu9.9 [2,519 kB]
Get:25 http://au.archive.ubuntu.com/ubuntu focal-updates/main amd64 libstdc++-9-dev amd64
9.4.0-1ubuntu1~20.04.1 [1,722 kB]
Get:26 http://au.archive.ubuntu.com/ubuntu focal-updates/main amd64 g++-9 amd64 9.4.0-
1ubuntu1~20.04.1 [8,420 kB]
Get:27 http://au.archive.ubuntu.com/ubuntu focal/main amd64 g++ amd64 4:9.3.0-1ubuntu2 [1,604
B]
Get:28 http://au.archive.ubuntu.com/ubuntu focal-updates/main amd64 build-essential amd64
12.8ubuntu1.1 [4,664 B]
Get:29 http://au.archive.ubuntu.com/ubuntu focal/main amd64 dctrl-tools amd64 2.24-3 [61.5 kB]
Get:30 http://au.archive.ubuntu.com/ubuntu focal-updates/main amd64 dkms all 2.8.1-5ubuntu2
[66.8 kB]
Get:31 http://au.archive.ubuntu.com/ubuntu focal/main amd64 libfakeroot amd64 1.24-1 [25.7 kB]
Get:32 http://au.archive.ubuntu.com/ubuntu focal/main amd64 fakeroot amd64 1.24-1 [62.6 kB]
Get:33 http://au.archive.ubuntu.com/ubuntu focal/main amd64 libalgorithm-diff-perl all 1.19.03-2
[46.6 kB]
Get:34 http://au.archive.ubuntu.com/ubuntu focal/main amd64 libalgorithm-diff-xs-perl amd64
0.04-6 [11.3 kB]
Get:35 http://au.archive.ubuntu.com/ubuntu focal/main amd64 libalgorithm-merge-perl all 0.08-3
[12.0 kB]
Get:36 http://au.archive.ubuntu.com/ubuntu focal/main amd64 libfile-fcntllock-perl amd64 0.22-
3build4 [33.1 kB]
Get:37 http://au.archive.ubuntu.com/ubuntu focal/main amd64 manpages-dev all 5.05-1 [2,266 kB]
Get:38 http://au.archive.ubuntu.com/ubuntu focal-updates/multiverse amd64 virtualbox-guest-
dkms all 6.1.38-dfsg-3~ubuntu1.20.04.1 [676 kB]
Get:39 http://au.archive.ubuntu.com/ubuntu focal-updates/multiverse amd64 virtualbox-guest-utils
amd64 6.1.38-dfsg-3~ubuntu1.20.04.1 [889 kB]
Fetched 38.8 MB in 7s (5,699 kB/s)
Extracting templates from packages: 100%
Selecting previously unselected package libcc1-0:amd64.
(Reading database ... 266103 files and directories currently installed.)
Preparing to unpack .../00-libcc1-0_10.3.0-1ubuntu1~20.04_amd64.deb ...
Unpacking libcc1-0:amd64 (10.3.0-1ubuntu1~20.04) ...
Selecting previously unselected package binutils-common:amd64.
Preparing to unpack .../01-binutils-common_2.34-6ubuntu1.3_amd64.deb ...
Unpacking binutils-common:amd64 (2.34-6ubuntu1.3) ...
Selecting previously unselected package libbinutils:amd64.
Preparing to unpack .../02-libbinutils_2.34-6ubuntu1.3_amd64.deb ...
Unpacking libbinutils:amd64 (2.34-6ubuntu1.3) ...
Selecting previously unselected package libctf-nobfd0:amd64.
```

