Programming with Visual Studio

Python, C++, Fortran

WEB: HTML/CSS/JavaScript

Miguel Ángel Rapado Tamarit
Belén Moreno Santamaría
Juan Antonio Hernández Ramos

Department of Applied Mathematics
School of Aeronautical and Space Engineering
Technical University of Madrid (UPM)

Cover:
If you want something, go for it. Cover design Belén Moreno Santamaría

This manual is aimed at the Aerospace Engineering students of the Technical University of Madrid who may start programming different languages with Microsoft Visual Studio. It is also for anyone interested in start using this complete tool from scratch. Additional content can be downloaded from https://github.com/jahrWork. The manual has been written with the idea of increasing all the information presented and improving the contents. Ideas for something to be added, deleted or changed would be kindly appreciated. In addition, corrections and feedback about anything that could be improved or better explained are welcomed. All contributions can be made in:

Miguel Ángel Rapado Tamarit
marapadotamarit@gmail.com

2019 First Edition
2020 Second Edition Revised and Expanded

Miguel Ángel Rapado Tamarit
Belén Moreno Santamaría
Juan Antonio Hernández Ramos

ISBN 978-1727581539

Preface

Learning how to program does not only involve an understanding of the vocabulary and the grammatical rules of a specific programming language, but also needs of a huge amount of concepts around the language and the computer like programming theory, the purpose of the language, the programming paradigm, software design, etc. One question that appears when someone has decided which language to learn is; how can I ask my computer to execute the code I am going to write?. The answer needs from the concepts of Compiler, Interpreter and maybe from the decision of using a Text Editor or an Integrated Development Environment (IDE).

Whether we start programming in a self-taught way or guided by a teacher, sometimes it is taken for granted that the basic concepts are known or they are treated really quickly. In addition, the issues related to the programming environment are usually underestimated since they are considered negligible compared to the programming language topics. However, the amount of time saved when the development environment is controlled is huge, specially when the project is comprised by a large number of files. We must not forget that a program like a web application, a numerical simulation or an Arduino project are not a unique file. They are usually composed by files, folders, projects, solutions and generally a strong organisation of all the elements. This guide is intended to help users to faster develop programs by using Visual Studio, one of the most powerful IDE.

This guide is specially aimed at the Aerospace Engineering students of the Technical University of Madrid who may start programming with dif-

ferent programming languages. We want them to start using Visual Studio quickly and without any complication. Furthermore, they become familiarised with an environment that could be really useful for their future career in case they continue in the world of software writing.

For these purposes, the chapter 1 deals with the installation of Visual Studio in a Windows OS and some essential tools associated to the IDE. It also treats how to configure the environment in order to start using some really powerful functions.

Chapters 2, 3, 4 and 5 cover different programming languages; Python, C++ and Arduino, Fortran and JavaScript with HTML and CSS. In all of the chapters, it is explained how to install all the components involved. Later a project is created for each language and a really simple program called "Hello world" is executed in order to validate the installation. In addition, in each language: configuration aspects, tips, common questions, how to manage packages, modules, libraries, projects and solutions, etc. are treated. By doing so, the student can reduce the number of hours dedicated to the setting up of the computer while familiarising with the environment. In the case of Python or Arduino, the essential management of packages and libraries respectively are treated. The Fortran section deals with concepts related to the configuration of the project, the use of graphic libraries and a pile of interesting questions. In the case of the web programming, some fundamental concepts of HTML/CSS and JavaScript are explained. All the guide is complemented with tips result of the dedication for a long time to coding in the Department of Applied Mathematics to Aerospace Engineering.

Finally, the chapter 6 covers the use of Git and GitHub, a Version Control Systems (VCS) compatible with Visual Studio, and once again, a powerful tool when managing codes.

The experience of the students of Numerical Computation and Computer Science of the Aerospace Engineering degree of the Technical University of Madrid have made possible this guide since their comments and doubts have accurately contributed to the development of the content.

Miguel Ángel Rapado

July 2019

Contents

Chapter 1

Introduction

1.1 Development Environments

The aim of this manual is to guide the user in the first steps of Python, C++, Fortran and Web (HTML/CSS/JavaScript) programming with Visual Studio. When starting to code in any language there are a number of essential concepts that should be perfectly understood and they are not always clear for beginners. We cover some of those concepts in this introduction.

Three important concepts we must not confuse are **Compiler**, **Interpreter** and **Integrated Development Environment (IDE)**. On the one hand we have Compilers and Interpreters, they are in charge of transforming the code written in a programming language (Fortran for example) into another programming language (the target language). They generally translate the source code from a high-level language (HLL, languages closer to human language and independent of the particular computer where is written) into a low-level language (e.g. assembly language, object code or machine code, those closer to the hardware) in order to create an executable program. While the compiler takes the whole program and translates it at a time and generating an intermediate object code (less memory efficiency),

the interpreter takes all the code line by line and performs the compilation and execution simultaneously. On the contrary, when a compiler is used, the execution comes after the compilation. Compilers are generally faster than interpreters but the detection and removal of errors are more difficult since the interpreter allows resuming the translation when the error is fixed. In addition, it is said that the interpretation or compilation of the code are a property of the implementation and not the language. Hence, some programming languages like C, C++ or Fortran are implemented with a compiler while Python make use of an interpreter in one stage of the implementation.

On the other hand we find IDEs (like Visual Studio). They are a software application that offers some facilities for software development. Some of the functionalities that the IDE provides are a source code editor (similar to a text editor), build automation tools, version management tools, a debugger, intelligent code completion and a long etcetera. It is possible to write programs without an IDE but it is less comfortable specially when too many source codes are involved in the program. This guide explains how to install and start using **Microsoft Visual Studio** (which accepts 36 different programming languages) and all the necessary tools associated to the mentioned programming languages. Some of these tools are for example the **Intel® Fortran Compiler** (Parallel Studio XE 2018) or the compiler for C++ (**Intel® C++ Compiler** or similar) which allow compiling programs. In addition, a Python interpreter, the Visual Micro extension (Arduino projects) and some web tools will be included in the IDE. Please notice that the first thing to install in your computer is the IDE, Visual Studio. Then all the compilers, interpreters and extensions needed are added to the environment so it can recognise the different languages.

It has been mentioned before that working with an IDE is not mandatory for programming, it is just an useful tool that can make easier the software development. The process of coding can be developed by means of a code editor or a simple text editor, later with the use of the command prompt we can call the compiler/interpreter of our programming language and build any project. Hence, one question that may appear is: What are the differences and advantages of using an IDE instead of a Code Editor?

Apart from the list mentioned, the **Integrated Development Environment** provides more tools like project manager, deep code analysis to feed auto-completion, support for many programming languages (in the case

of Visual Studio), etc. The IDE is where we write the code and convert it into a product, a compiled program, a web app, etc., using a lot of tools and everything in one single program.

On the contrary, we find **Code Editors** which are more than text editors since they have enhanced features. When they are optimized for source code editing, they provide syntax highlighting, navigation tools, search possibilities and they are often easily customized. The bottom line is that they are limited to the edition of plain text, they do not compile and debug the code in their nature, however, they accept plug-ins and extensions to imitate the use of an IDE. Finally, a third option would be programming with **command-line tools**.

In order to break into the world of software development, we consider that an IDE is the proper tool to use. With Microsoft Visual Studio we can program more efficiently thanks to all the tools that it incorporates by default, with no need of adding plug-ins. A powerful file management tool, some really interesting tools like “Find All References”, “Go To definition” or “Intrinsic Parameter Info” are some examples of the available utilities that Visual Studio incorporates. We can compile all the languages, manage source codes and navigate through them, manage all the files involved in the project and debug the code with no need of external programs.

In the IDE, we have access to all the essential tools related to software development just learning how to use one program and we do not need to check the compatibility between the tools used since they are all implemented in the same environment. While the potential of Visual Studio is more remarkable when developing a big software project, it is a really comfortable tool to use even with little projects. It is sometimes said that a Code Editor is going to be very useful with languages that does not need to be compiled (like JavaScript or a static HTML page for example) since they are lighter. On the contrary, languages like Fortran that are being compiled and executed constantly are going to be easily used with an IDE, which offers quickly (and with no need of external plug-ins) these capabilities. This is specially important when our project needs from the pre-process, compilation and linked of a big amount of codes (with an Editor we should do it manually).

In addition, IDE’s often use the **Make** tool, this is a build automation tool in charge of building the libraries and executable files of our program

starting with the source code. It makes use of files called **Makefiles** where a set of directives are stored specifying how to generate the target program. This tool decides automatically the order of compilation for all the source codes involved in the project by reading the dependencies between them. Furthermore, if we change the source code of a file and rebuild the whole project, the Make tool will update the project by compiling and rebuilding only one file, with no need of building the whole project again. This can be done by reading the modification date of all the files and as a result the process is much faster, specially when the project has a large number of source codes.

In conclusion, considering the tools offered by Microsoft: Visual Code and Visual Studio, the first one does not offer the same potential (it is a Code Editor) as Visual Studio. That is why we use Visual Studio in this guide.

1.2 Configuration and Project Files/Folders

A program written in a human-readable programming language with our IDE is not understood by the computer. The Central Processing Unit (CPU) of the computer, in charge of the execution of the program, only processes machine code instructions which reduces the program to a number of very specific tasks in the registers of the CPU itself. The concepts of **Source** and **Object Codes** are essential. Both types of codes collect the computer instructions that conform our program. While the source code is written in a human-readable programming language, the object code is generated by the compiler and stores the program in a computer language like machine code (binary for example) or an intermediate language.

The different programming languages that exist store the mentioned codes in different types of files that can be distinguished by the file extension. Just as an example we will see that a source code written in Fortran language is stored in a `.f90` file while the object code generated by the compiler receives the extension of `.obj`. In the case of a C++ program, the source code can use the extension `.cpp` among others and the object code will be stored in the same extension as Fortran language (`.obj`). Apart from these files, a list of common used extensions can be found in a specific programming language (libraries, modules, images, scripts, etc.), this

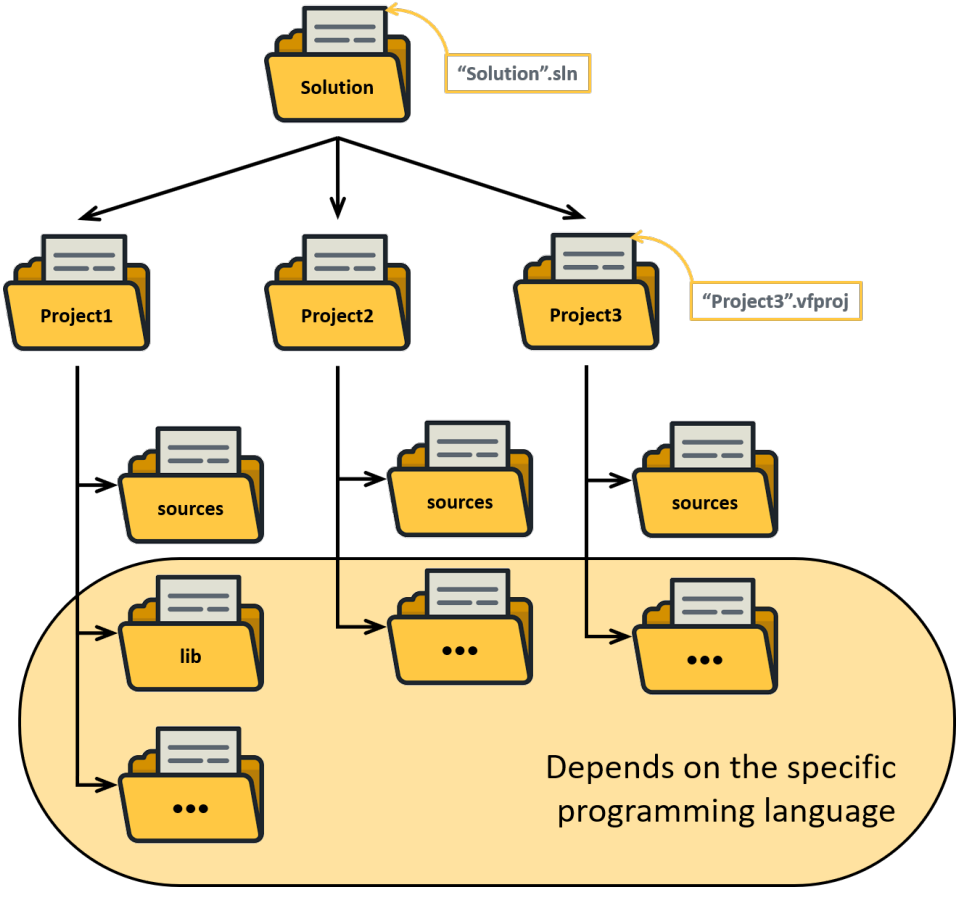

Figure 1.1: Scheme of the main folders that we should have in a solution containing different projects.

information is also treated during this guide.

An important concept to handle is the difference between the **logical files/folders order** that is generated in our projects and the **real order** of the files and folders stored in the hard drive. In the first case, we add all the source codes, images, libraries, etc. to the project and we organize all the elements in the Solution explorer of the tool. We can create folders or move files as we want. However, each file can be stored anywhere in our hard drive, maybe in the folder of a different project/solution, in the desktop, in the cloud, etc. This is specially important for languages like Fortran where there is not relation between both schemes. We can add a

library file to the logical folder called “libraries” while the real file is stored in our Desktop. The Fortran project will store the information about the location of the library. In addition, if we create a new file in that folder (using Visual Studio), it is by default stored in the project folder in the hard drive, but not necessarily in a folder called with the same name. The folders that organize all the necessary content for the program are different and we must manage not only the order used in the solution (seen in the solution explorer in Visual Studio) but also the real location of the files.

It is not the same for languages like Python or JavaScript. Once the project is started, if we create a folder or a code file using Visual Studio, the same folder or file is created in our hard drive in the same location where our project and solution are (both projects and solutions are explained later). In this case we do not have to take an strict control of files since the structure is repeated and consistent.

Let’s think about a code file that we use in many different projects (stored in our desktop, outside our project folder, for example). If we are using Fortran, we can add that file to our project with no need of copying the file to the project folder. Different projects will have the path associated to the file and if we make a modification to the file, all the projects benefit from that update in the next compilation. Once again, this is different for languages like JavaScript. If we add the file to the project, a copy is automatically made in the project folder and a modification in the code stored in the desktop will be totally useless. If the file is used in 3 different projects, a modification in the code can become a mess. We strongly recommend to delete all files that has been previously copied to a project in order to avoid unnecessary duplications (there cannot be two files with the same name). If the code is going to be used in different projects we will build a library with that code (treated in each chapter) and include it in the different works.

Important Notice

For all these reasons, we are going to follow the same files/folders scheme in our projects from now on. Whether it is automatically created or not, we have to store all the source codes in a folder called "sources" inside our folder project (see Figure 1.1). If the folder is not created by default in both the real and the logical folder scheme, we have to do it manually. In addition, a list of files and folders are created together with the source files in order to make the project work properly. These additional elements depend on the programming language we are using and they would be treated in the specific chapter. As a practical implementation of this topic, take a look at the example 1.4.

Nevertheless, our programs are not only comprised by code files. Visual Studio works with the concepts of projects and solutions, each of them with their **configuration files and extensions associated**. It is important to clarify the nomenclature that the program gives to the project and solution files since we are going to live together with them from now on. Simplifying, it could be said that a *project* is a specific program written in a programming language while a *solution* is a set of different programs. In the official Microsoft Visual Studio documentation a project is defined as:

> "...In a logical sense, a project contains of all the source code files, icons, images, data files and anything else that will be compiled into an executable program or web site, or else is needed in order to perform the compilation. A project also contains all the compiler settings and other configuration files that might be needed by various services or components that your program will communicate with."
>
> (https://msdn.microsoft.com/en-us/library/b142f8e7.aspx)

Related to solutions, the Visual documentation says:

> "A project is contained, in a logical sense and in the file system, within a solution, which may contain one or more projects, along with build information, Visual Studio window settings, and any miscellaneous files that aren't associated with any project. In a literal sense, the solution is a text file with its own unique format; it is generally not intended to be edited by hand.
> A solution has an associated *.suo file that stores settings, preferences and configuration information for each user that has worked on the project."
>
> (https://msdn.microsoft.com/en-us/library/b142f8e7.aspx)

Both the solution and the project have files associated. The file where the solution is stored has `.sln` extension and contains information that the environment uses to load it, load the projects within the solution and other information attached. As an example, we can find there the file format version, the newest version of Visual Studio that saved the file, the older version of VS that can open the solution and a list of parameters needed for loading the project files. In the case of the project, the extension of the file depends on the programming language used. If an Intel Fortran project has been created, the extension is `.vfproj` while in the case of a C++ compiler, `.icproj` would be the extension. This file stores information about the configuration of the project in the different building modes; Release, Debug and those created by us. It also stores the name and relative path of all the files needed for opening and building the project and the admitted extensions that can be opened.

The solution and project files are XML files (Extensible Markup Language), this means that the data structure is machine-readable and human-readable at the same time. Although both kind of files are automatically generated by Visual Studio and they are not made for being modified by hand, we can do it if required. We could open the file with a text editor, read the information and change parameters directly with no need of using VS. The solution settings for each user are stored in the `.suo` file that Visual Studio automatically creates when the solutions is saved. Unlike the men-

tioned files, this one is stored in a binary format so it is not human-readable and can not be modified by hand.

All the configuration files mentioned are physically stored in our hard drive, in the folder reserved for the solution and the projects.