```
Preparing to unpack .../03-libctf-nobfd0_2.34-6ubuntu1.3_amd64.deb ...
Unpacking libctf-nobfd0:amd64 (2.34-6ubuntu1.3) ...
Selecting previously unselected package libctf0:amd64.
Preparing to unpack .../04-libctf0_2.34-6ubuntu1.3_amd64.deb ...
Unpacking libctf0:amd64 (2.34-6ubuntu1.3) ...
Selecting previously unselected package binutils-x86-64-linux-gnu.
Preparing to unpack .../05-binutils-x86-64-linux-gnu_2.34-6ubuntu1.3_amd64.deb ...
Unpacking binutils-x86-64-linux-gnu (2.34-6ubuntu1.3) ...
Selecting previously unselected package binutils.
Preparing to unpack .../06-binutils_2.34-6ubuntu1.3_amd64.deb ...
Unpacking binutils (2.34-6ubuntu1.3) ...
Selecting previously unselected package libitm1:amd64.
Preparing to unpack .../07-libitm1_10.3.0-1ubuntu1~20.04_amd64.deb ...
Unpacking libitm1:amd64 (10.3.0-1ubuntu1~20.04) ...
Selecting previously unselected package libatomic1:amd64.
Preparing to unpack .../08-libatomic1_10.3.0-1ubuntu1~20.04_amd64.deb ...
Unpacking libatomic1:amd64 (10.3.0-1ubuntu1~20.04) ...
Selecting previously unselected package libasan5:amd64.
Preparing to unpack .../09-libasan5_9.4.0-1ubuntu1~20.04.1_amd64.deb ...
Unpacking libasan5:amd64 (9.4.0-1ubuntu1~20.04.1) ...
Selecting previously unselected package liblsan0:amd64.
Preparing to unpack .../10-liblsan0_10.3.0-1ubuntu1~20.04_amd64.deb ...
Unpacking liblsan0:amd64 (10.3.0-1ubuntu1~20.04) ...
Selecting previously unselected package libtsan0:amd64.
Preparing to unpack .../11-libtsan0_10.3.0-1ubuntu1~20.04_amd64.deb ...
Unpacking libtsan0:amd64 (10.3.0-1ubuntu1~20.04) ...
Selecting previously unselected package libubsan1:amd64.
Preparing to unpack .../12-libubsan1_10.3.0-1ubuntu1~20.04_amd64.deb ...
Unpacking libubsan1:amd64 (10.3.0-1ubuntu1~20.04) ...
Selecting previously unselected package libquadmath0:amd64.
Preparing to unpack .../13-libquadmath0_10.3.0-1ubuntu1~20.04_amd64.deb ...
Unpacking libquadmath0:amd64 (10.3.0-1ubuntu1~20.04) ...
Selecting previously unselected package libgcc-9-dev:amd64.
Preparing to unpack .../14-libgcc-9-dev_9.4.0-1ubuntu1~20.04.1_amd64.deb ...
Unpacking libgcc-9-dev:amd64 (9.4.0-1ubuntu1~20.04.1) ...
Selecting previously unselected package gcc-9.
Preparing to unpack .../15-gcc-9_9.4.0-1ubuntu1~20.04.1_amd64.deb ...
Unpacking gcc-9 (9.4.0-1ubuntu1~20.04.1) ...
Selecting previously unselected package gcc.
Preparing to unpack .../16-gcc_4%3a9.3.0-1ubuntu2_amd64.deb ...
Unpacking gcc (4:9.3.0-1ubuntu2) ...
Selecting previously unselected package libdpkg-perl.
Preparing to unpack .../17-libdpkg-perl_1.19.7ubuntu3.2_all.deb ...
Unpacking libdpkg-perl (1.19.7ubuntu3.2) ...
Selecting previously unselected package make.
Preparing to unpack .../18-make_4.2.1-1.2_amd64.deb ...
Unpacking make (4.2.1-1.2) ...
Selecting previously unselected package dpkg-dev.
Preparing to unpack .../19-dpkg-dev_1.19.7ubuntu3.2_all.deb ...
Unpacking dpkg-dev (1.19.7ubuntu3.2) ...
Selecting previously unselected package libc-dev-bin.
```

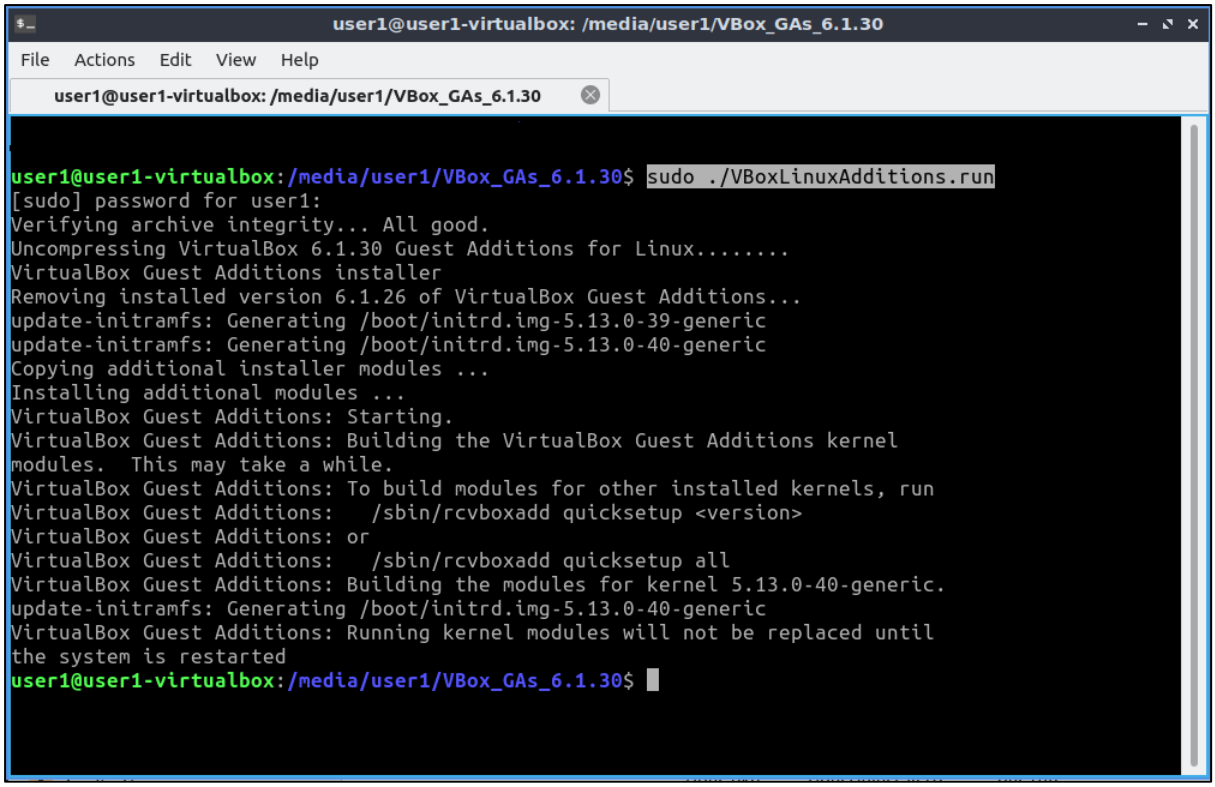
```
Preparing to unpack .../20-libc-dev-bin_2.31-0ubuntu9.9_amd64.deb ...
Unpacking libc-dev-bin (2.31-0ubuntu9.9) ...
Selecting previously unselected package linux-libc-dev:amd64.
Preparing to unpack .../21-linux-libc-dev_5.4.0-126.142_amd64.deb ...
Unpacking linux-libc-dev:amd64 (5.4.0-126.142) ...
Selecting previously unselected package libcrypt-dev:amd64.
Preparing to unpack .../22-libcrypt-dev_1%3a4.4.10-10ubuntu4_amd64.deb ...
Unpacking libcrypt-dev:amd64 (1:4.4.10-10ubuntu4) ...
Selecting previously unselected package libc6-dev:amd64.
Preparing to unpack .../23-libc6-dev_2.31-0ubuntu9.9_amd64.deb ...
Unpacking libc6-dev:amd64 (2.31-0ubuntu9.9) ...
Selecting previously unselected package libstdc++-9-dev:amd64.
Preparing to unpack .../24-libstdc++-9-dev_9.4.0-1ubuntu1~20.04.1_amd64.deb ...
Unpacking libstdc++-9-dev:amd64 (9.4.0-1ubuntu1~20.04.1) ...
Selecting previously unselected package g++-9.
Preparing to unpack .../25-g++-9_9.4.0-1ubuntu1~20.04.1_amd64.deb ...
Unpacking g++-9 (9.4.0-1ubuntu1~20.04.1) ...
Selecting previously unselected package g++.
Preparing to unpack .../26-g++_4%3a9.3.0-1ubuntu2_amd64.deb ...
Unpacking g++ (4:9.3.0-1ubuntu2) ...
Selecting previously unselected package build-essential.
Preparing to unpack .../27-build-essential_12.8ubuntu1.1_amd64.deb ...
Unpacking build-essential (12.8ubuntu1.1) ...
Selecting previously unselected package dctrl-tools.
Preparing to unpack .../28-dctrl-tools_2.24-3_amd64.deb ...
Unpacking dctrl-tools (2.24-3) ...
Selecting previously unselected package dkms.
Preparing to unpack .../29-dkms_2.8.1-5ubuntu2_all.deb ...
Unpacking dkms (2.8.1-5ubuntu2) ...
Selecting previously unselected package libfakeroot:amd64.
Preparing to unpack .../30-libfakeroot_1.24-1_amd64.deb ...
Unpacking libfakeroot:amd64 (1.24-1) ...
Selecting previously unselected package fakeroot.
Preparing to unpack .../31-fakeroot_1.24-1_amd64.deb ...
Unpacking fakeroot (1.24-1) ...
Selecting previously unselected package libalgorithm-diff-perl.
Preparing to unpack .../32-libalgorithm-diff-perl_1.19.03-2_all.deb ...
Unpacking libalgorithm-diff-perl (1.19.03-2) ...
Selecting previously unselected package libalgorithm-diff-xs-perl.
Preparing to unpack .../33-libalgorithm-diff-xs-perl_0.04-6_amd64.deb ...
Unpacking libalgorithm-diff-xs-perl (0.04-6) ...
Selecting previously unselected package libalgorithm-merge-perl.
Preparing to unpack .../34-libalgorithm-merge-perl_0.08-3_all.deb ...
Unpacking libalgorithm-merge-perl (0.08-3) ...
Selecting previously unselected package libfile-fcntllock-perl.
Preparing to unpack .../35-libfile-fcntllock-perl_0.22-3build4_amd64.deb ...
Unpacking libfile-fcntllock-perl (0.22-3build4) ...
Selecting previously unselected package manpages-dev.
Preparing to unpack .../36-manpages-dev_5.05-1_all.deb ...
Unpacking manpages-dev (5.05-1) ...
Selecting previously unselected package virtualbox-guest-dkms.
```