Before continuing with some examples, a note on nomenclature: firstly, during the whole text **italics** are used to indicate a sequence of steps to follow in order to access some options, configurations, etc. in our computer. For example: click on *File/New/Project...* in your IDE. Secondly, **typewriter font** is used for files and folders names, commands to type in console, pieces of code, etc. For example: the file has `.f90` extension. Finally, if some name should be written in a specific field, that name is written in **quotes**, for example: write "Hello World" as project name.

1.3 Example 1

In order to help the reader in the first steps with Visual Studio, a repository has been created in GitHub with the solutions and projects explained here. Apart from the examples created in the guide, some extra content can be found, specially in the fields of Applied Maths, Simulation software with Fortran, Energy efficiency in buildings or Embedded software (C++). The repository is located in the url: https://github.com/jahrWork (Figure 1.2).

Let's download, open and execute an example of this repository in order to get familiar with the solutions and projects. Follow these steps:

1. Open the url of the repository in your browser.
2. Click on *Visual-Studio-projects*.
3. Click on the green button (*Clone or download*) and click on *Download ZIP*.
4. In the folder where the ZIP file has been downloaded, unzip the folder and open it.
5. Double click on `Projects.sln` in order to open the solution in Visual Studio. If you just want to open the projects of a specific language, open the folder you want and double click on the `.sln` file.
6. You find in the Solution Explorer a mix of projects in different pro-

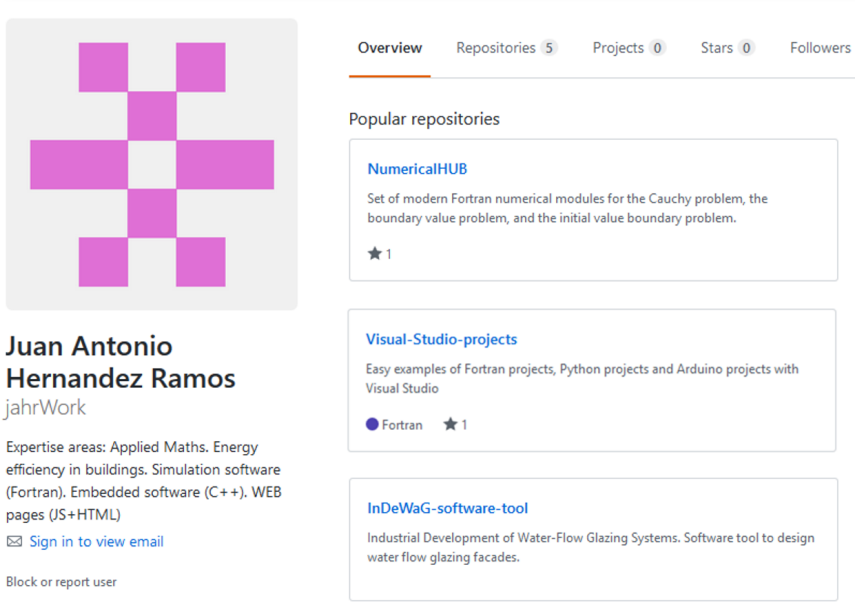

Figure 1.2: jahrWork repository in GitHub created by Juan Antonio Hernandez Ramos.

gramming languages, make right click on *MainFortranGraph* and click on *Set as StartUp Project* in order to select this project as the execution by default when clicking on the start button.

7. Click on *BUILD/Build MainFortranGraph* and wait until VS finishes.
8. Click on *DEBUG/Start Without Debugging* or press `Ctrl+F5`. The result of the execution is the plot of a sine curve.

1.4 Example 2

It is assumed now that a working version of Visual Studio is already installed in your computer (see chapter 2). In this section we create a unique solution called "Semester1" and as many projects as milestones have to be completed during the semester. We use Fortran language so the proper compiler must be already installed (see chapter 5.1). The files and folders structure mentioned above is used for this example.

Follow these steps:

1. Open your installed version of Visual Studio.
2. Click on *File/New/Project...* (Figure 5.6).
3. In the *Intel(R) Visual Fortran* menu select *Console Application* and then click on *Main Program Code*.
4. Write the name of the first project to create in the field *Name*, e.g. "Milestone1". Notice that the solution gets the same name by default.
5. Change the *Location* of the solution, a directory will be created there.
6. Write now the name of the solution "Semester1", it does not have to be the same as the name of the project (Figure 1.3).
7. Select the option *Create directory for solution*.
8. Click on *OK*.

Open now the path where your new solution has been created and open your solution folder. You find a `.sln` file with the name "Semester1" and a folder `Milestone1` containing the project data. Right now in this project folder you have a file `Milestone1.vfproj` with the configuration of the project, this configuration data is read by VS when the solution is opened. In addition, the first source code file `.f90` (with the same name of the project) has been created.

The folders structure is really different in the Solution Explorer of Visual. Three folders appear inside the project domain: *Header Files*, *Resource Files* and *Source Files*. The last one contains the file `Milestone1.f90` mentioned above, with the main program code in Fortran (the Figure 1.4 shows the initial structure of folders and files). In this code the "Hello World" example is already written so we can build the project and execute it without debugging. The results will be printed in a command window.

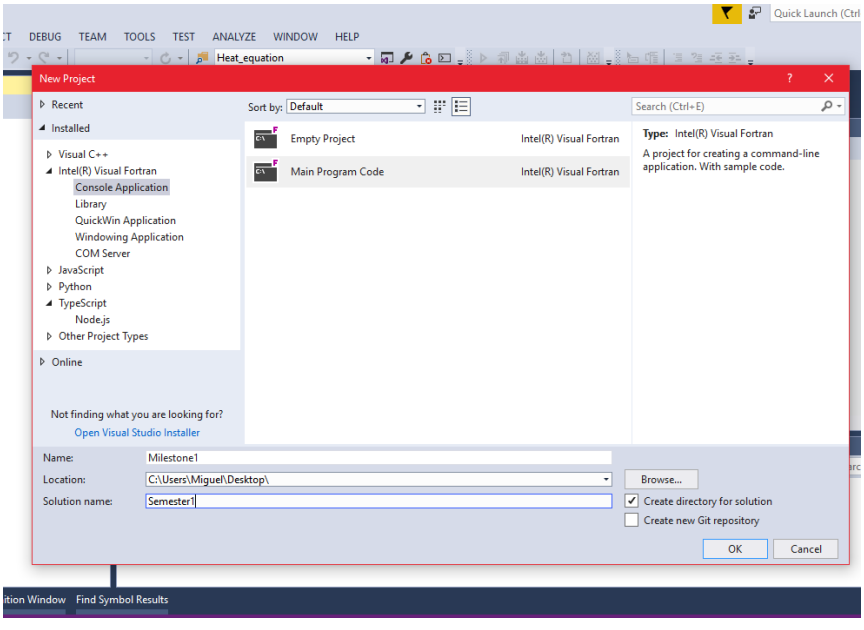

Figure 1.3: Creation of a Fortran project within a solution in Visual Studio.

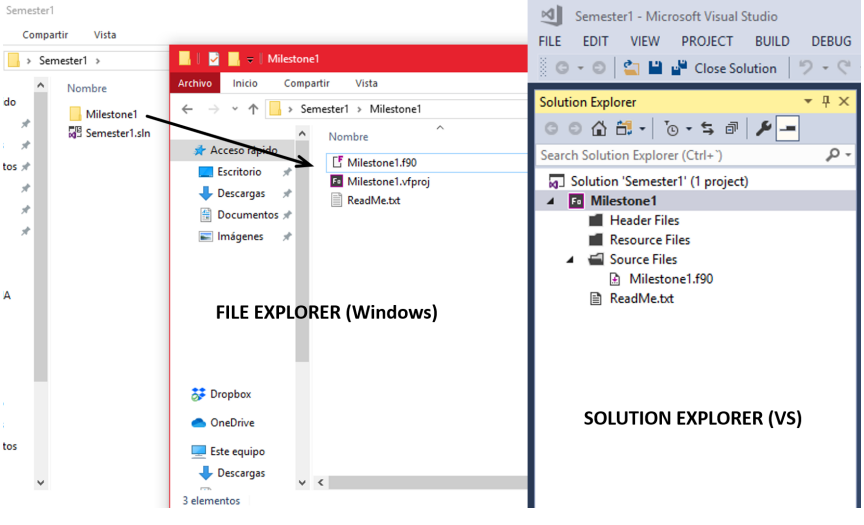

Figure 1.4: Initial folders and files structure in the File Explorer of Windows and the Solution Explorer of Visual Studio.

However, our first project is going to grow and some files and folders will be created. The following steps show how to imitate the structure seen in the Figure 1.1 in order to maintain a strict folder order.

1. Open in the File Explorer of Windows OS the project folder: `...\Semester1\Milestone1`.
2. Create there a folder called "sources". This folder will contain all the source codes of your program.
3. Still using the File Explorer of Windows OS, cut the file `Milestone1.f90` and paste it inside the recently created folder. The Figure 1.5 shows the final structure of folders and files.
4. If you try to open this file in VS (double click on the name of the file in the Solution Explorer of the IDE), you will receive an error since the file is not in the initial location anymore.
5. Remove this file in the Solution Explorer by making right click on the name of the file and pressing *Remove* and then clicking on *Remove* once again.
6. Make right click on the name of the folder `Source Files` in the Solution Explorer and click on *Add/Existing item...*. Open "sources" and select `Milestone1.f90`. The file is once again in the project but with the new location.
7. From now on, all your codes can be stored in the same file "sources" (referring to the real order of the files) and they all can be stored in "Source Files" (logical order seen by the Solution Explorer).

Finally, all the projects to be created during the rest of the semester (Milestone2, Milestone3, etc.) can be stored in the same solution. The section 5.4 tells how to include new projects and files in a solution already created. At the end of the semester you will have something like the Figure 1.6.

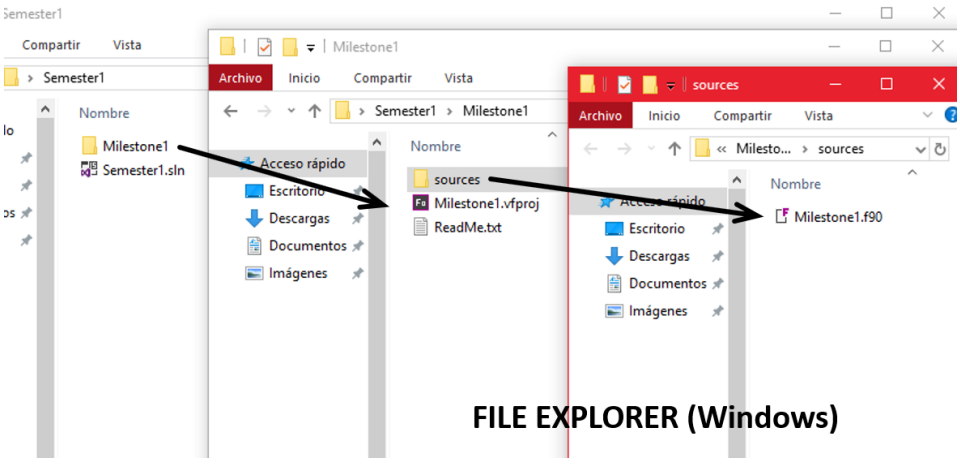

Figure 1.5: Folders and files structure in the File Explorer of Windows after organizing the Fortran project.

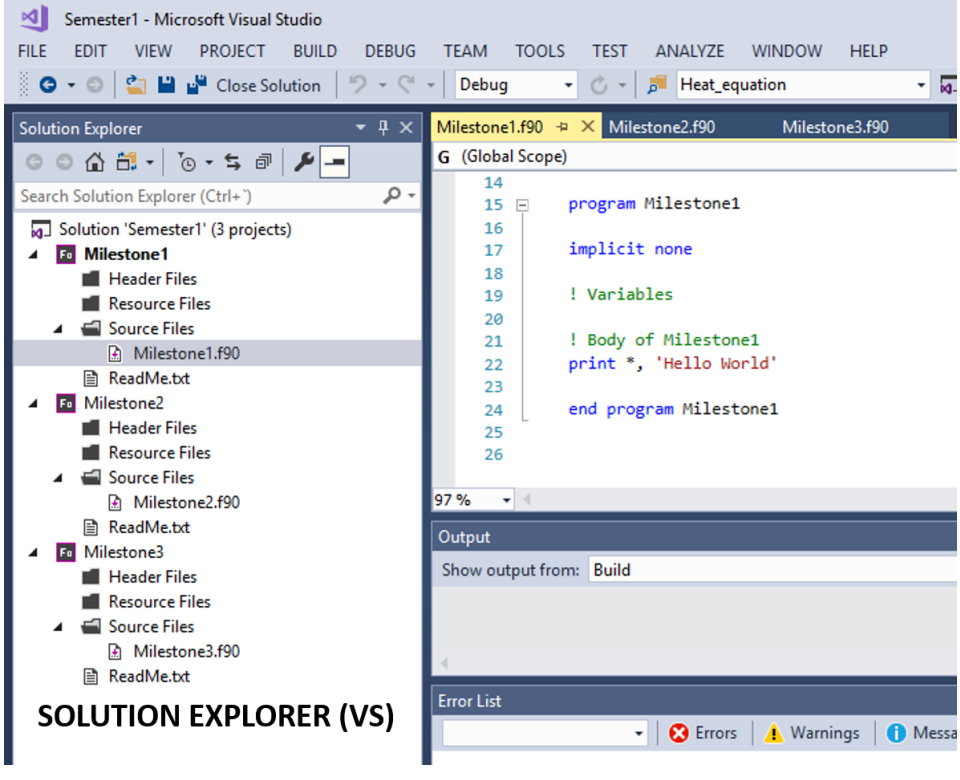

Figure 1.6: Folders and files structure in the Solution Explorer of Visual Studio after creating three projects in the same solution.

Chapter 2

Visual Studio

2.1 Installing Visual Studio

The first thing we are going to do is to install the latest version of Visual Studio, the IDE, in our computer. We have to **download the installer** of the program in the tab *Downloads* of the official website (https://visualstudio.microsoft.com/es/downloads/). There, Microsoft offers the installers for the Community, Professional and Enterprise versions[1]. Please notice that Professional and Enterprise releases have to be paid in case no license is obtained. However, the Community version is free and it is enough in most of the cases for the use we are going to make. If you are interested in the **VS Enterprise distribution** see how to install it with an academic license in the section 2.5.

[1] In the same webpage Microsoft give us the possibility of downloading another tool: Visual Studio Code. In the introduction of this guide we explain why we use Visual Studio instead of this editor.

Important Notice

- Please notice that Visual Studio **MUST be installed in the first place**. The compilers, interpreters and extensions needed will be included later and with them all the tools that ensures the compatibility with all releases of Visual Studio installed in the computer.
- The examples of this book are created using Visual Studio 2017 and 2019. Typically, the Microsoft webpage offers the latest version of the IDE, however, you will find a previous distribution to download in case your computer is running an older Windows version (like Windows 8 for example).

When opening the url we quickly find the section reserved for the last release of Visual Studio (see Figure 2.1). In case the Visual Studio downloads are not shown, we can look for it by scrolling down the webpage or writing "Microsoft Visual Studio" in the space reserved for the search of downloads. We can obtain the installer of the Community version by clicking on *Free Download* in that distribution. A file is downloaded on our computer, the name should be something similar to `vs_yyyyyyyy_xxxxx.exe` where the distribution appears instead of "y" and a number of version appears in the place of the "x". We have chosen the Community version so we obtain a file similar to `vs_community__2074385403.1558468341.exe`. In case a more complete IDE version is needed, later it is explained how to obtain a student license and download the Enterprise distribution of the tool.

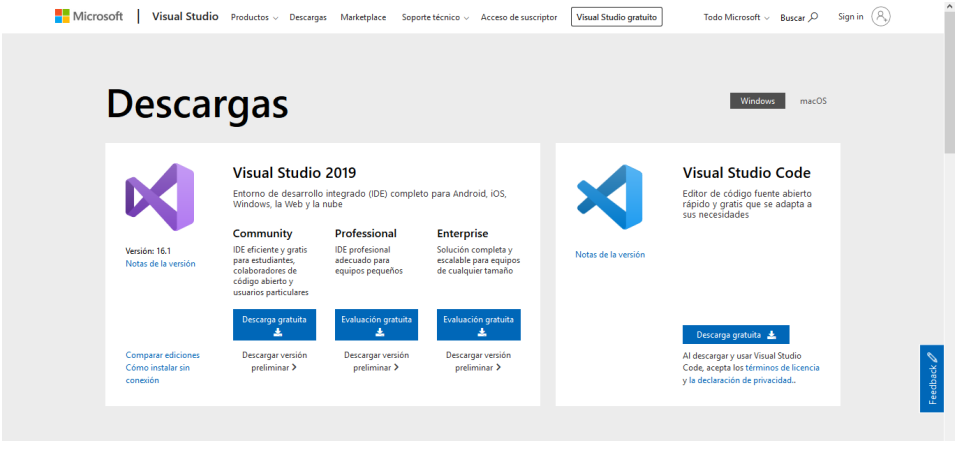

Figure 2.1: Downloads tab of Microsoft Visual Studio webpage where the different installers for the distributions available can be downloaded.

Installing Visual Studio Community version:

1. Execute the installer you have obtained. Accept first that the execution is done as an administrator and later accept the privacy conditions and the terms of the license of the software by clicking on *Continue*. Wait for the program to download and start installing (see Figures 2.2 and 2.3).