```
Preparing to unpack .../37-virtualbox-guest-dkms_6.1.38-dfsg-3~ubuntu1.20.04.1_all.deb ...
Unpacking virtualbox-guest-dkms (6.1.38-dfsg-3~ubuntu1.20.04.1) ...
Selecting previously unselected package virtualbox-guest-utils.
Preparing to unpack .../38-virtualbox-guest-utils_6.1.38-dfsg-3~ubuntu1.20.04.1_amd64.deb ...
Unpacking virtualbox-guest-utils (6.1.38-dfsg-3~ubuntu1.20.04.1) ...
Setting up manpages-dev (5.05-1) ...
Setting up libfile-fcntllock-perl (0.22-3build4) ...
Setting up libalgorithm-diff-perl (1.19.03-2) ...
Setting up binutils-common:amd64 (2.34-6ubuntu1.3) ...
Setting up linux-libc-dev:amd64 (5.4.0-126.142) ...
Setting up libctf-nobfd0:amd64 (2.34-6ubuntu1.3) ...
Setting up libfakeroot:amd64 (1.24-1) ...
Setting up fakeroot (1.24-1) ...
update-alternatives: using /usr/bin/fakeroot-sysv to provide /usr/bin/fakeroot (fakeroot) in auto
mode
Setting up libasan5:amd64 (9.4.0-1ubuntu1~20.04.1) ...
Setting up virtualbox-guest-utils (6.1.38-dfsg-3~ubuntu1.20.04.1) ...
Created symlink /etc/systemd/system/multi-user.target.wants/virtualbox-guest-utils.service →
/lib/systemd/system/virtualbox-guest-utils.service.
Setting up make (4.2.1-1.2) ...
Setting up libquadmath0:amd64 (10.3.0-1ubuntu1~20.04) ...
Setting up libatomic1:amd64 (10.3.0-1ubuntu1~20.04) ...
Setting up libdpkg-perl (1.19.7ubuntu3.2) ...
Setting up libubsan1:amd64 (10.3.0-1ubuntu1~20.04) ...
Setting up libcrypt-dev:amd64 (1:4.4.10-10ubuntu4) ...
Setting up libbinutils:amd64 (2.34-6ubuntu1.3) ...
Setting up libc-dev-bin (2.31-0ubuntu9.9) ...
Setting up libalgorithm-diff-xs-perl (0.04-6) ...
Setting up libcc1-0:amd64 (10.3.0-1ubuntu1~20.04) ...
Setting up liblsan0:amd64 (10.3.0-1ubuntu1~20.04) ...
Setting up dctrl-tools (2.24-3) ...
Setting up libitm1:amd64 (10.3.0-1ubuntu1~20.04) ...
Setting up libalgorithm-merge-perl (0.08-3) ...
Setting up libtsan0:amd64 (10.3.0-1ubuntu1~20.04) ...
Setting up libctf0:amd64 (2.34-6ubuntu1.3) ...
Setting up libgcc-9-dev:amd64 (9.4.0-1ubuntu1~20.04.1) ...
Setting up libc6-dev:amd64 (2.31-0ubuntu9.9) ...
Setting up binutils-x86-64-linux-gnu (2.34-6ubuntu1.3) ...
Setting up libstdc++-9-dev:amd64 (9.4.0-1ubuntu1~20.04.1) ...
Setting up binutils (2.34-6ubuntu1.3) ...
Setting up dpkg-dev (1.19.7ubuntu3.2) ...
Setting up gcc-9 (9.4.0-1ubuntu1~20.04.1) ...
Setting up gcc (4:9.3.0-1ubuntu2) ...
Setting up dkms (2.8.1-5ubuntu2) ...
Setting up g++-9 (9.4.0-1ubuntu1~20.04.1) ...
Setting up g++ (4:9.3.0-1ubuntu2) ...
update-alternatives: using /usr/bin/g++ to provide /usr/bin/c++ (c++) in auto mode
Setting up build-essential (12.8ubuntu1.1) ...
Setting up virtualbox-guest-dkms (6.1.38-dfsg-3~ubuntu1.20.04.1) ...
Loading new virtualbox-guest-6.1.38 DKMS files...
Building for 5.4.0-126-generic
```

```
Building initial module for 5.4.0-126-generic
Done.

vboxguest.ko:
Running module version sanity check.
 - Original module
   - No original module exists within this kernel
 - Installation
   - Installing to /lib/modules/5.4.0-126-generic/updates/dkms/

vboxsf.ko:
Running module version sanity check.
 - Original module
   - No original module exists within this kernel
 - Installation
   - Installing to /lib/modules/5.4.0-126-generic/updates/dkms/

vboxvideo.ko:
Running module version sanity check.
 - Original module
   - No original module exists within this kernel
 - Installation
   - Installing to /lib/modules/5.4.0-126-generic/updates/dkms/

depmod....