2. When finished, some options will appear in a window. There are four different tabs: Workloads, Individual Components, Language Packages and Installation folder. This tool called Visual Installer is essential for obtaining compilers and other tools used with different programming languages. In Workloads we choose *Desarrollo para el escritorio con C++* and in Language Packages we choose: English or both Spanish and English (see figure 2.4).

3. Click on *Install* and wait (it might be slow).

4. Once finished close all windows and restart the computer.

5. Execute Visual Studio. The first time some welcome windows appear (see figure 2.5), first we are going to Start Session. Although Community distribution is free for the people we first obtain a license for 30 days and we have to initiate session in order to activate the product. Click on Initiate Session and in the new window that appears use your

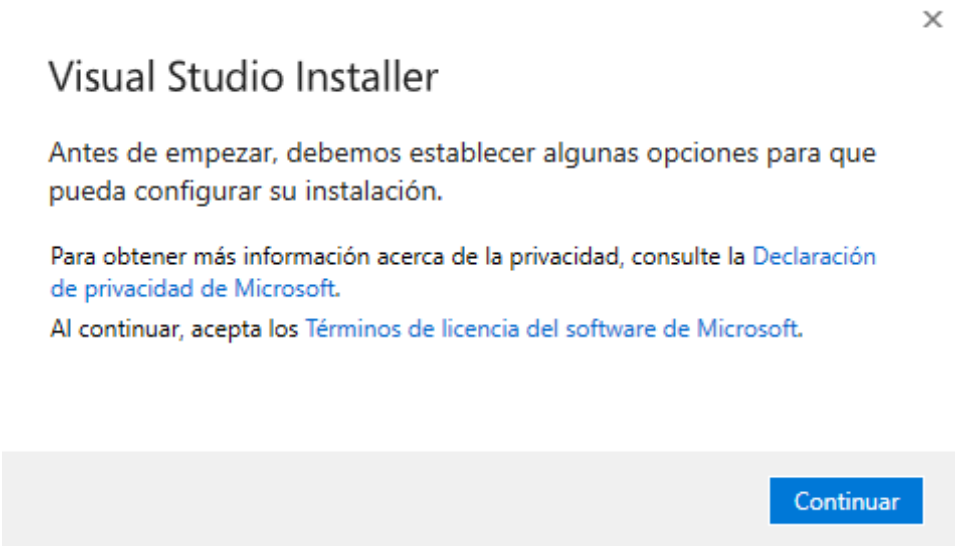

Figure 2.2: First window that appears when executing the Visual Studio installer.

Microsoft account or your institutional email[2], in case you do not have a Microsoft account create it using the available link: *Create one!*.

6. Now the IDE starts to configure everything so it can take awhile. An Start window appears but you can click on *Continue without code* and you access directly to the environment.

7. Now you have to check if the product is already activated with a license. Notice that the initial of your name appears in the top right part of the IDE so you have already started session. If you click on the letter of your name and *Account Configuration* (see figure 2.6) you will see that your credentials are activated. In the right part of that window it may say that you are using a 30 days license, click on *Search for an updated license* and it automatically activates the product (see figure 2.7).

8. Your IDE is ready to work now.

[2]If you use the institutional email of the UPM (finished with *@alumnos.upm.es*), you will be redirected to the organization's sign-in page of the UPM. There you have to write again the same email and the password for that account and confirm the process. A webpage with the header "Servicio de Identidad de RedIRIS" may appear.

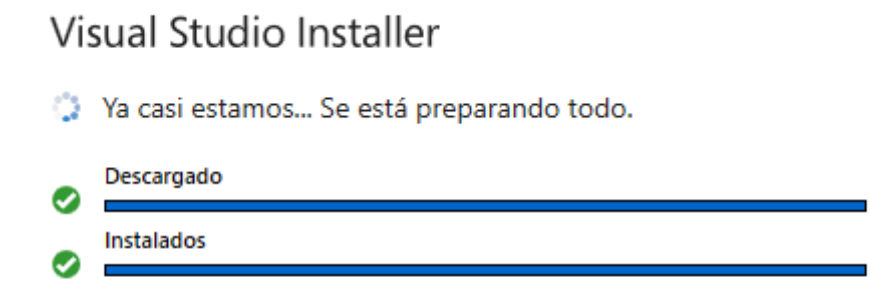

Figure 2.3: Download and installation process of the Visual Studio environment.

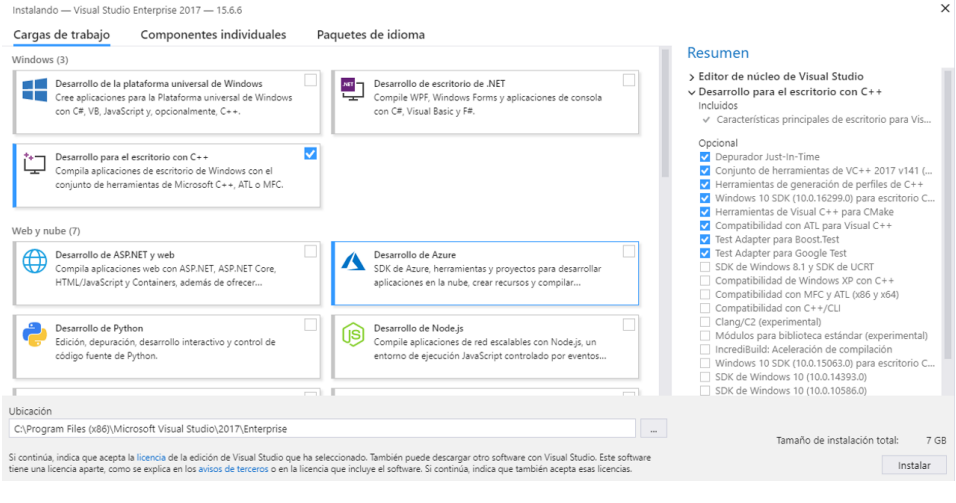

Figure 2.4: Workload to be chosen in the installation process, *desarrollo para el escritorio con C++* is selected in this step. In language package we choose at least English.

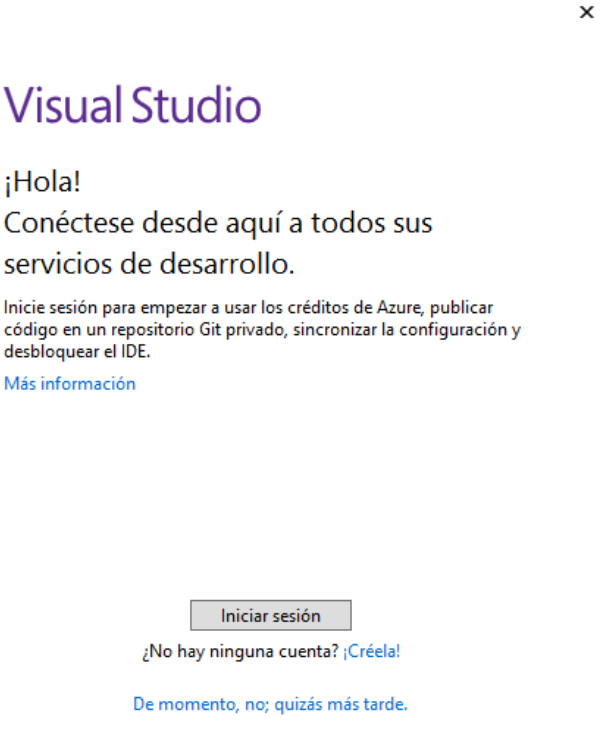

Figure 2.5: Start window that appears after the installation, we have to start session now in order to use the options of Community version with no limit.

Figure 2.6: Account Configuration inside the IDE, here we can activate the license and manage our accounts. You can use an institutional email or the Microsoft account.

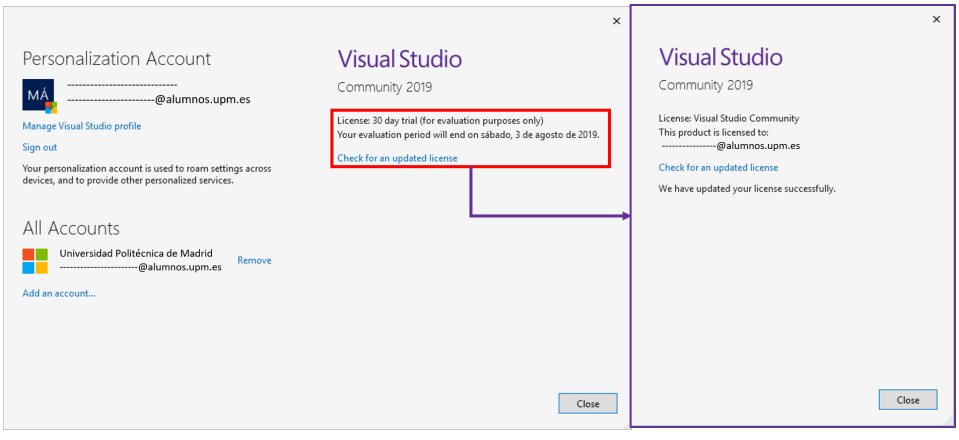

Figure 2.7: Account Configuration menu, click on *Search for an updated license* if your product is not already activated.

Together with the environment itself, the **Visual Studio Installer** has been included in our computer, we can check it in the Start menu of Windows. We have seen this tool before during the process (Figure 2.4) and we are going to open it a few more times if we install all the programming languages and tools explained in this guide. For example, all the necessary tools needed for writing Cordova projects (smartphone applications) or the minimum templates used in a basic webpage can be installed using this installer.

It is interesting to become familiarized with the different ways of installing new functionalities, tools, extensions and updates in this IDE. First of all, we can use the **Visual Studio Installer** which can be opened through Windows OS using its shortcut as an administrator and also through the IDE by clicking on *Tools/Get Tools and Features...*. Both ways show us the workloads that are already installed and the new functionalities available. We can also click on ***Tools/Extensions and Updates...*** and navigate through the big amount of complements that appears. For example, the Arduino tool that is provided by *Visual Micro* in order to write, upload, debug Arduino programs or burn microcontrollers with Visual Studio can be installed using this window. Finally, some programming languages like Python needs from the **installation of packages**, a topic that is treated further in the section 3.

Important Notice

- You are going to create a lot of projects in Visual Studio from now on. Even though the menu for creating projects is in the same place for all the programming languages (*File/New/Project...*), we have noticed that the window that appears in the different versions of the IDE (2017 vs 2019 for example) differs. The fields to fill and the name of the available projects are the same but the menu is different; in Visual Studio 2017 you will find something like Figure 5.7 while in the 2019 version something like Figure 4.2.
- The Visual Studio installer has to be periodically updated. If you are going to install anything with the VS installer and a message appears when opening the tool, do not doubt in updating it by clicking on the pertinent option. Once finished you can resume the installation process.

2.2 Configuring Visual Studio features

Now we are going to look through a list of useful functions and general common features that Visual Studio offers. All of them, when controlled, make the process of programming simpler:

1. Language
2. Line numbers
3. Navigation through files
4. Look for references
5. Open the definition of source functions and variables
6. Show description of intrinsic and new functions
7. Executing the program: *Start without Debugging*
8. Outlining blocks
9. Additional requirements
10. Common IDE configuration
11. Windows inside the IDE
12. Find in Files, Find and Replace

```
29
30          contains
31
32  subroutine sine_graph
33
34      integer, parameter:: N = 100
35      real :: x(0:N), y(0:N)
36      integer :: i;
37      real :: PI = 4*atan(1.);
38      real :: a, b;
39
40      a = 0
41      b = 2* PI
42      x = [( a + (b-a)*i / N, i=0, N)]
43      y = sin(x)
44      call scrmod("reverse")
45      call metafl ('XWIN')
46      call qplot (x, y, N+1)
```

Figure 2.8: Example of the line numbers that leads all the code lines of a program.

1. **Language**

   This manual has been written assuming that the IDE is configured in English, if you have used Spanish as default language, it can be changed in *Tools/Options/International Settings*. There are options and commands with no intuitive translation to Spanish so we recommend getting used to the English version of the IDE and the English name of the programming concepts.

2. **Line numbers**

   The line numbers are unique assignments for each of the code lines of your program (see Figure 2.8). This essential tool is used in all the programming languages mentioned in this guide, they are specially helpful when a list of errors appears in the compilation and each error must be found (in a specific file and line) in order to fix them. It is a good idea to change the configuration of the IDE to show them by default, independently of the language. Go to *Tools/Options...* and in the menu *Text Editor/All Languages/General* select the option *Line numbers*. These numbers are also an easy way of referencing your code in case of writing documentation about it.

Figure 2.9: Navigation buttons: *Navigate Backward* and *Navigate Forward*.

3. **Navigation through files**

   There are some basic functionalities that are very useful when managing big codes. The following features are activated by default in the IDE and both simplify the navigation across files.

   Navigation buttons: *Navigate Backward* and *Navigate Forward* (Figure 2.9) have the same roles as Back and Forward in Windows OS but, instead of navigate across folders, we navigate across our opened and recently viewed codes.

4. **Look for references**

   Even more powerful tools when we navigate through codes are *Go To Definition* and *Find All References*. In order to enable the possibility of looking for all references, we click on *Tools/Options.../Text Editor/Fortran/Advanced* and we select *True* in the option *Enable Find All References* (as can be seen in the Figure 2.10).

   Find All References function, as the name says, will look for the text we want in the whole project. If it is a variable, we can know how many times (and where) is used, changed, printed, etc. and if it is a subroutine we can find where is called and where is defined. In order to use it we have to click with the right button in the name of the element and click in this option.

Important Notice

Notice that Visual Studio has a very useful search engine that can be accessed with toolbars. The command needed is *Find in files* or just *Find* and it can also be minimised and put everywhere in the screen, ready to be used whenever. It is treated later in the item 12.

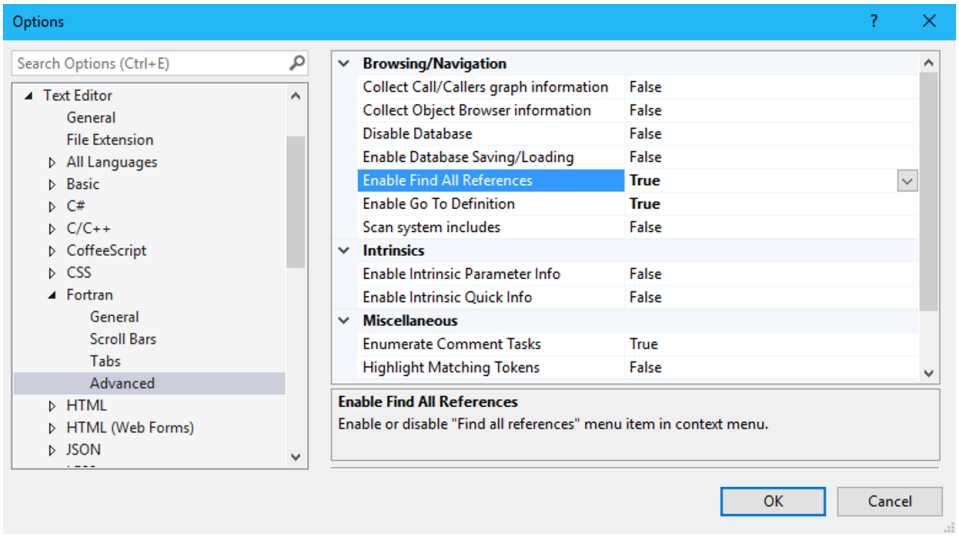

Figure 2.10: *Enable Find All References* and *Enable Go To Definition* options to activate in the Visual Studio configuration.

5. **Open the definition of source functions and variables**

   Exactly like the previous tool, *Go to Definition* function have to be enabled in *Tools/Options.../Text Editor/Fortran/Advanced* and selecting *True* in the option *Enable Go To Definition* (see the Figure 2.10).

   Go to Definition function allows us to find the declaration of any variable in our code. If we for example manage a big program with multiple files and we do not know the meaning of a variable, we can click with the right button of our mouse in the variable and select this option. The window automatically moves to the line where the variable is declared, whether it is in the same file or in a different one (opening then the specific module or file with the declaration). If this tool is used with a subroutine or function, a different file will also be opened in a new window, showing then the definition of the requested piece of code. It is important to notice that, for example in the case of Fortran, it does not work if the subroutine is in a module or library already compiled but if we have included the source code in our project, we will be able to navigate quickly through files.