DKMS: install completed.
Processing triggers for systemd (245.4-4ubuntu3.18) ...
Processing triggers for man-db (2.9.1-1) ...
Processing triggers for libc-bin (2.31-0ubuntu9.9) ...
user1@user1-virtualbox:/media/user1/VBox_GAs_6.1.34
```

Required step to install the VirtualBox Guest Additions:

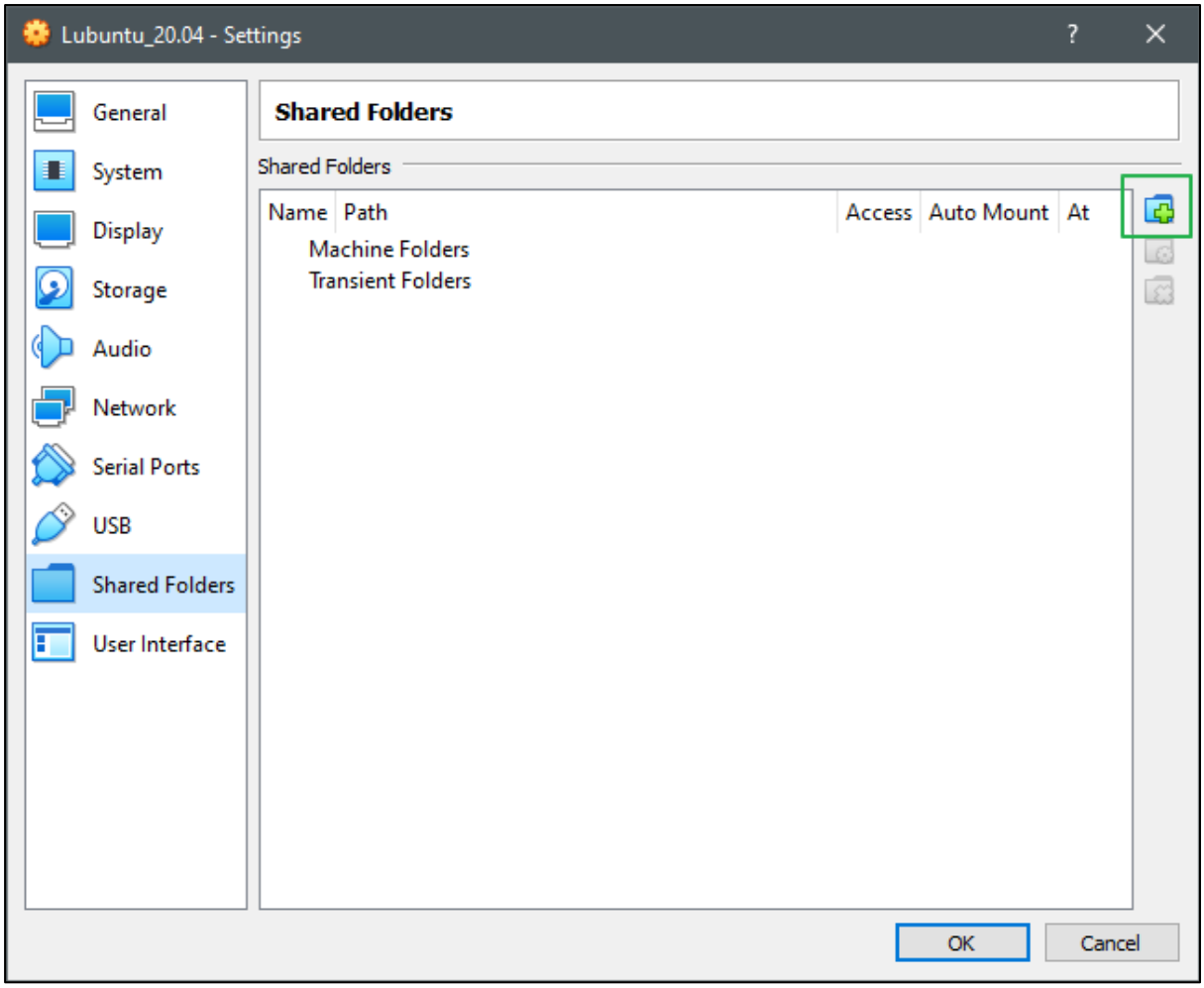
Type **`sudo ./VBoxLinuxAdditions.run`** into the command line then press [enter].



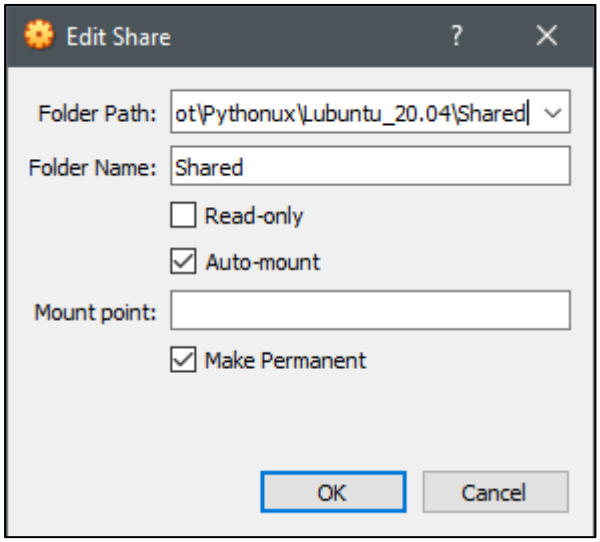
Create the shared folder in VirtualBox.

Open Shared Folder Settings dialog.

Select “Adds new shared folder.”

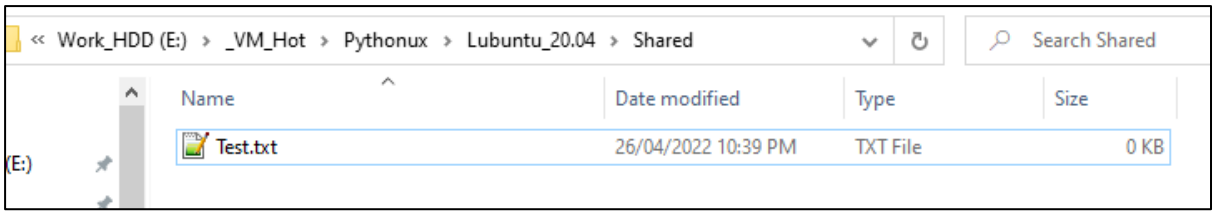


Select the shared folder location on your host machine. I usually create a folder named “Shared” in the directory with the virtual machine files.

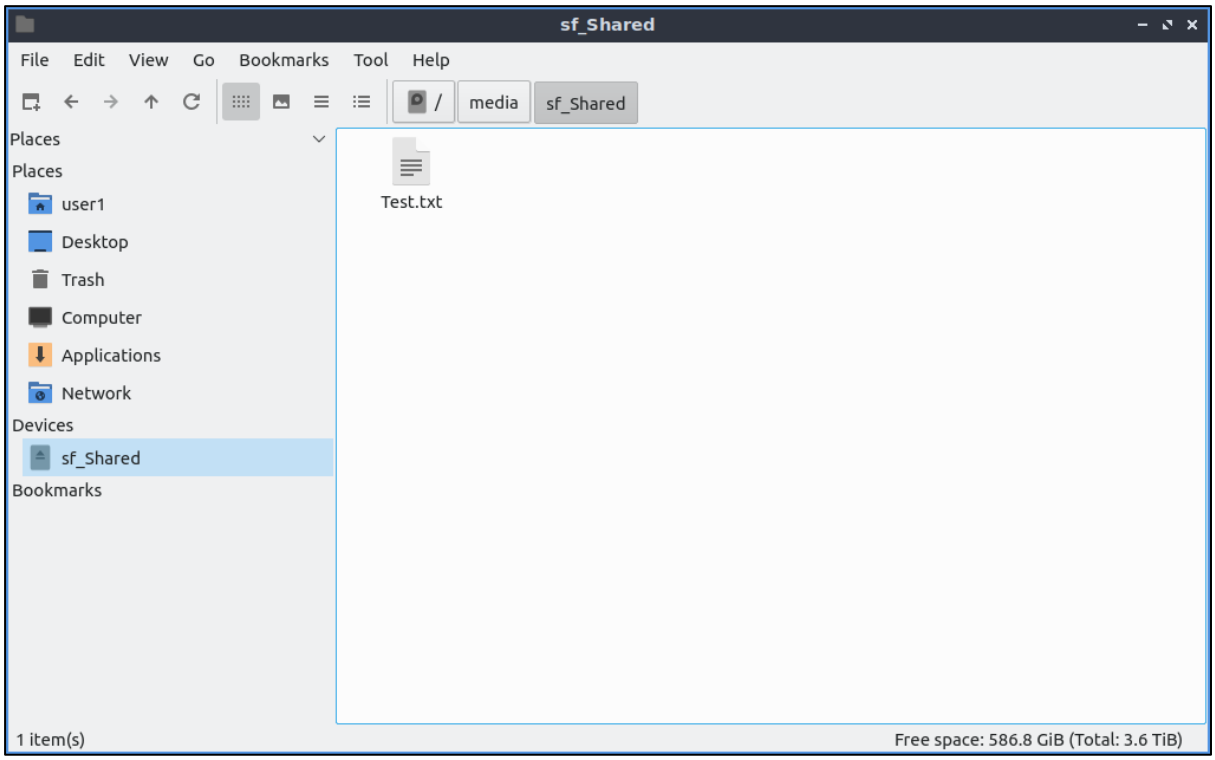


Restart the Virtual machine client.

After the restart you will be able to see and navigate the shared folder location.



View of the Windows Host – Shared folder



View of the Lubuntu Client – Shared folder access

If you find that you have issues with permission accessing the sherd folder:

[Check this for permissions issues]

Setting the shared folder permission and mount point for Ubuntu.

Open a terminal window.

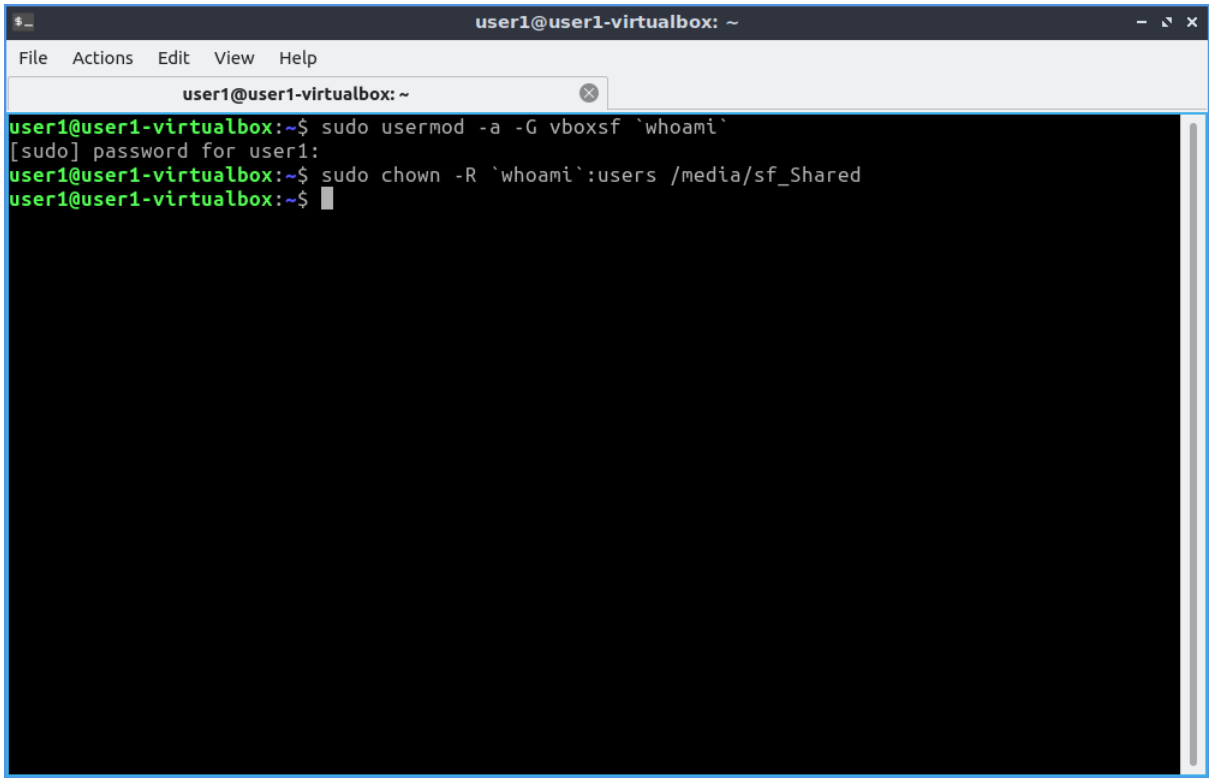
To enable the access to this folder for a regular user add your user to the vboxsf group.

```
sudo usermod -a -G vboxsf `whoami`
```

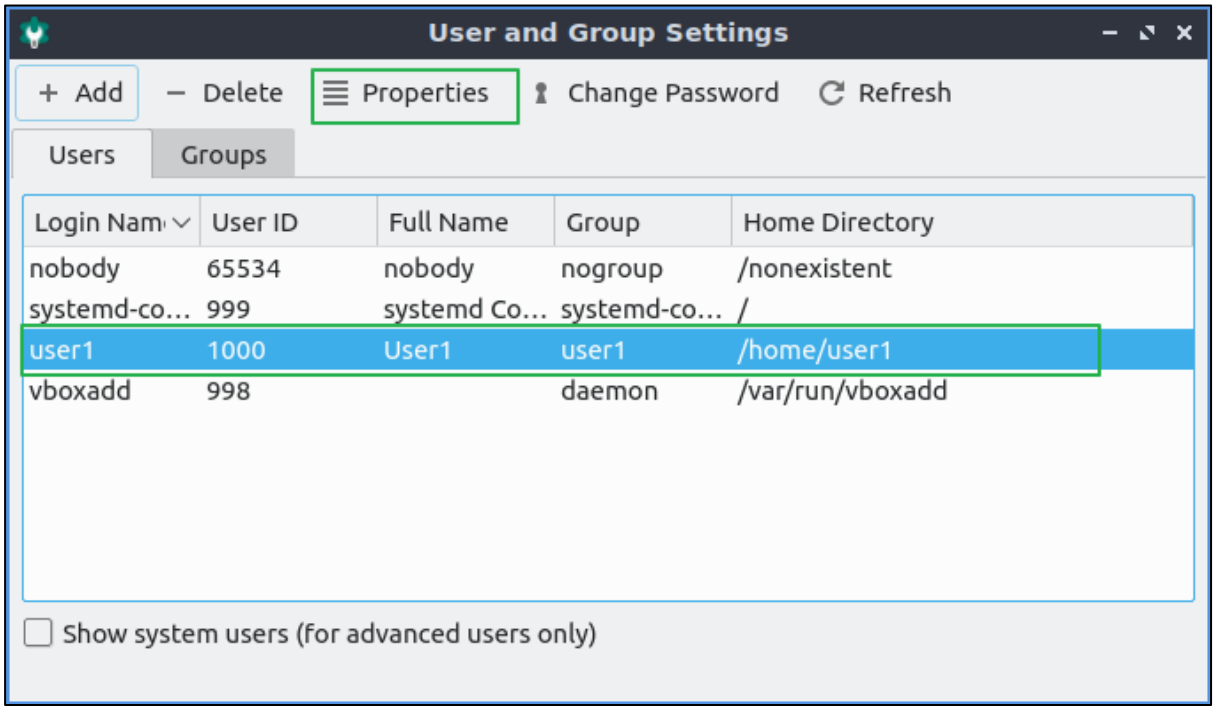
And change the permission of the media folder.

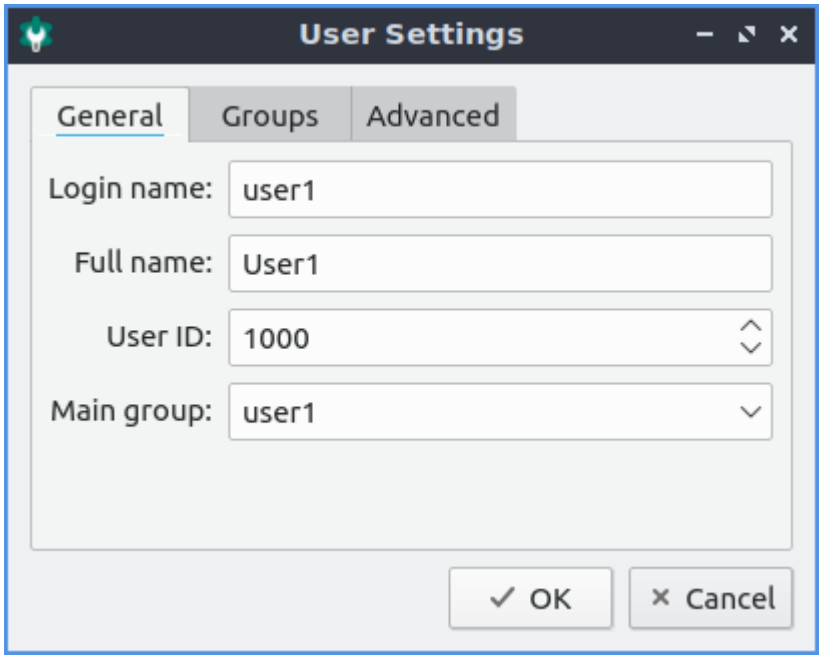
```
sudo chown -R `whoami`:users /media/sf_Shared
```

If you find that you are having issues with permissions (not being able to read write to some file locations or compiling code in the IDEs gives erroneous files search errors) you may need to check the account permissions. You can do this in the next section on “Users and Groups”.

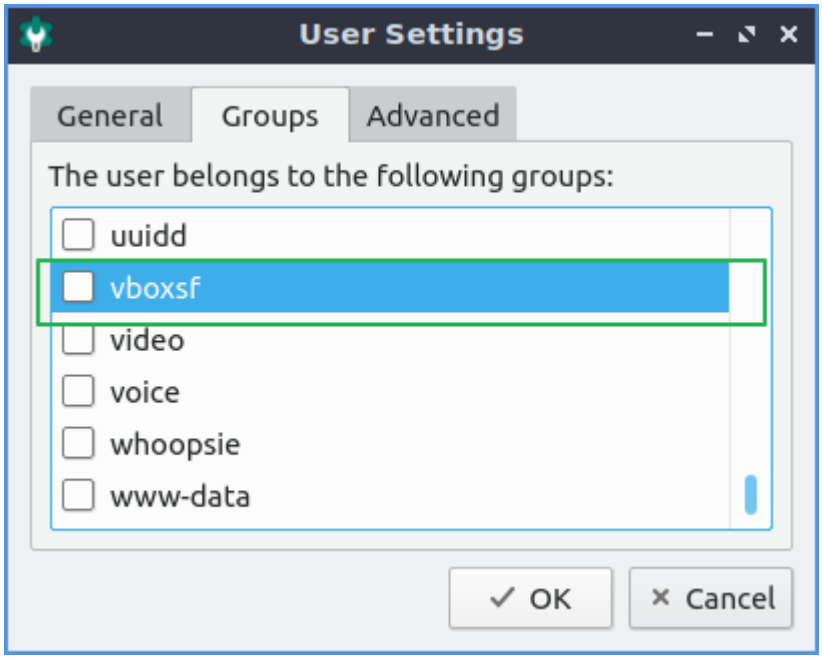


You can also do this from the Start menu “Users and Groups” GUI.





Select vboxsf

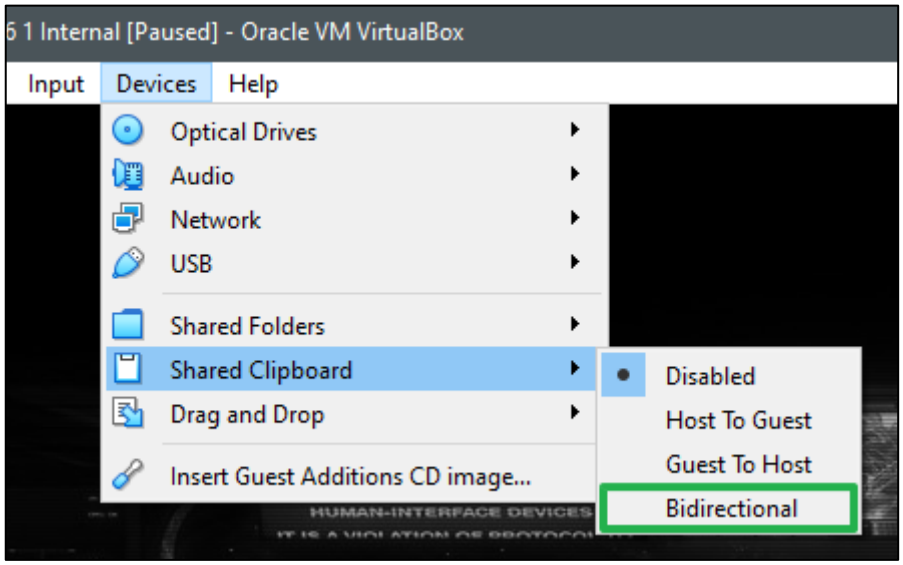


Then click [OK].

The default admin user account groups contain the following:
[■] adm [■] cdrom [■] dip [■] lpadmin [■] plugdev [■] sambashare [■] sudo [■] vboxsf

Setting “Shared Clipboard” in VirtualBox

You can copy files folders and selected text between the host and client as well if you find it convenient. I sometimes use this for code snippets. Change the setting to “Bidirectional”. You can change this setting at any time without requiring a restart.



You can also set the same For drag and Drop if you wish.

VT:100 Terminal emulators

Linux comes with a variety of options for different terminal emulators. Lubuntu ships with QTerminal and XTerm as the defaults. QTerminal may not always display TV:100 terminal emulation correctly in which case we may need to use XTerm or UXTerm. The default XTerm font size is quite small making the terminal difficult to see so you may need to adjust the default font size. It is also difficult to set programmatically at run time.

We can set defaults on our own machine using the ./home/user/.Xresources file...

Create the file if it does not exist and add the following.

.Xresources (/home/username/.Xresources)
xterm.termName: xterm-256color ! Reverse colors > on: white on black, off: black on white xterm*vt100.reverseVideo: off ! Ensure using UTF-8 ! xterm.vt100.locale: true ! Force UTF-8