6. **Show description of intrinsic and new functions**

   Enable Intrinsic Parameter Info and *Enable Intrinsic Quick Info* make faster the writing of code, both show quick information about func-

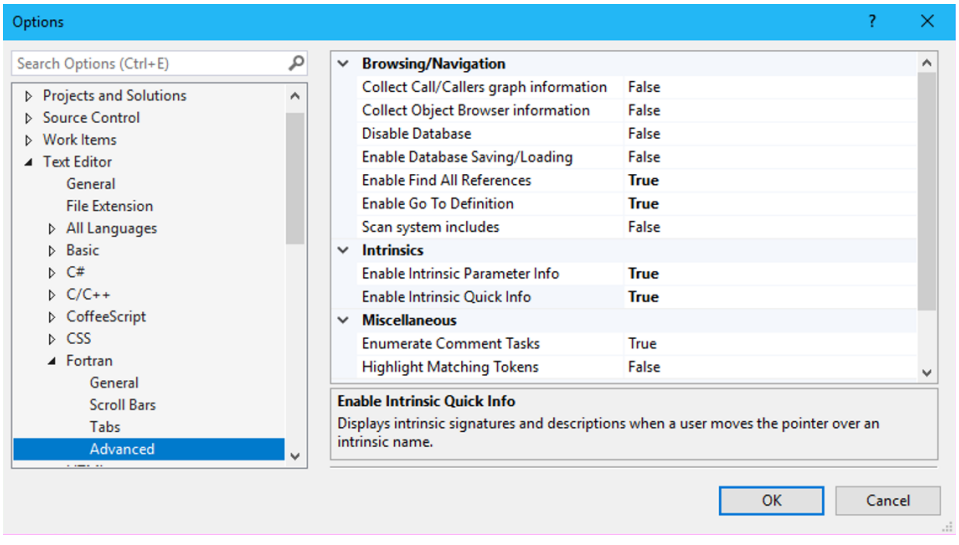

Figure 2.11: *Enable Intrinsic Parameter Info* and *Enable Intrinsic Quick Info* are both really useful when our codes become long and they are spread in multiple source files.

tions and subroutines dynamically. To activate these features go to the menu *Tools/Options.../Text Editor/Fortran/Advanced* and mark *True* in the mentioned names (see Figure 2.11). The first one, **Enable Intrinsic Parameter Info**, "*displays the signature of an intrinsic in a tooltip when a user types the parameter list start character*". This means that some essential information about the procedures and arguments will appear when typing a parenthesis after an intrinsic function or subroutine. The Figure 2.12 shows an example with the cosine function. The second option, **Enable Intrinsic Quick Info**, displays intrinsic information and descriptions when we place the pointer over an intrinsic name. In the Figure 2.13, for example, a list of arguments associated to the subroutine is shown when the pointer stays on the function `Wave_equation_2D`.

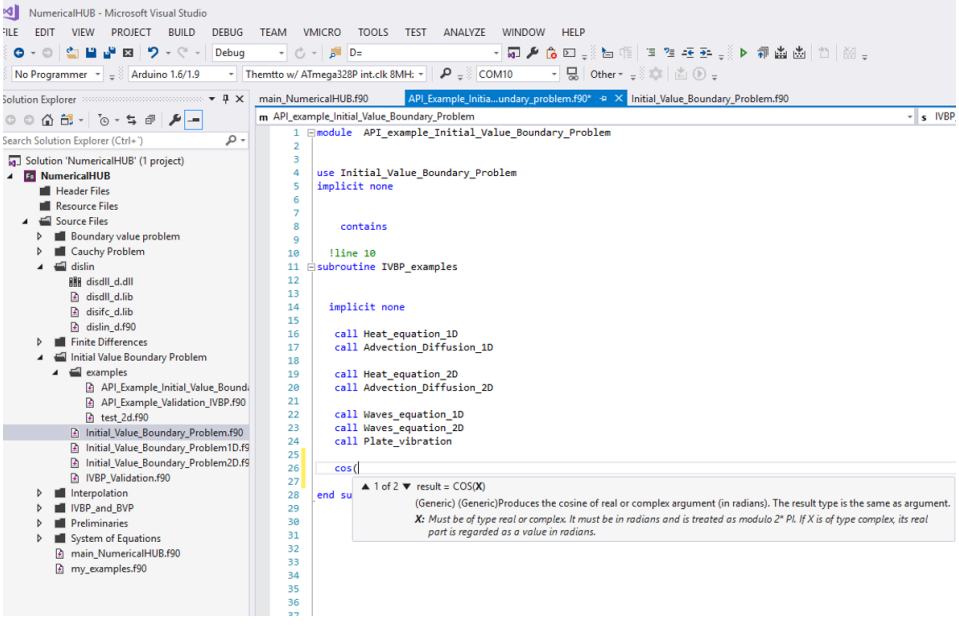

Figure 2.12: Example of *Intrinsic Parameter Info*.

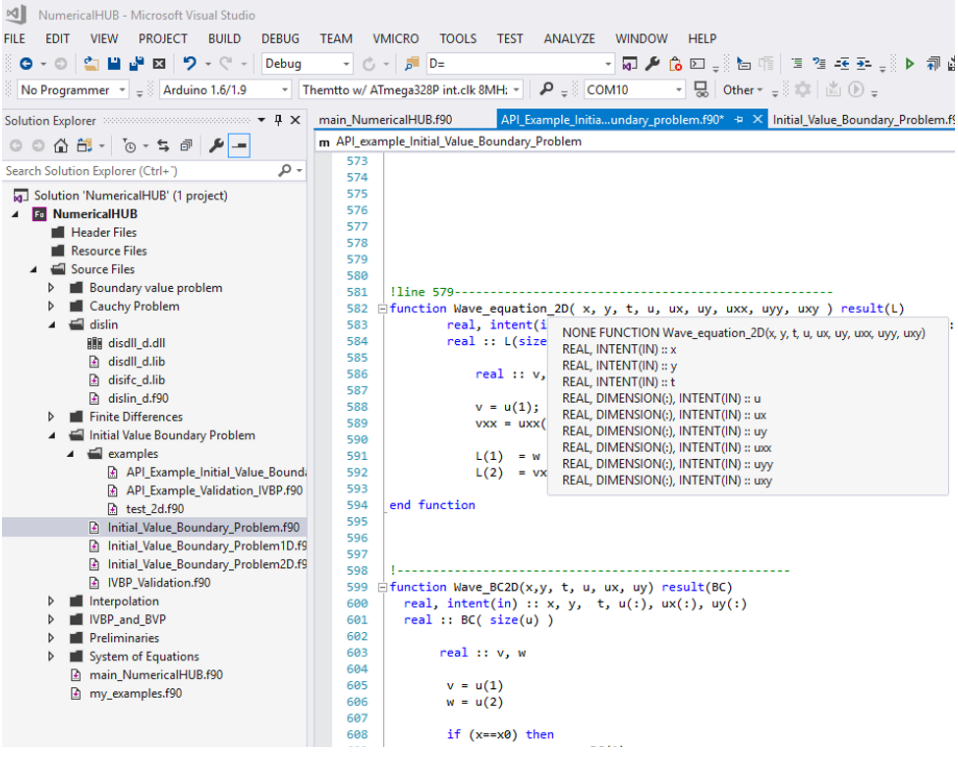

Figure 2.13: Example of *Intrinsic Quick Info*.

7. **Executing the program: *Start without Debugging***

   This essential command of Visual Studio allows to **run the application/program** in all the different programming languages treated here. Once the program is written it can be executed by clicking on *DEBUG/Start without Debugging*, pressing *Ctrl+F5* or pressing the icon

   

   of your toolbars. If you cannot see the mentioned icon in your toolbars take a look at the section 2.4.

   Notice that Visual Studio will launch your application without attaching the Debugger, you won't be able to pause the execution by breakpoints or use default debugging tools of the IDE. However, this is by far the most common method for executing programs in this guide. The behaviour of the command depends on the specific language used. For example, in Fortran it **saves** the modified files, **compiles**, **links** and **executes** all the project selected in the Solution Explorer. By contrast, in Python this command invokes the interpreter and the code is read and interpreted so the results can be shown in a command window.

8. **Outlining blocks**

   Outlining the source code is one of the most common code folding patterns that IDEs and code editors offer. This feature allows to collapse a code block to a single line normally based on the syntax of the specific language used (other mechanisms appear depending on the indentation, the tokens or the preferences of the user). In general, the user can hide and show pieces of code that define a structure e.g. functions, subroutines, loops, if statements, etc. Then the code can be easily managed (specially if there is a large amount of text in the program) since only the headers of the hidden block are seen. Getting an overview of code and navigating through is now easier without being distracted by other code. The Figure 2.14 shows an example.

   Depending on the language you are configuring the way to activate these features can vary, as an example we can go to *TOOLS/Options.../Text Editor* and deploy the Fortran options. There click on

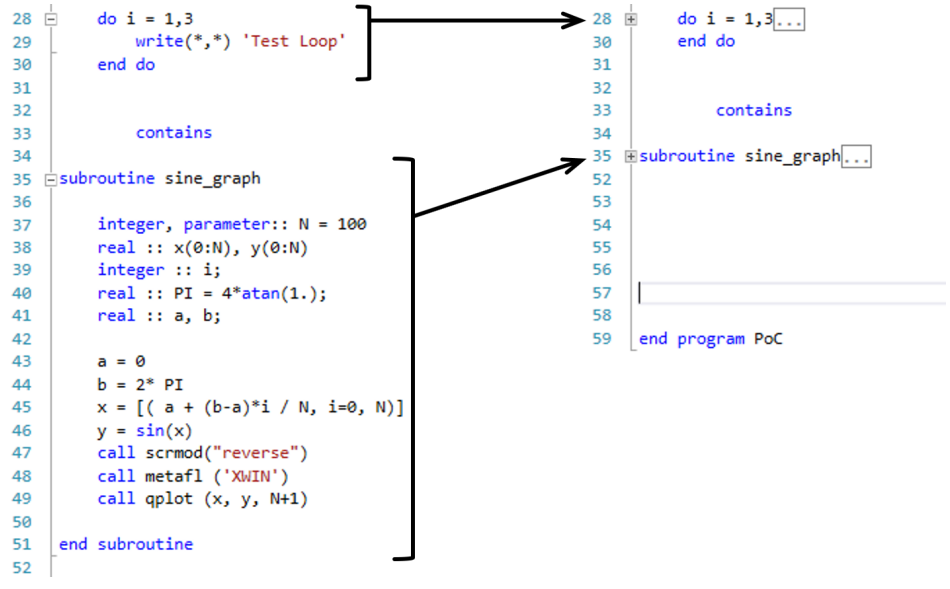

Figure 2.14: Example of two pieces of code hidden by the user, one is a loop and the other a subroutine. By clicking on + or - we can deploy or hide the code.

Advanced and in the menu called *Outlining* mark *True* in the options *Enable Outlining* (which is actually activated by default) and *Outline Statement Blocks* in case you also want to outline statements like loops. Notice that this options could be in a different menu for each programming language.

If there is a piece of code where the outlining option does not appear automatically and you want to hide it, just do the following: select all the text to be outlined, right click on the selection, deploy *Outlining* menu and click on *Hide selection*. Then, that block of text can be shown and hidden on demand.

9. **Additional requirements**

There are some tools that are common for a lot of Microsoft programs and in the case of the software development can be really useful. First of all, the different **extensions of the files** created in a software project are essential and the whole guide is full of references to some kind of files. Make sure that you have already activated in your Windows OS the option that shows the extensions of the files. If it is not activated, open any folder in your computer, click on *View* and mark the option *File name extensions*. From now on, you will see when a

file is a C++ source code, a library or maybe a Fortran module.

Part of the Visual Studio potential can be accessed using the **right button** of the mouse, exactly like the context menu of Windows OS. The options shown depend on the element where we deploy the pop-up menu and it is desirable to become familiarised with some commonly used functions. Just as an example, *Go To Definition* and *Find All References* will appear if we use the right button in the code (variables, calls to functions, intrinsic functions, etc.). If this button of the mouse is used in the solution explorer we can see different behaviour in the case of clicking on the name of projects and the name of the solution. In the project the pop-up menu mainly shows the possibilities of Building, Rebuilding and Cleaning the project, adding folders or files and opening the properties of the project (which we will see in other chapters how important they are). However, in the name of the solution, the menu shows similar options but related to the whole solution (Build, Rebuild, Clean, Add existing or new projects, Rename the solution, open the solution properties, etc.). Try opening this menu in different places to get used to it.

Exactly like in Windows OS we can make **bigger or smaller** what it is displayed in the screen, with the codes it is similar. Once you have your first piece of code opened in the IDE, try to make it bigger or smaller by pressing `Ctrl` and moving the scroll wheel of the mouse. The same effect could be achieved with the mouse pad.

Finally, two different ways of **selecting pieces of code** can be used. The common way is pointing the cursor of the mouse in one line of the code, press `Ctrl+Shift` and click on a different part, all the lines between these two point will be selected. An useful alternative in order to select a vertical block of text is achieved by pressing the `Alt` key at the same time that the left button of the mouse and dragging the mouse in those lines we want to select.

10. **Common IDE configuration**

    We can click on ***Tools/Options...*** and start customizing the IDE in the ***Environment*** menu. For example, the colour theme in the *General* section (Dark, Light, Blue, etc.) or the auto-saving information frequency in the *AutoRecover* tab can be decided here. We can also change the fonts and colours of the programming language or the default options for the *Startup* (first window that appears when opening Visual Studio).

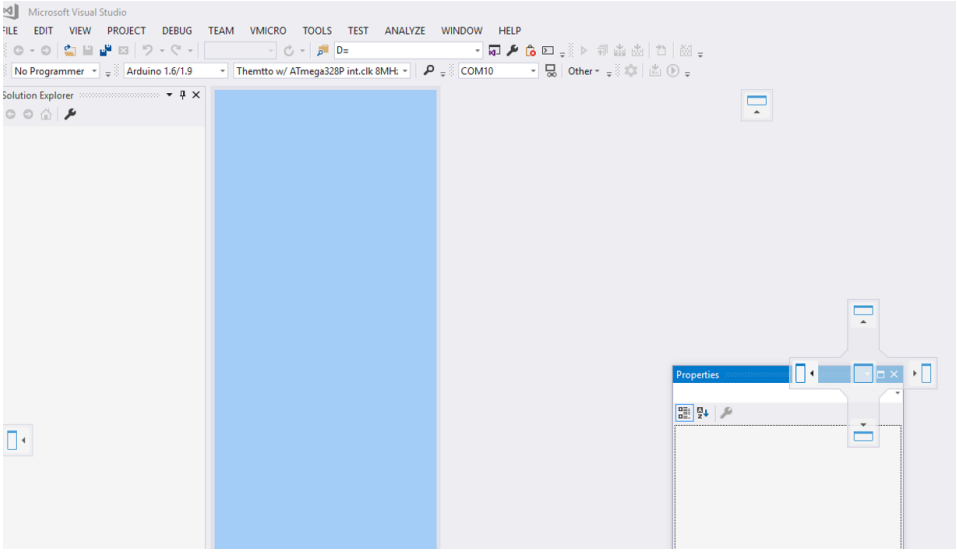

Figure 2.15: Example of windows distribution with a Fortran project already opened, *Solution Explorer* and *Output* windows are going to be intensively used.

Since some compilers (e.g. GFortran) do not recognise tabs as characters, we could find warnings each time we compile a code with tabs instead of spaces. Hence, it is recommended to check that *Insert Spaces* is marked in the *Tabs* menu of the ***Text Editor*** section (at least in the programming languages you are interested in).

11. **Windows inside the IDE**

    Related to the different **windows** around the environment, the behaviour is the same as Windows OS. We can change position, make them fixed or floating, configure different options, etc. We can decide for each window in a little arrow located in the right upper part, also close those we do not need or open new ones in ***View*** tab. *Solution Explorer*, *Properties Window*, *Bookmark Window* or *Output* are some windows we are going to need from now on. The Figure 2.15 shows an example of distribution of two essential windows: **Solution Explorer** and **Output**, we can choose our favourite position and move them. If we want to move the windows we grab it and Visual Studio will show us some default positions we could choose, we just have to drop it in the little boxes shown (or drop it in a floating point).

12. **Find in Files, Find and Replace**

    Find in Files is a powerful Visual Studio feature that allows to search in a set of files terms. It can look for the term in the entire solution, only in the opened document, in the opened project, etc. This feature is complemented by the tool *Replace in Files* which can automatically replace all the terms of a project or document by another term. Both tools can be found in the same window: *Find and Replace*.

    Open the window by clicking on *Edit/Find and Replace/Find in Files*, it will be shown at the right part of the IDE by default. Then you can change the position of the window, hide it temporally (with the option of showing it quickly), make it floating or close it, all can be done with the symbols in the top-right part of the window. Once you have the tool opened, change between both features and write a term to look for or replace. The results of the search will appear in the *Find Results* window, which is automatically opened, then it is possible to navigate through the different results in the code itself by pressing *Find previous* and *Find Next*.

    It is a commonly used feature in the IDE so we recommend to maintain the window accessible in the environment. It is treated how to show new commands and toolbars in the section 2.4. Take a look at the Figure 2.19 and check the symbol of the tool *Find in Files*. In addition, *Go To Find Combo* bar is shown in the same toolbar and both icons have to be activated since they are not shown by default.

2.3 Importing the environment configuration

This manual is accompanied by a settings file called `Exported-2019-07-03.vssettings` (or updated name) which makes automatically all the changes explained in this guide. Open this url to download the file:

`https://github.com/jahrWork/Visual-Studio-projects`

We can import this configuration and then all the options treated will be automatically applied. Go to *Tools/Import and Export Settings/Import selected environment settings* and click on *Next*, in the next window decide if creating or not a backup of your actual configuration, depending on your needs. Click on *Next* again and then in *Browse*, look for the file downloaded from the Git repository and open it. Finally click on *Next* and *Finish*.

If you change the configuration of the environment it is interesting to make a **backup copy** of the configuration in order to restore it easily if you reinstall Visual Studio, work in two different computers or change your personal computer. Go to *Tools/Import and Export Settings/Export selected environment settings*, click on *Next* twice and choose where to save it.

Important Notice

- Please notice that the file `Exported-2019-07-03.vssettings` (or similar name) that comes with this manual only stores the **IDE configuration** and not the preferences and options related to the project configurations of the different programming languages.
- It is also important to notice that the configuration file could keep **sensitive information**, which means that maybe some options store links to folders in the computer where the configuration was made and exported. In this case, when the IDE follows the links an error message appears and you must change the link to the corresponding folder in your computer. This can also happen when you create your own configuration files and use it in more than one computer.