```
xterm.vt100.locate: false
xterm.vt100.utf8: true

! Set default xterm window size:
xterm*vt100.geometry: 80x24

! Initial font size (ctrl+right click VT fonts menu > 0:default 1:unreadable 2:tiny 3:small 4:medium
5:large 6:huge 7:enormous):
xterm*vt100.initialFont: 6

! Enable/Disable TrueType fonts:
xterm*vt100.renderFont: false

! Default font and size:
xterm*vt100.faceName: DejaVu Sans Mono
xterm*vt100.faceSize: 12

! Chinese fonts:
xterm*vt100.faceNameDoublesize: AR PL UMing HK

! Enable Right-side scrollbar:
xterm*vt100.ScrollBar: true
xterm*vt100.rightScrollBar: true
xterm*vt100.leftScrollBar: false
xterm.vt100.scrollbar.width: 15

! Scrollbacks:
xterm*vt100.savelines: 16384

! Stop output to terminal from jumping down to bottom of scroll again:
xterm*vt100.scrollTtyOutput: false

! Cursor blink enable:
xterm*vt100.cursorBlink: true

! Enable clipboard:
xterm*vt100.selectToClipboard: true

! Double-click to select whole word and URLs:
xterm*vt100.charClass: 33:48,36-47:48,58-59:48,61:48,63-64:48,95:48,126:48

! Right Click for paste instead of middle button / Ctrl-C to copy, Ctrl-V to paste / Ctrl-M to maximize,
Ctrl-R to restore terminal window
xterm*vt100.translations: #override \n\
        ~Ctrl ~Meta <Btn3Down>: ignore() \n\
        Meta <Btn3Down>: clear-saved-lines() \n\
        ~Ctrl ~Meta <Btn3Up>: insert-selection(SELECT, CUT_BUFFER0) \n\
        ~Ctrl ~Meta <Btn2Down>: start-extend() \n\
        Ctrl Shift <Key>C: copy-selection(CLIPBOARD) \n\
        Ctrl Shift <Key>V: insert-selection(CLIPBOARD) \n\
        Ctrl <Key>M: maximize() \n\
```

```
            Ctrl <Key>R: restore() \n\

! Enable <Alt> key combination:
xterm*vt100.metaSendsEscape: true

! Make <Alt> work:
xterm*vt100.eightBitInput:   false
xterm*vt100.eightBitOutput:  true

! Fix backspace key
xterm.vt100.backarrowKey: false
xterm.ttyModes: erase ^?

! Every shell is a login shell by default (for inclusion of all necessary environment variables):
xterm*vt100.loginshell: true

! Specifies the number of pixels between the characters and the window border. The default is 2.
xterm*vt100.internalBorder: 2

! ###################################################################

! "xrdb -query > current_X_resources" --> view current X resources
! "xrdb -load ~/.Xresources" --> load and replace old entries
! "xrdb -merge ~/.Xresources" --> merges with any existing X resources
```

Restart the window manager for the changes to take effect.

Change default terminal:

sudo update-alternatives --config x-terminal-emulator

Select XTerm number and then [Enter]

I use Xterm or UXterm as it displays Curses windows correctly.

... or from within the XTerm window using [Ctrl] and right click.

You can also start XTerm from the command line with different font settings.

xterm -font 9x15 ./myapp

For Geany or other IDEs, go to Preferences.

Change Terminal: [x-terminal-emulator -e "/bin/sh %c"] (Geany default)

to

Terminal: [xterm -font 9x15 -e "/bin/sh %c"]

Other startup options...
xterm*font: *-fixed*-*-*-18-*

xterm –fa ‘monospace’ –fs 14

Checking dependencies and pointer width

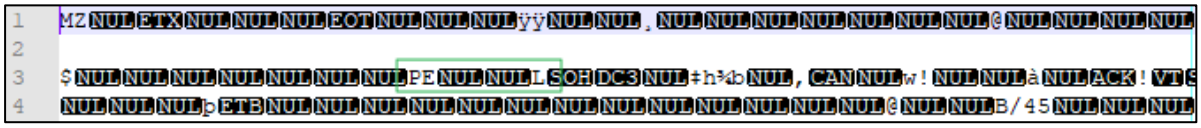
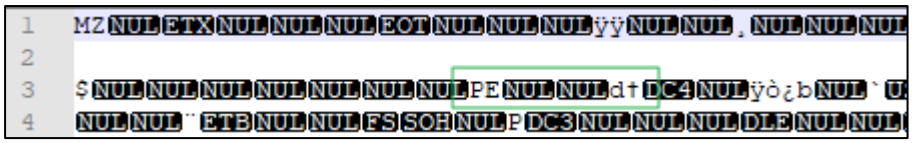
Sometimes we may need to check if our compiled output file is a 32-bit version or 64-bit version.

The windows PE file or Linux ELF file will have two possible flags set in the header of the executable PE..L denotes 32-bit and PE..d(t) denotes 64-bit, for Linux ELF file a 1 (32-bit) or 2 (64-bit) flag is set.

Windows 32-bit vs 64-bit PE files

Although GCC MinGW-w64 compiler chains default to 64-bit compiles, it is not guaranteed. To check, open the created exe or dll (aka pe file) with a HEX editor or Notepad++ and search for “PE”. It will be within the first 256 bytes. PE..L denotes 32-bit and PE.. d† denotes 64-bit as shown below.

‘.’ Denotes Zero or NUL (‘\0’ or Hx00)

NOTES: Checking for 32-bit or 64-bit exe/dll output.
Open the application or an EXE file with a text or hex editor (in my case notepad++) and locate “PE..” (Ideally found in first 256 chunk of data). In case of 32bit: PE..L.. (hex code: 504500004Cxxxx) = 32 bit      In case of 64bit: PE.. d†.. (hex code: 504500006486xx) = 64 bit   

We can also use the application Sysinternals Sigcheck.

https://learn.microsoft.com/en-us/sysinternals/downloads/sigcheck

sigcheck.exe path\PE-file

Example:

```
C:\Users\Axle\Downloads\Curses\Sigcheck>sigcheck C:\Dev-Cpp-
Embarcadero\TDM-GCC-64\bin\gdb64.exe
```

```
Sigcheck v2.90 - File version and signature viewer
Copyright (C) 2004-2022 Mark Russinovich
Sysinternals - www.sysinternals.com

c:\dev-cpp-embarcadero\tdm-gcc-64\bin\gdb64.exe:
        Verified:       Unsigned
        Link date:      10:00 AM 1/01/1970
        Publisher:      n/a
        Company:        n/a
        Description:    n/a
        Product:        n/a
        Prod version:   n/a
        File version:   n/a
        MachineType:    64-bit
```

Linux 32-bit vs 64-bit ELF files

Checking the header for the 1 or 2 flag can be done from the command line as follows using:

od -t x1 -t c executable | head -n 2 or *od -An -t x1 -j 4 -N 1 executable*

Example 1: od -t x1 -t c executable | head -n 2

```
$ od -t x1 -t c /usr/bin/codeblocks | head -n 2
0000000  7f  45  4c  46  02  01  01  00  00  00  00  00  00  00  00  00
        177   E   L   F 002 001 001  \0  \0  \0  \0  \0  \0  \0  \0  \0
$
```

The 5th byte of a Linux binary executable file (ELF format, see Wikipedia) is 1 for a 32 bit executable, 2 for a 64 bit executable.

Example 2: od -An -t x1 -j 4 -N 1 executable

```
$ od -An -t x1 -j 4 -N 1 /usr/bin/codeblocks
 02
$
```

02 = 64-bit

Using the “file” command: file ELF_File

Example:

```
$ file /usr/bin/codeblocks
/usr/bin/codeblocks: ELF 64-bit LSB shared object, x86-64, version 1
(SYSV), dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2,
BuildID[sha1]=c57f539540feda120c1b10a0ecf5566bb7165f82, for GNU/Linux
3.2.0, stripped
$
```

Checking library dependencies

Sometimes we may need to check what library dependencies our final executable is relying upon. We may do this to check if we have create an executable using “Static” or “Shared” shared libraries as intended, and also to check what libraries may need to be exported with our final “Release executable.

In **Linux** this is can easily be achieved with a tool that ships with the GCC compiler ldd.

To check dependencies for your executable type ldd [option]... path/file...

Example:

```
$ ldd -d /bin/gcc
        linux-vdso.so.1 (0x00007fff5070f000)
        libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f47eabf4000)
        /lib64/ld-linux-x86-64.so.2 (0x00007f47eae02000)
$
```

In the example we can see that the GNU C compiler is dependent upon:

- linux-vdso.so.1 – vDSO Used by the Linux kernel and C library to load user applications.
- libc.so.6 – "libc" the standard C library for linux.
- ld-linux-x86-64.so.2 – Used by Linux to link to shared libraries.

If the output shows no dependencies then it likely application built using the “-static” flag.

In **Windows** this requires some additional applications. Microsoft Sysinternals has a small command line tool named ListDLLs which can give a list of dependencies. Note that it will list all DLLs in the system used by the application and it is up to the developer to disseminate between common runtime libraries shipped with windows and additional runtimes required to be shipped, downloaded with the application.

Example 1:

```
Microsoft Windows [Version 10.0.19044.2130]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Axle\Downloads\Curses\ListDlls>Listdlls64.exe C:\Dev-Cpp-
Embarcadero\TDM-GCC-64\bin\gcc.exe