2.4 Modifying shortcut icons

The Figure 2.16 shows some commands that are going to be highly used when programming so we recommend to enable the quick access in the toolbars. The first two of them comment and uncomment the code selected and next two decrease and increase the indentation. The fifth one is the *Start without Debugging* button (already explained) and the last active buttons are in charge of compiling and building our program. While the first compiles the current file, the second builds the project or library selected and the last builds the whole solution (which we know can contain more than one project). These functionalities will be explained in the chapters of the different programming languages.

We can personalise all the **toolbars** and **commands** that the IDE shows

Figure 2.16: Very useful commands when programming, it is advisable to show them in the toolbars.

and it can be done in two ways. Firstly, by clicking on the right part of each toolbar we see the option *Add or Remove Buttons* which deploys a list of default options for that bar (Figure 2.17). Secondly, we can click on *Tools/Customize* and we access two tabs: *Toolbars* (Figure 2.18), which can show and hide a big list of toolbars and *Commands* where we can select any of them, add and delete commands and also add new buttons that are not in the bar by default (figures 2.19, 2.20 and 2.21).

In these mentioned figures we can see that the *Text Editor*, *Standard*, *Build* and *Micro* toolbars are shown. The Figure 2.19 shows that *Standard* bar includes new commands like *Find in Files*, *Solution Explorer* and *Properties Window* while *Build* bar (figure 2.20) includes *Compile* command and the most important; ***Start without debugging***. It is necessary to have quick access to this command because most of the times we won't use the default debugger and we just want to run the code directly without debugging. Finally, the toolbar *Text Editor* (figure 2.21) is really useful while writing code because of the *Comment* and *Uncomment* commands or the new ones added, *Line Indent* and *Line Unindent*. We should take the time to organize buttons and include those we use more.

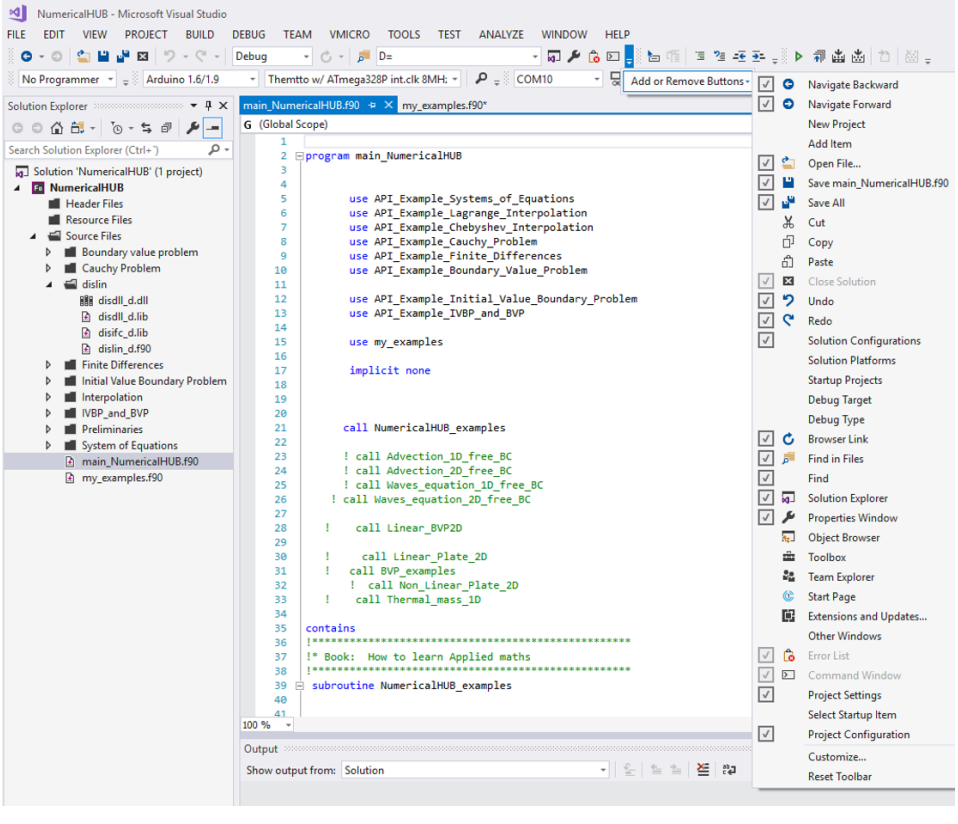

Figure 2.17: Add and Remove options for the buttons that appear in the toolbars.

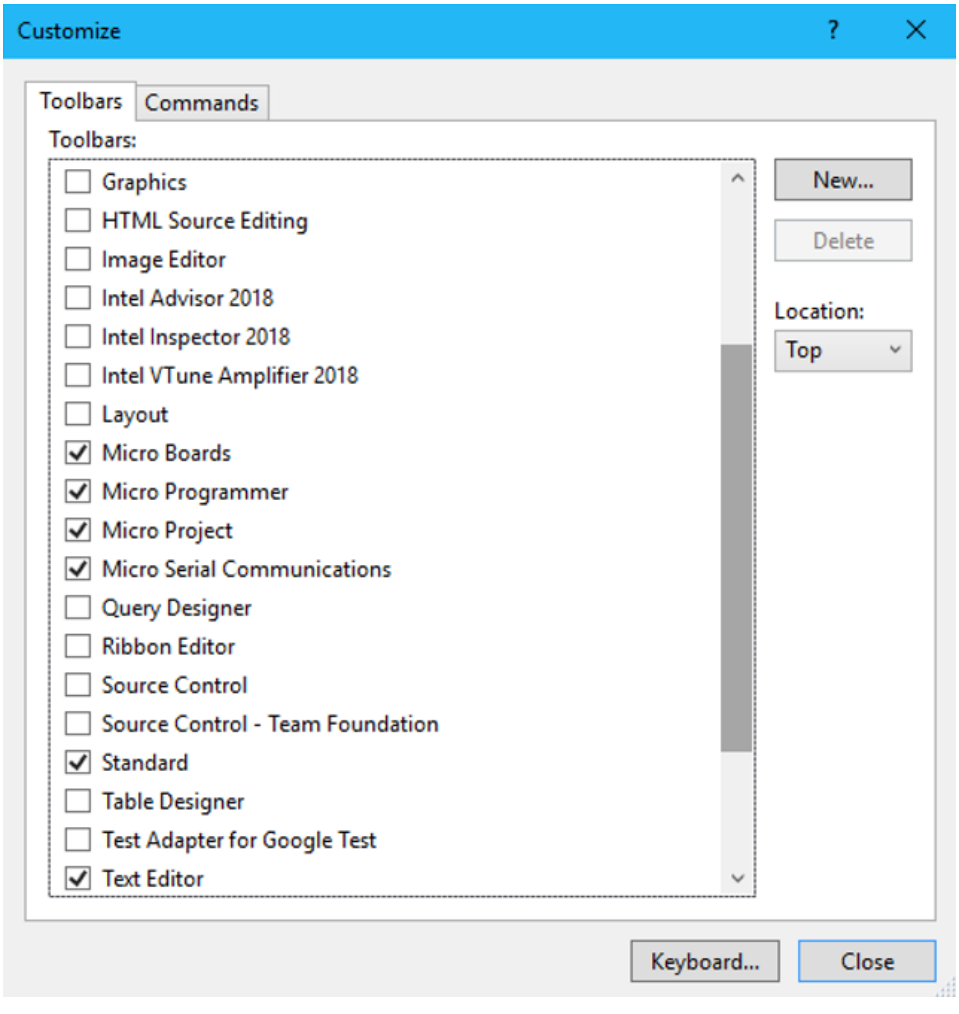

Figure 2.18: *Toolbars* tab in the menu *Tools/Customize*, it allows to show and hide toolbars for the IDE.

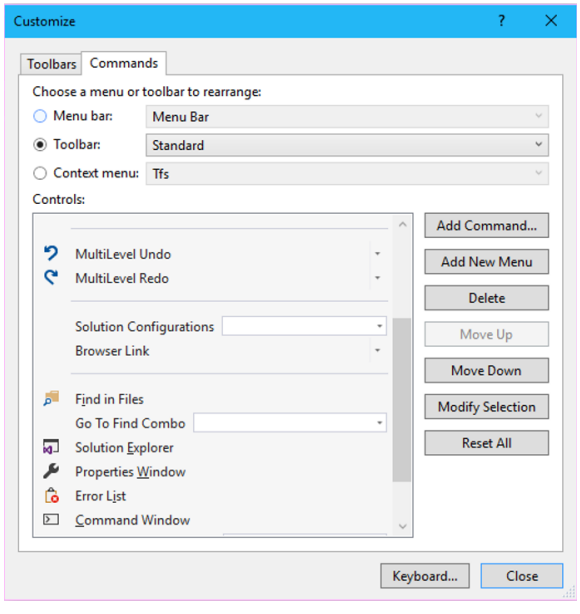

Figure 2.19: *Commands* tab in the menu *Tools/Customize*, it adds commands to the selected toolbar or menu, in this case the *Standard* Toolbar.

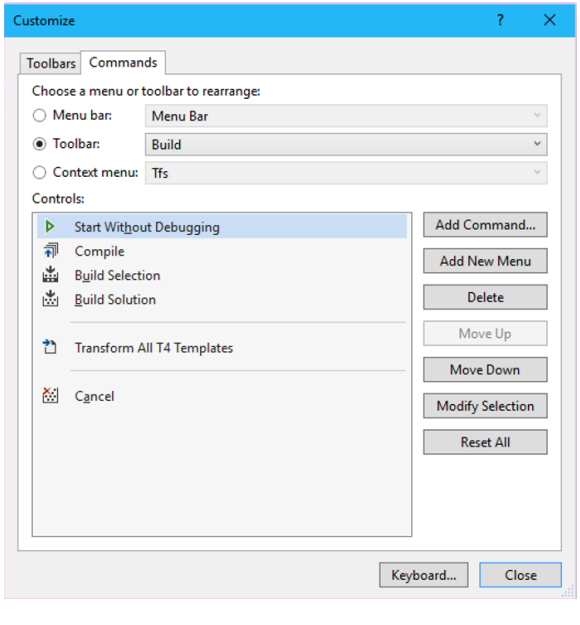

Figure 2.20: *Commands* tab in the menu *Tools/Customize*, it adds commands to the selected toolbar or menu, in this case the *Build* Toolbar.

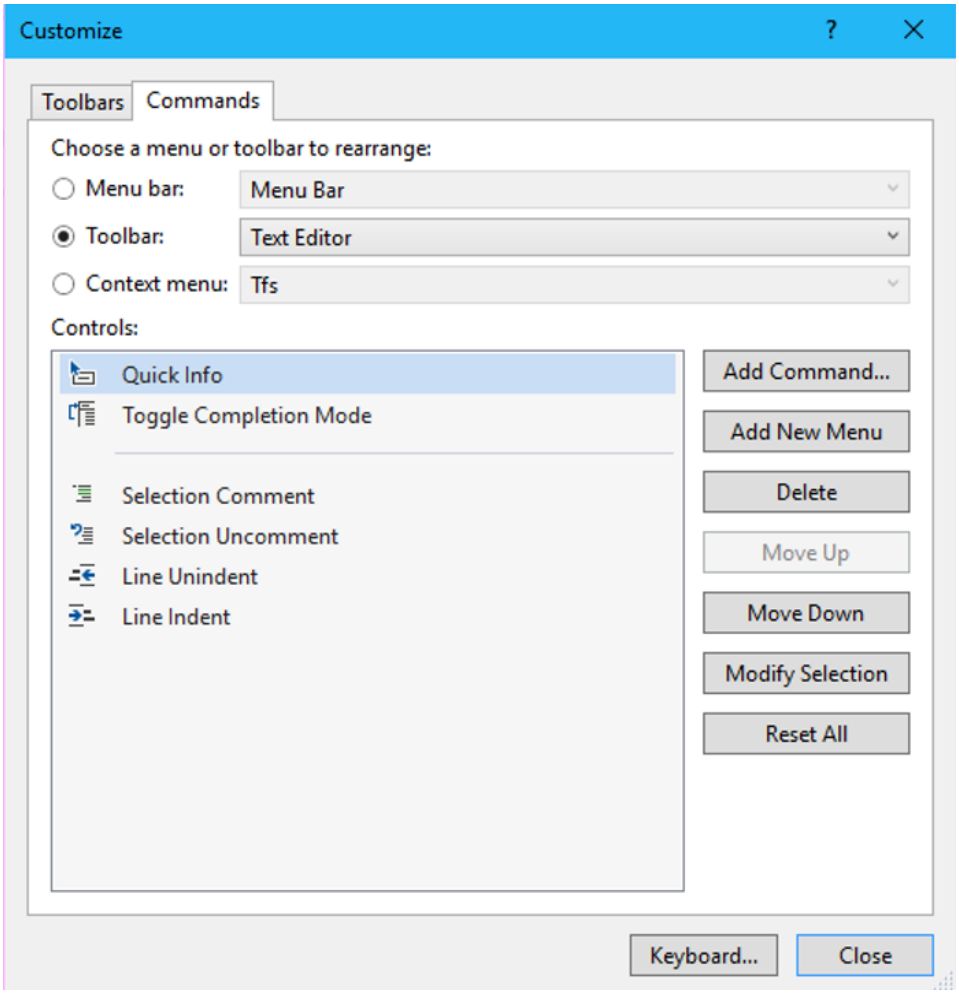

Figure 2.21: *Commands* tab in the menu *Tools/Customize*, it adds commands to the selected toolbar or menu, in this case the *Text Editor* Toolbar.

2.5 Enterprise distribution of Visual Studio

If you are interested in the Enterprise distribution of Visual Studio you can download the installer from the same webpage used for the Community version (https://visualstudio.microsoft.com/es/downloads/) or follow all the steps mentioned below. In this case you need an academic license to activate the product. We are going to see how to obtain that license using *Azure Dev Tools for Teaching*, a Microsoft repository created for university departments and students.

Obtaining an academic license and the installer for Visual Studio Enterprise.

1. Click directly on (Figure 2.22):

   https://azureforeducation.microsoft.com/devtools

2. Click on the blue button *Sign In* and Initiate session using your institutional email (finished with *@alumnos.upm.es*, Figure 2.23).

3. You will be redirected to the authentication service of the UPM. Write your email once again and write the password for that email account (Figure 2.24).

4. After accepting a message that allows you to maintain the session opened, you access the Microsoft Azure environment with the tab *Education* shown (Figure 2.25).

5. There, you may see the last release of Visual Studio in the right part of the window. However, you can also click on *Software*, look for *Visual Studio Enterprise* in the list or write it in the search bar.

6. Click on the name of the tool and you will have access to the download of the installer and the Key for activating the product (Figure 2.26).

7. Download the installer for the Enterprise distribution and keep safe the Key, you are going to use it later.

In order to install Visual Studio Enterprise version with an academic license in Windows you can follow the same process seen in the installation

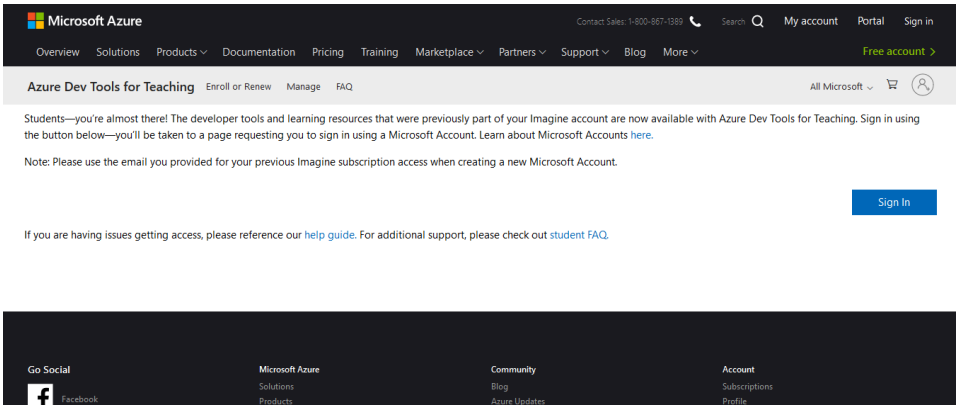

Figure 2.22: Main webpage of Microsoft Azure Dev Tools for Teaching where we have to click on *Sign In*.

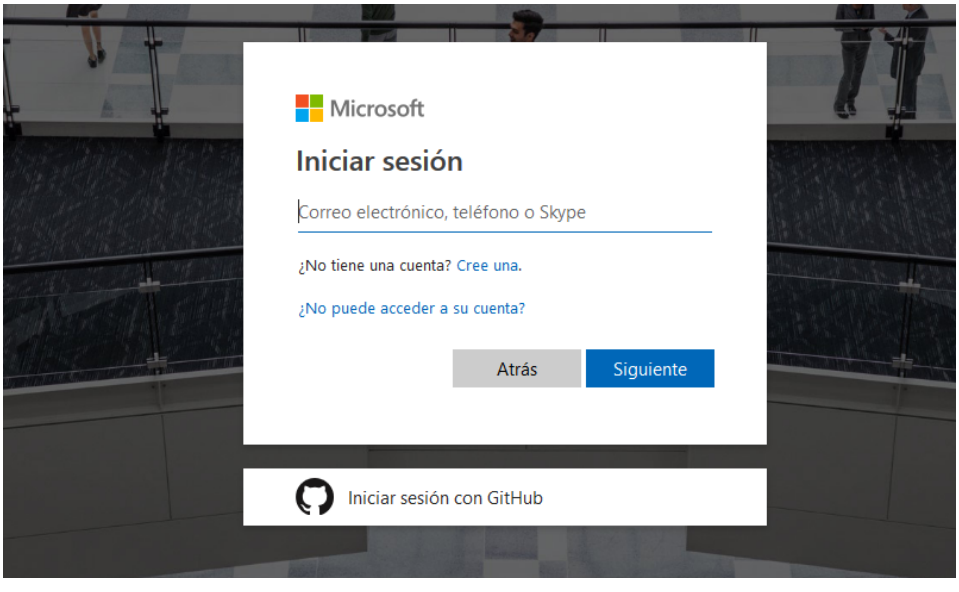

Figure 2.23: Start session window, the institutional email must be used here (finished with @alumnos.upm.es).