Listdlls v3.2 - Listdlls
Copyright (C) 1997-2016 Mark Russinovich
Sysinternals

No matching processes were found.
```

The GNU C compiler used in MinGW-64 is compiled with “-static” so any additional libraries are included in the executable. In other words the gcc.exe application has no external dependencies.

Example 2:

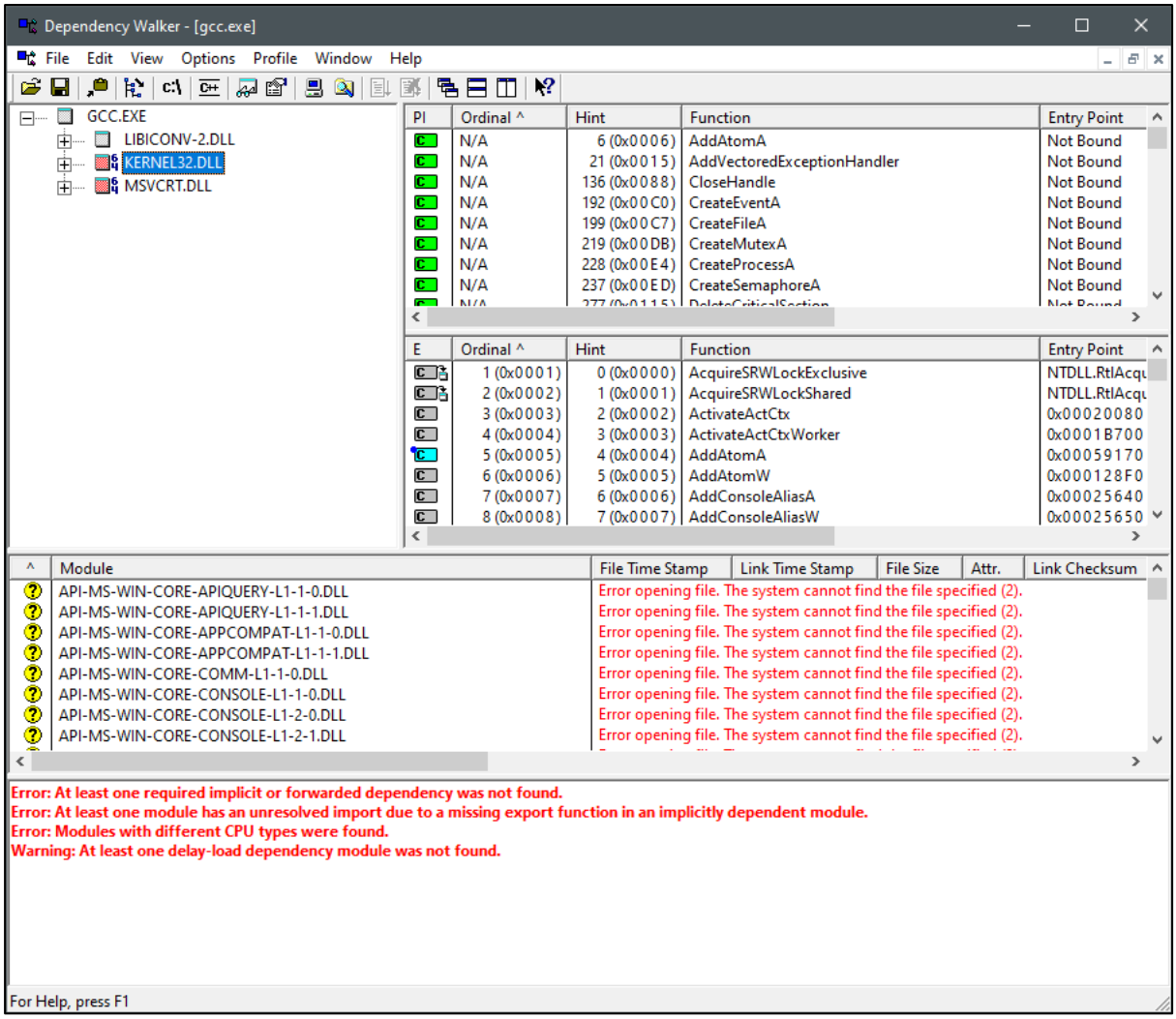
```
Microsoft Windows [Version 10.0.19044.2130]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Axle\Downloads\Curses\ListDlls>Listdlls64.exe cmd.exe

Listdlls v3.2 - Listdlls
Copyright (C) 1997-2016 Mark Russinovich
Sysinternals

------------------------------------------------------------------------------
cmd.exe pid: 5284
Command line: "C:\WINDOWS\system32\cmd.exe"

Base                Size      Path
0x00000000c12c0000  0x67000   C:\WINDOWS\system32\cmd.exe
0x00000000b0ad0000  0x1f8000  C:\WINDOWS\SYSTEM32\ntdll.dll
0x00000000af420000  0xbd000   C:\WINDOWS\System32\KERNEL32.DLL
0x00000000ae440000  0x2d2000  C:\WINDOWS\System32\KERNELBASE.dll
0x00000000af4e0000  0x9e000   C:\WINDOWS\System32\msvcrt.dll
0x00000000aece0000  0x354000  C:\WINDOWS\System32\combase.dll
0x00000000ae1e0000  0x100000  C:\WINDOWS\System32\ucrtbase.dll
0x00000000b0450000  0x125000  C:\WINDOWS\System32\RPCRT4.dll
0x000000009f940000  0x35000   C:\WINDOWS\SYSTEM32\winbrand.dll
0x00000000aeb90000  0x9c000   C:\WINDOWS\System32\sechost.dll
```

In this example CMD.exe relies on many of the system DLLs including the CRT msvcrt.dll.

You can download a copy of ListDLLs below:

https://learn.microsoft.com/en-us/sysinternals/downloads/listdlls

Another tool that is commonly used is “Dependency Walker”. Dependency Walker is far more advanced and comprehensive and will list the complete dependency tree for all libraries. Generally it is only the top level of the dependency tree that an application accesses directly.



You can download Dependency Walker from the link below. Be sure to get the correct download for your system x86 or x64:

https://www.dependencywalker.com/

Some of the Linux subsystems for windows such as WSL, Cygwin and MSYS2 may have ldd.exe available.

When checking for dependencies and to discover if our application is using “Static” or “Shared” linking we are normally checking for libraries that we know we have entered for the linker.

Debuggers and analysis

[Add information on debuggers, executable analysis, and code analysis]

[All Linux versions to be tested]

Process and Memory Monitors

Windows

Process explorer (Systinternals)

Process Monitor (Systinternals)

RAMMap64 (Systinternals)

HeapMemView (nirsoft)

Linux

htop

btop

Xfce Task Manager

LXTask

Linux process explorer?

GNOME System Monitor

Debuggers

Windows

gdb

SimpleProgramDebugger

x64dbg

edb

Immunity

Ghidra

WinDbg (Windows SDK)

OllyDbg

Linux

gdb

Ghidra

radare2

Cutter

REDasm

EDB (Evan's Debugger)

EXE-DLL Dependencies

Windows

DendancyWalker

ListDlls

Linux

ldd

readelf –d

lsof

lddtree

nm

https://www.baeldung.com/linux/shared-library-exported-functions

Hex Editors

Windows

HxD

Linux

Okteta

Bless

EXE-DLL Info

Windows

Sigcheck

Microsoft Application Inspector

Linux

EXE-DLL/SO Analysis

Windows

Spybot FileAlyzer

Resource Hacker

MiTeC EXE Explorer

CFF Explorer Suite

PE Tools

Pestudio

Linux

PE-bear

radare2

https://docs.remnux.org/discover-the-tools/examine+static+properties/pe+files

ELF/PaX Utilities

DLL_Exp_Imports

ADllExports v1.3 (Older application, but works exceptionally well)
https://www.kamburov.net/aleksandar/projects/index.html

DLL Export Viewer (Nirsoft NOTE: 32-bit and 64-bit versions)
https://www.nirsoft.net/utils/dll_export_viewer.html

DllExportFinder (Jacquelin POTIER)

StaticImportFinder (Jacquelin POTIER) Use on your exe to see what libraries it is using.

Code Analysis

SourceMonitor

Code Compare

Windows

WinMerge

Linux

Meld

KDiff3

Kompare

Unpackers

UniExtractRC3

Packers

UPX