Figure 2.24: Authentication service of the UPM, you must use your institutional email and the same password used in the email account.

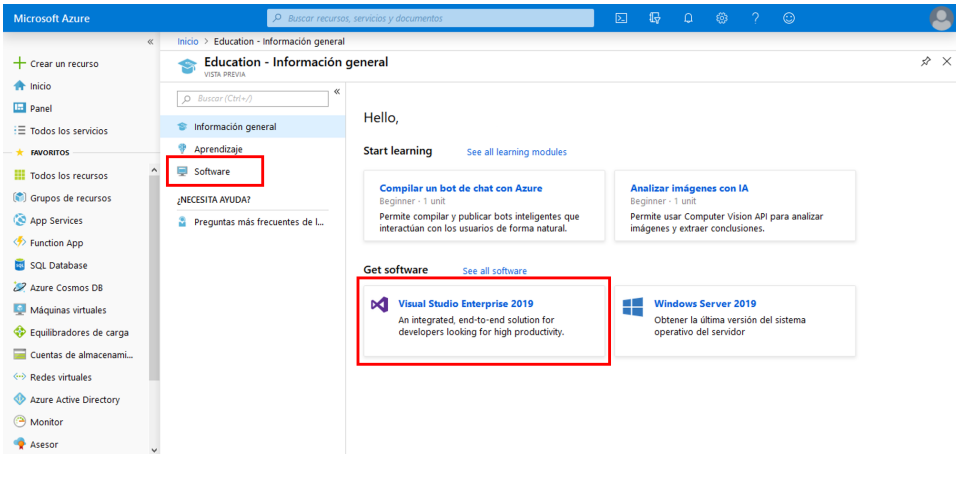

Figure 2.25: Start window of Microsoft Azure, Visual Studio Enterprise is already shown but we could look for it in the *Software* space if needed.

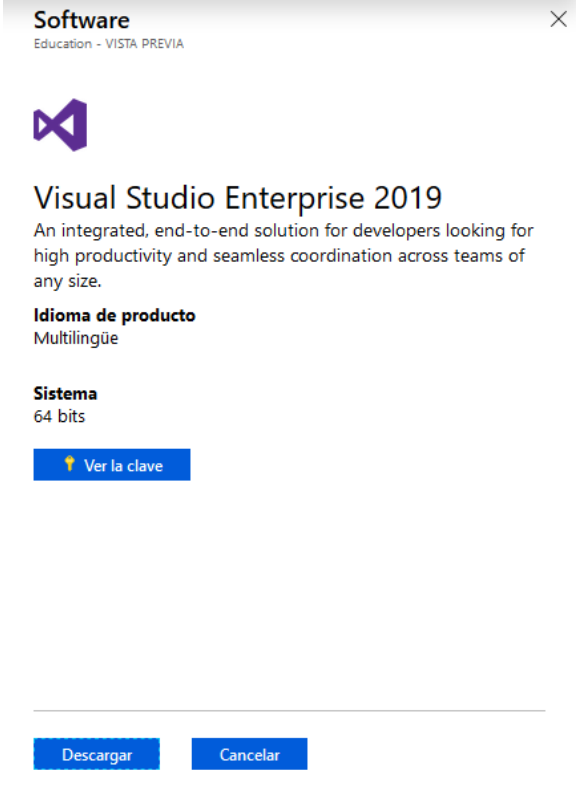

Figure 2.26: Menu where we can download the installer of Visual Studio Enterprise and consult the key used to activate the tool.

of the Community version, executing this time the installer of the Enterprise version. The name now is similar to `vs_enterprise_xxxxx.exe`. As a difference, in the step 7 you have to activate the product using your Key:

7. Notice that in the top right part of the IDE it is shown that you have already started session. If you click on the letter of your name and *Account Configuration* (see figure 2.6) you will see that your credentials are activated. In the right part of that window it may say that you are using a 30 days license, click on *Unlock with a Product Key* (see figure 2.27) and it will show a window waiting for the Key obtained (see figure 2.28). Click on *Apply* and the message "Product Key applied" should appear.

8. Your IDE is ready to work now.

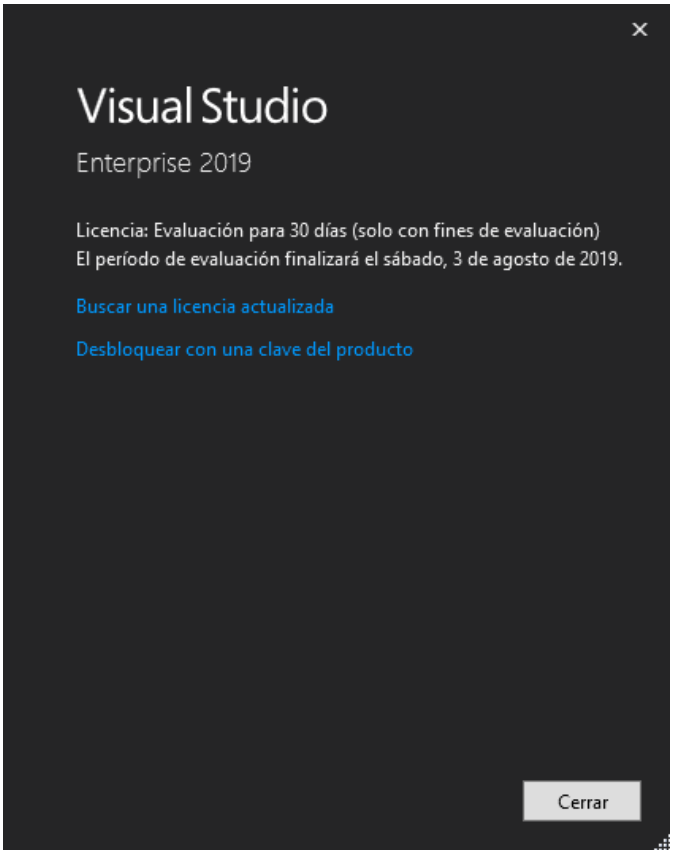

Figure 2.27: License information in the Account Configuration window in the IDE, before activating the product you will have a 30 days license.

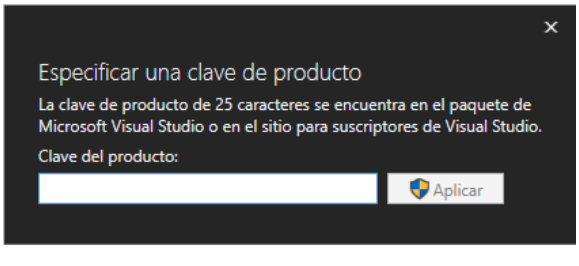

Figure 2.28: Activation window, copy here the key obtained with *Azure Dev Tools for Teaching* and click on *Apply*.

Chapter 3

Python Projects

3.1 Installing Python Interpreter

Visual Studio supports Python projects and solutions by selecting during the Visual Studio installation the **Python Development** workload. If we do not have installed it yet, we can always do it when needed thanks to the Visual Studio Installer tool. In order to do that:

1. Open *Visual Studio Installer* application as an administrator.

2. Click on *Modify* on the space reserved for your currently version of Visual Studio (Figure 3.1).

3. Now, in the *Workloads* tab, select *Python Development*. Check that the first three elements of the optional modules to be installed are selected (`Python 3 xx-bit (3.x.x)` included), it can be checked on the right part of the window as it is shown in the Figure 3.2.

4. Click on *Modify* and wait until the installation has finished.

Figure 3.1: Main window of the Visual Studio Installer with the release Enterprise 2017 shown.

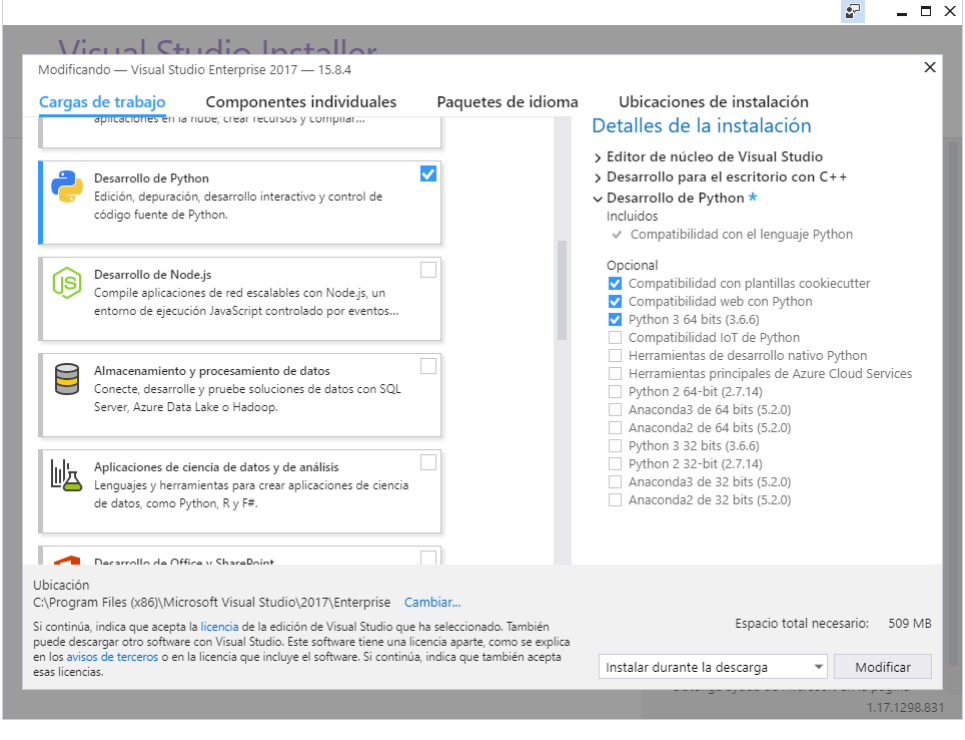

Figure 3.2: Workloads ready to be installed using the Visual Studio Installer. The Python development is already selected and the needed characteristics chosen.

3.2 Create a Python project

We will see during this guide that the process for creating a new project is similar for all the programming languages, in the case of a Python project:

1. Open your version of Visual Studio.
2. Click on *File/New/Project...*
3. In the *Python* menu select *Python Application*.
4. Choose a name for the project (Figure 3.3).
5. Change the *Location* of the solution, a directory will be created there.
6. Select option *Create directory for solution*.
7. Choose the name of the solution, it does not have to be the same as the name of the project.
8. Click on *OK*.

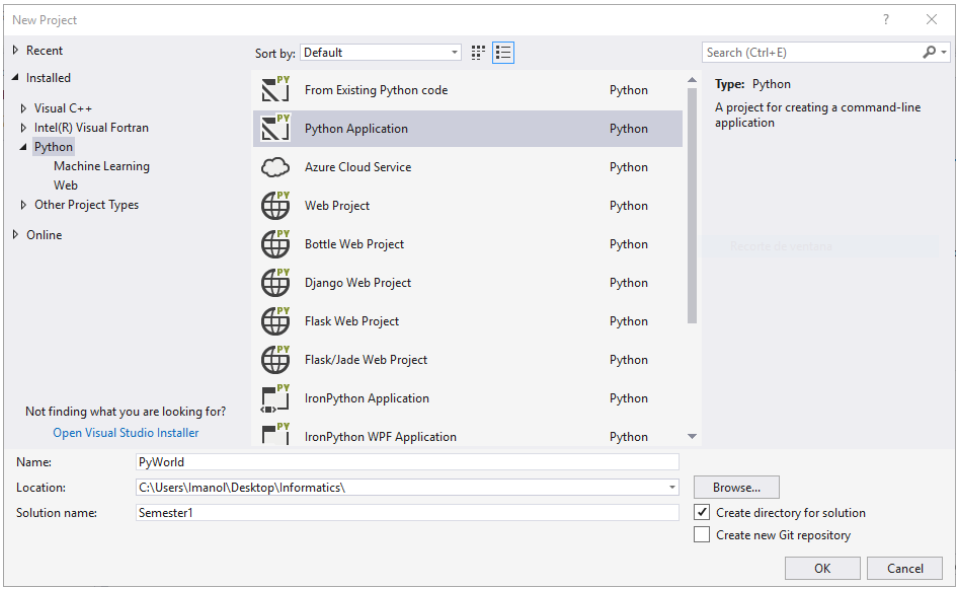

Figure 3.3: Window *New Project* where creating a Python project.

3.3 Execute the "Hello World" example

This first example ensures that the installation has finished correctly. Like in the rest of the programming languages, it writes a "Hello World" message on the console. In this case, there is not a template already written with this example so we will write it from scratch. Open the project created in the previous section, you will find a `.py` file with the same name as the project, open it and write the following code:

```
print("Hello World!")
```

Now, in order to execute this Python script, the interpreter has to be called for translating it to machine code dynamically:

1. Click on *DEBUG/Start Without Debugging* or click on the corresponding icon. The program is then executed on a new command prompt.

Important Notice

More project/solution examples with Python language can be downloaded from:
https://github.com/jahrWork/Visual-Studio-projects.

3.4 Installing and removing Python Packages

Python can download projects developed by other people and include them in our own solutions. These projects or **packages** have been tested by experts to ensure a proper performance and can help us while coding with no need of developing the code from scratch. In order to install a package follow the next steps:

1. Open your Python Project.
2. If the Solution Explorer is not opened, click on *View/Solution Explorer*.
3. Unfold the *Python Environments* menu inside the directory of the specific project in the solution explorer.
4. Right click on `Python x.x (xx-bit)` and select *Install Python package...* (Figure 3.4).
5. Type the *Package name* to be installed and press on *Enter* (Figure 3.5).
6. A new window will ask you for **Administrator privileges** as shown on Figure 3.6. Select the third option: *Always elevate when installing or removing packages* and Visual Studio will grant permissions automatically.
7. The process of downloading and installing starts.

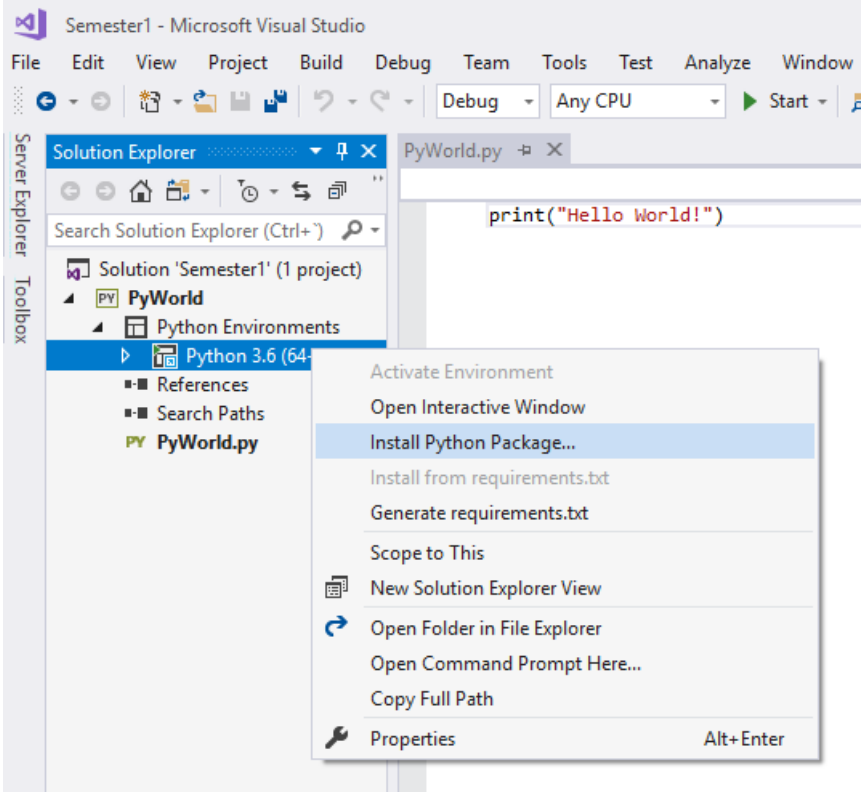

Figure 3.4: First step to install new Python packages on the Python environment.

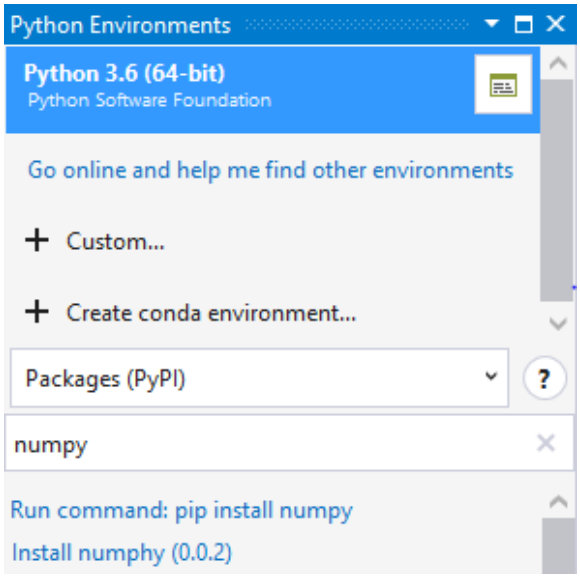

Figure 3.5: Second step to install new Python packages on the python environment. Write here the name of the package to be installed.

If a package is not needed anymore, it can be removed from the environment by following the next steps:

1. Open a Python Project.
2. Unfold the *Python Environments/Python x.x (xx-bit)* menu in the Solution Explorer.
3. Right click on the *Package name* to be removed and select *Remove* (Figure 3.7).

3.5 Include modules & packages from other Projects

Python Projects can "import" modules and packages from other projects on separated directories. With this way we can have the latest updates of an actively developed module or package without duplicating code. There are two ways for including external modules and packages:

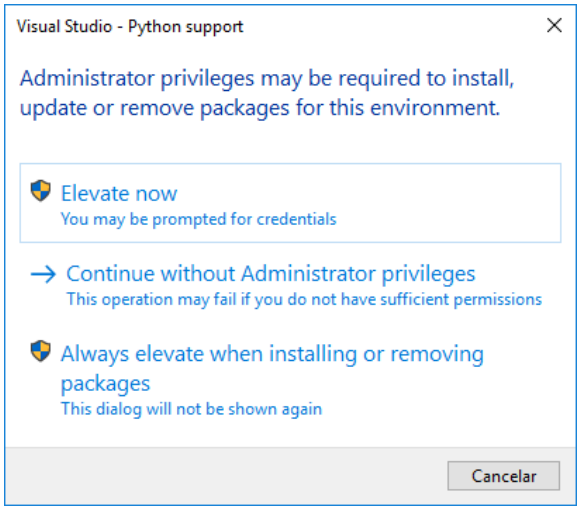

Figure 3.6: Confirmation of the administrator privileges in order to install or remove packages. Click on *Always elevate when installing or removing packages*.

Modules/Packages from the same solution

- In the solution explorer, in a project, right click on *References* and select *Add Reference...*
- Check Python Projects in order to import their modules and packages as shown on Figure 3.8.

Modules/Packages outside solution

- In the solution explorer, in a project, right click on *Search Paths* and select *Add Folder to Search Path...*
- Select the root folder where the Python module/package to be imported is located (see Figure 3.9).

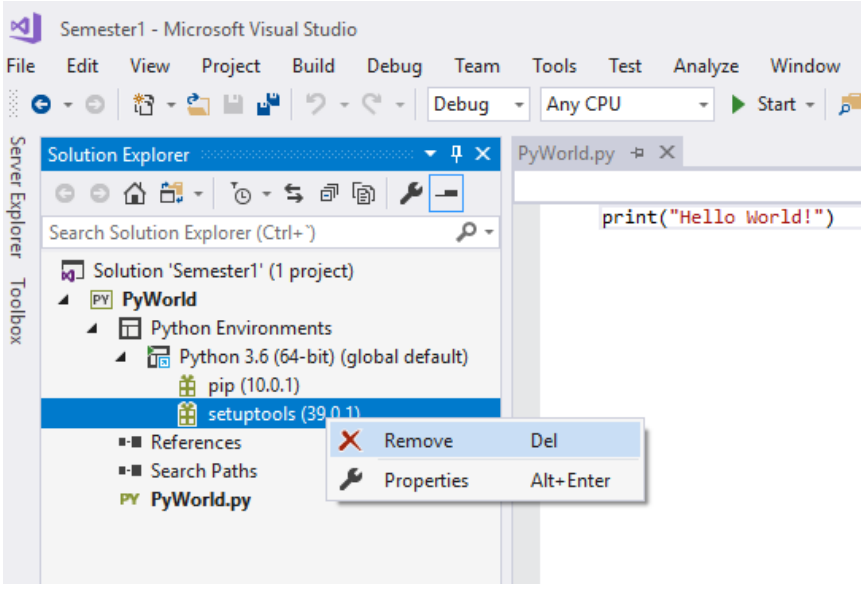

Figure 3.7: Option reserved for removing those Python packages not needed from the Python environment.

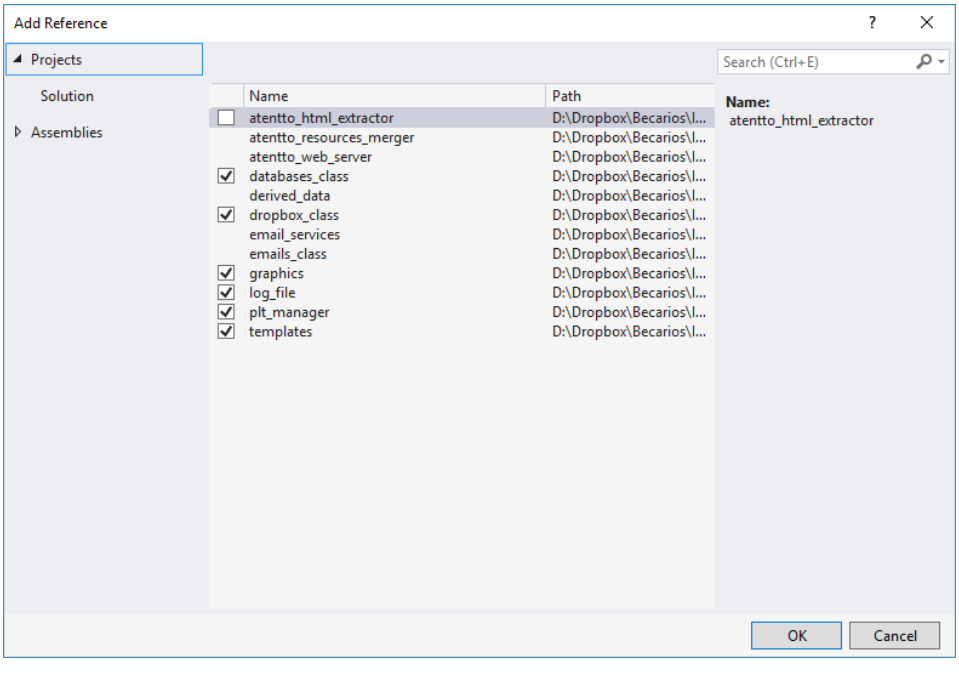

Figure 3.8: Available Python Projects for importing packages. Each project contains a package with the same name on its root directory.

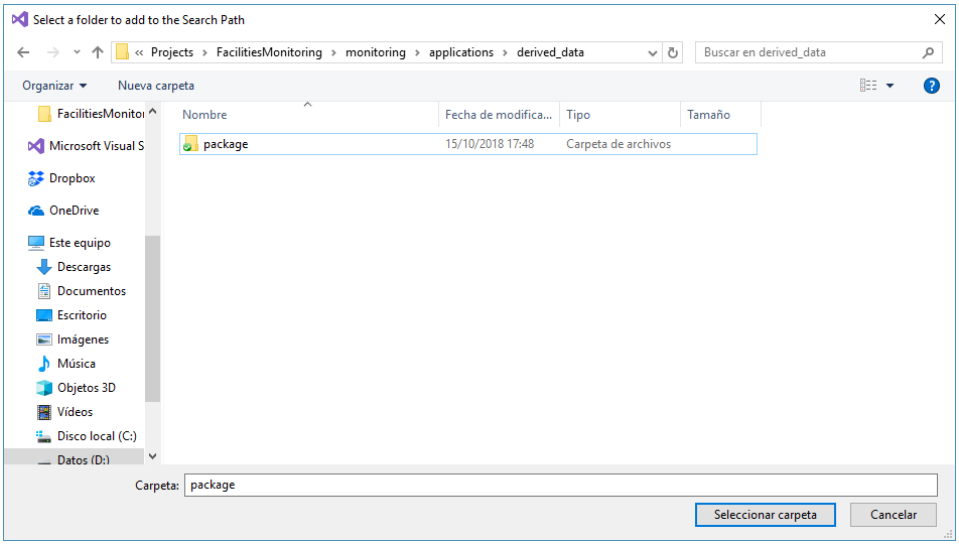

Figure 3.9: In this window we can look for the path of the package to be used by the Python Project.

Chapter 4

Arduino Projects

4.1 Install Visual Micro plug–in

Visual Micro is a plug–in for Microsoft Visual Studio with C++ installed. It allows to create Arduino IDE compatible projects to develop embedded software using the power of the Visual Studio IDE.

To install this plug–in, follow the next steps:

1. Open your release of Visual Studio.
2. Click on *Tools/Add ext and updates....*
3. Select *Arduino IDE Visual Studio.*
4. Close Visual Studio.
5. Visual Micro plug–in will start to install.
6. After Arduino IDE is installed, open Visual Studio.
7. Click on *VICRO/Ide/configure IDE location...* and specify where your Arduino folder is located.
8. Click *OK.*

4.2 Create an Arduino Project

1. Open your release of Visual Studio.
2. Click on *File/New/Project...* (Figure 5.6).
3. In the *Visual C++* menu select *Arduino Project*.
4. Change *Name* (project name) to "HelloWorld".
5. Change the *Location* of the solution to */Desktop*.
6. Change the *Solution name* to "ArduinoProjects".
7. Select option *Create directory for solution*.
8. Click *OK*.

Visual Studio will create a folder in the Desktop with the name "ArduinoProjects" to hold the complete solution. Inside this folder another folder named "HelloWorld" will contain the project `HelloWorld.ino`. In this case this project contains only one file. In order to be compatible with the Arduino IDE this folder and the Arduino `.ino` file should have the same name.

4.3 Execute the "Hello world" with Arduino

Now it is a good practice to determine if the Arduino compiler is properly installed. To verify the installation, we need an Arduino board and we will proceed with the following steps:

1. Download the latest version of the Arduino IDE and unzip its content to same folder.
2. Open the visual studio solution that we have created in the last section.

3. Select the location of the Arduino IDE from the VMICRO tab of the Visual Studio window.

4. Select the specific Arduino board that we have.

5. Plug the Arduino board with a USB wire.

6. Select the Serial port in which the Arduino board is discovered: COM1, COM2, ...

7. Copy and paste the following code in the `HelloWorld.ino` file:

```
// it configures the serial port with 9600 bauds
void setup()
{
    Serial.begin(9600);
}

// every 5 seconds a Hello world message is printed
void loop()
{
    Serial.println("Hello world");
    delay(5000);

}
```

8. Click on *VMICRO/Build and Upload*.

9. After the output windows shows: "The upload process has finished", click on the serial monitor for the selected port (Figure 4.1). It will open a serial COM17 window showing the results of the program. Every 5 seconds, the message : "Hello World" will appear on the serial terminal.

Important Notice

More project/solution examples with Arduino can be downloaded from https://github.com/jahrWork/Visual-Studio-projects.

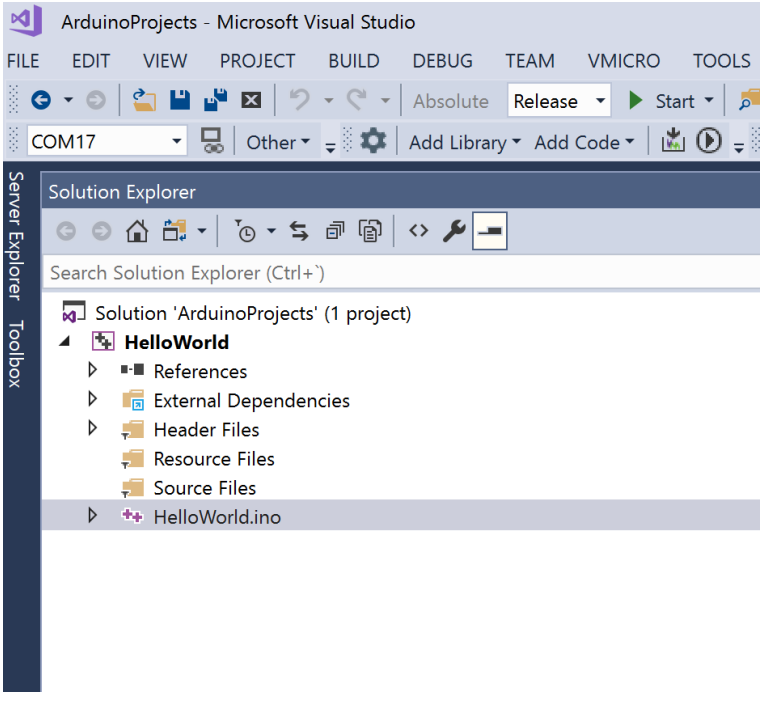

Figure 4.1: Icon to open the Serial monitor close to the selected port: COM17

4.4 Configuring complex projects

Visual Studio is useful when we are dealing with very complex projects in which many libraries of different codes files are involved.

Besides of the internal libraries of the Arduino IDE, Visual Studio can share external libraries with the Arduino project as any other programming language. To specifically configure some project, we propose to start from a simple project to increase gradually the complexity by adding different external libraries.

To do that, we will start from a new project named: *Automation*. This project is going to blink a led by using an external file or shared item named: *Leds*.

1. Open the visual studio solution that we have created: *ArduinoProjects*.
2. Add a new Arduino project to the same solution named: *Automation* as we did to create "HelloWorld".
3. Add a new share item: *File/New/Project/Add/Visual C++/Arduino Shared Code Project*.
4. Select *Add to solution*.
5. Name *Leds* the new project.
6. Create two files: `LedClass.h` and `LedClass.cpp` inside this share item.
7. Copy and paste the following code in the `Ledclass.h` file:

```
class Led
{

public:

    int pin;

    void on();
    void off();

};
```

8. Copy and paste the following code in the `Ledclass.cpp` file:

```
#include "LedClass.h"
#include"Arduino.h"

void Led :: on()
{
    digitalWrite(pin, 1);
}

void Led :: off()
{
    digitalWrite(pin, 0);
};
```

9. Right click on *References* of the project *Automation*, then click on *Add references* and select *Leds*.
10. Click on *VMICRO/Build and Upload*.

4.5 Installing C++ Compiler

If you have installed Visual Studio following this guide you do not have to install anything else since the C++ compiler was included in the first steps.

In case you do not have the necessary tools already included in the IDE open the Visual Studio Installer as an administrator and click on *Modify*. We have to select in the *Workloads* (in the tab reserved for Windows) the default tools related to "Development for Desktop with C++" (see Figure 2.4). We click on *Modify* and wait for the process to finish the download and installation.

4.6 Create a C++ project

To create a C++ project, proceed with the following steps:

1. Open Visual Studio.
2. Click on *File/New/Project...* (Figure 5.6).
3. Deploy the field *Language* and select *C++*, then click on *Empty Project* (see Figure 4.2).
4. Change the *Project Name*, for example "HelloWorld" in this case (see Figure 4.3).
5. Change the *Location* of the solution.
6. Change the *Solution name* to "C++Projects".
7. Click on *Create*.

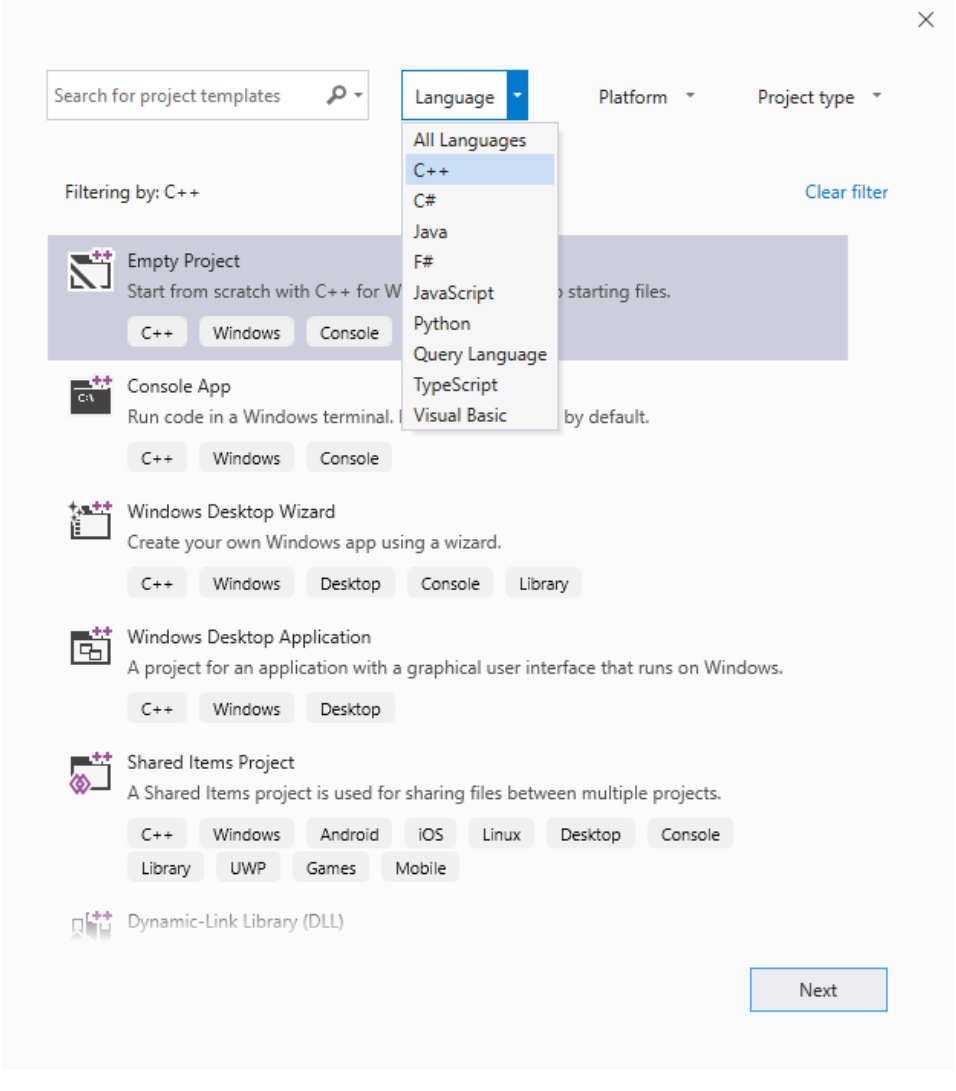

Figure 4.2: Window for creating a project in Visual Studio 2019 Community version. We have filtered by C++ and chosen Empty Project.

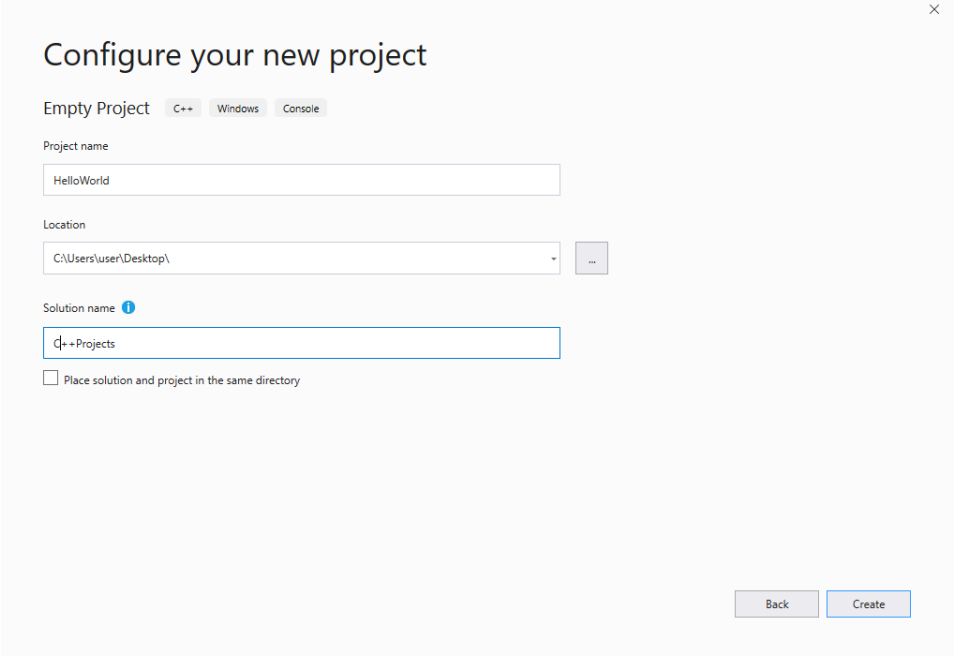

Figure 4.3: In this window we can create a new Project, a new Solution and specify the names for both and the location for the folders.

4.7 Execute the "Hello world" example with C++

Now we are going to execute our first example in C++, the Hello World program. Firstly, include a new source file to the project by clicking with the right button in the folder "Source Files" and on *Add/New Item...* (see Figure 4.4). Secondly, choose *C++ File (.cpp)* in the tab *Installed/Visual C++*, change the name of the file to `HelloWorld.cpp` and click on *Add* (Figure 4.5). You will find a blank file where copying the following code:

```
#include <iostream>
using namespace std;

int main()
{
    cout << "Hello, World!";
    return 0;
}
```

Listing 4.1: Hello World example in C++.

To compile, link and execute the program follow the next steps (the box in Figure 4.6 shows the icons for building and executing the project):

1. Click on *BUILD/Build solution* or click on the corresponding icon. The program is compiled and linked (Figure 4.6).
2. Click on *DEBUG/Start Without Debugging* or click the corresponding icon. The program is executed.

Important Notice

More project/solution examples with C++ language can be downloaded from:
https://github.com/jahrWork/Visual-Studio-projects.

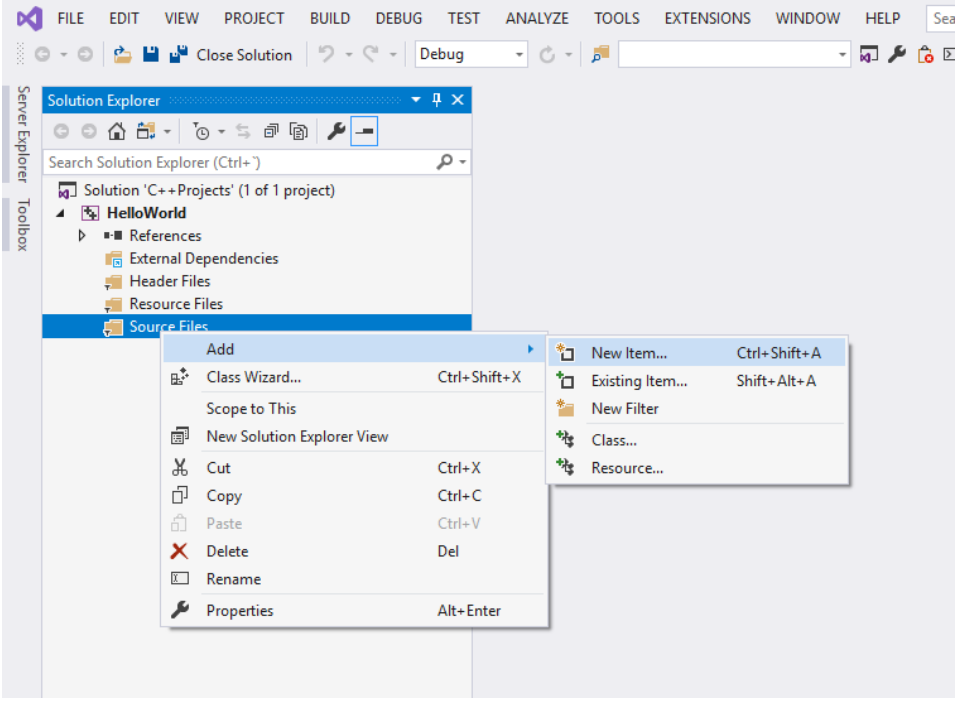

Figure 4.4: Menu where adding new or existing files to our project.

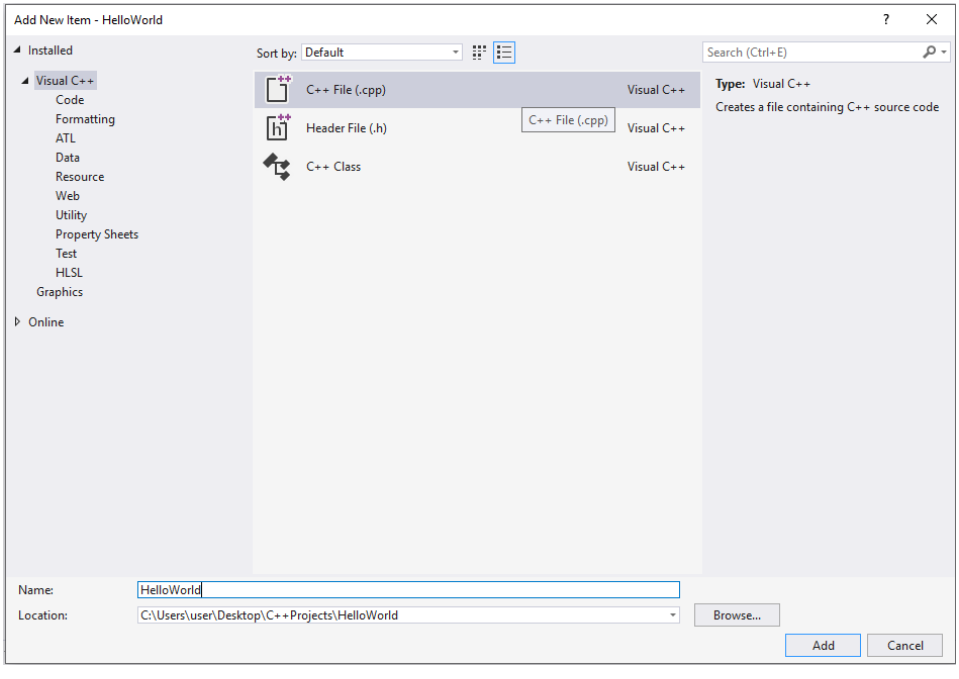

Figure 4.5: Window where choosing the C++ File. Name it “HelloWorld” and add it to the project. The `.cpp` file stores the source code for a C++ program.

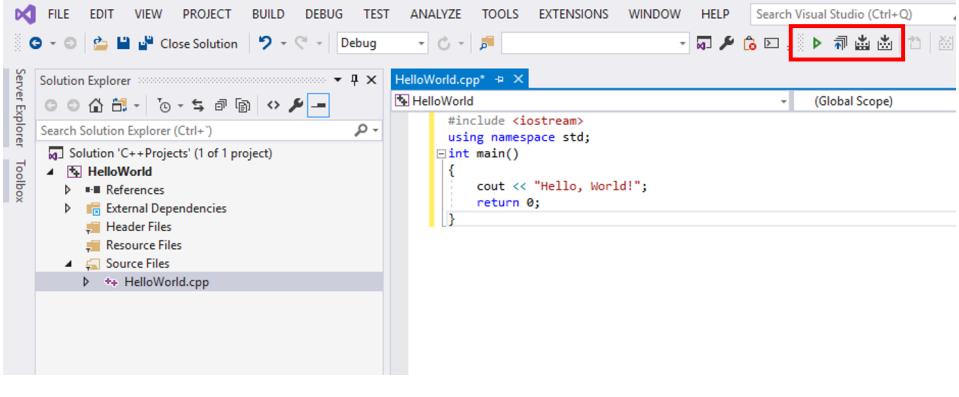

Figure 4.6: Final aspect of the project, click on *Build the project* and execute it without debugging to see the result.

Chapter 5

Fortran Projects

5.1 Installing Fortran Compiler

As it has been said before, we need now the compiler of the programming language we are interested in. Hence, Intel® Fortran Compiler must be installed in the computer and it should be recognized by the Visual Studio environment (previously installed). We are going to use an academic license, follow the steps below:

Important Notice

- Please notice that Visual Studio **MUST be installed in the first place**. Do not continue with this installation if the IDE is not already installed in your computer.
- The packages used for the installation of Intel® Fortran Compiler are the *Intel® oneAPI Base Toolkit* and *Intel® oneAPI HPC Toolkit*. Try to install always the latest version of the compiler offered by Intel in order to ensure the compatibility with the latest version of Visual Studio.

a) Downloading *Intel® oneAPI Base Toolkit* for students:

1. We first go to the official website of Intel Developer Zone, click on the next url to see the available tools:
   https://software.intel.com/content/www/us/en/develop/articles/free-intel-software-developer-tools.html
2. First of all, sign in if you already have an Intel account, you can find the option in the right upper part of the window. If you do not have an Intel Account create it by clicking on *Sign In* and then *Sign up here*.
   - Fill in all the Personal Information (you can use your institutional email in the field *Business Email Address*). Click on *Next Step* when finished.
   - Click on *Submit* for the Terms and Conditions.
   - Verify your e-mail address by clicking on the link available in the verification email you have received.
   - Sign In now with your new account in the previous url.
3. In order to obtain first the Base Toolkit click on *Get the Base Kit* (Figure 5.1).
4. Now choose from the available options, in this case *Windows/Web & Local/Local* and click on *Download* (Figure 5.2).
5. Save the file in your computer.

b) Downloading *Intel® HPC Toolkit* (High-Performance Computing) for students:

1. Follow the same process in order to download the HPC Toolkit clicking now on *Get the HPC Kit* in the third step (Figure 5.1).
2. Save the file in your computer.

c) Installing both tools:

1. Now you have both installers downloaded in your computer. Double click first on the installer of the Base Toolkit, named something similar to:
   `w_BaseKit_p_XXXX.X.X.XXXX_offline.exe`.
2. Accept the installation as an administrator and click on *Extract* the temporary files needed by the installer. Wait until the process finishes.

3. In the first window that appears click on *Continue* and in the next one mark the option *I accept the terms of the license agreement*, click on *Continue* for the Recommended Installation (Figure 5.3).
4. Now the installer shows warnings related to the needed modules (this step should not block the installation so we continue with the process), click on the right arrow.
5. Make sure the integration with your current version of Visual Studio is marked (Figure 5.4), click on the right arrow.
6. In the last window you have to decide about consenting or not the collection of private information, choose your options and click on *Install*.
7. Wait until the process finishes (it can be slow) (Figure 5.5).
8. When finished click on *Go to Installed Products* and then close the installer.
9. Follow the same process described with the other installer, named something similar to `w_HPCKit_p_2021.1.0.2682_offline.exe`.

After the installation has finished we can start creating our Fortran projects and executing programs.

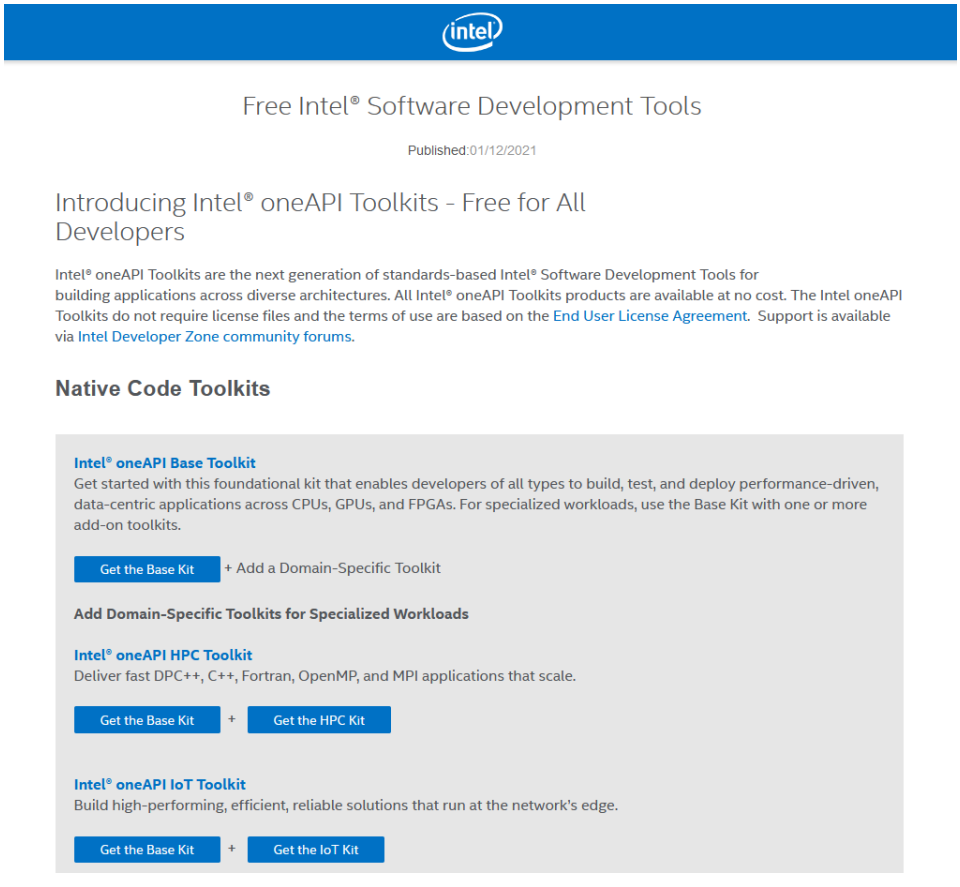

Figure 5.1: This website shows different toolkits offered by Intel. We are interested in the Base toolkit (always needed) and HPC toolkit (High-Performance Computing).

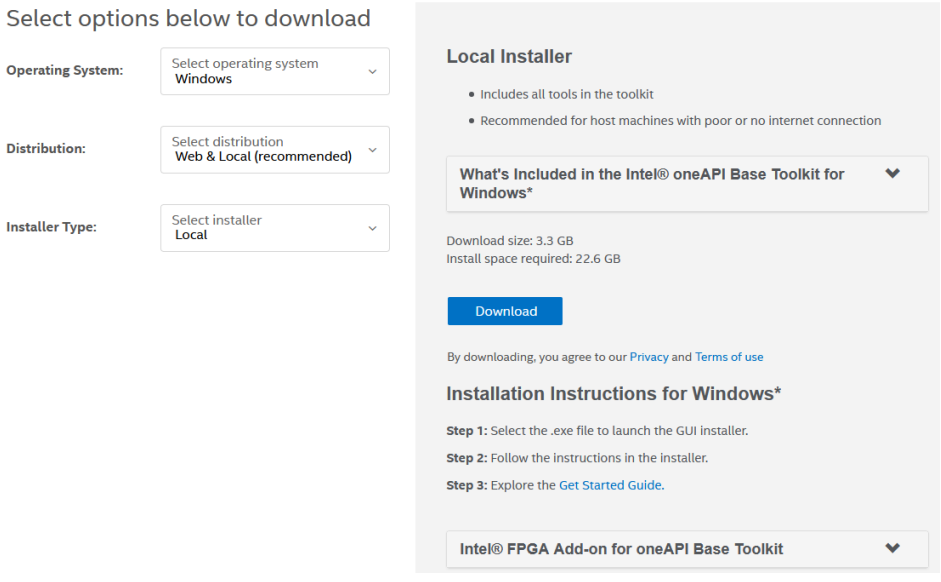

Figure 5.2: Second step of the download, select these options and click on *Download*.

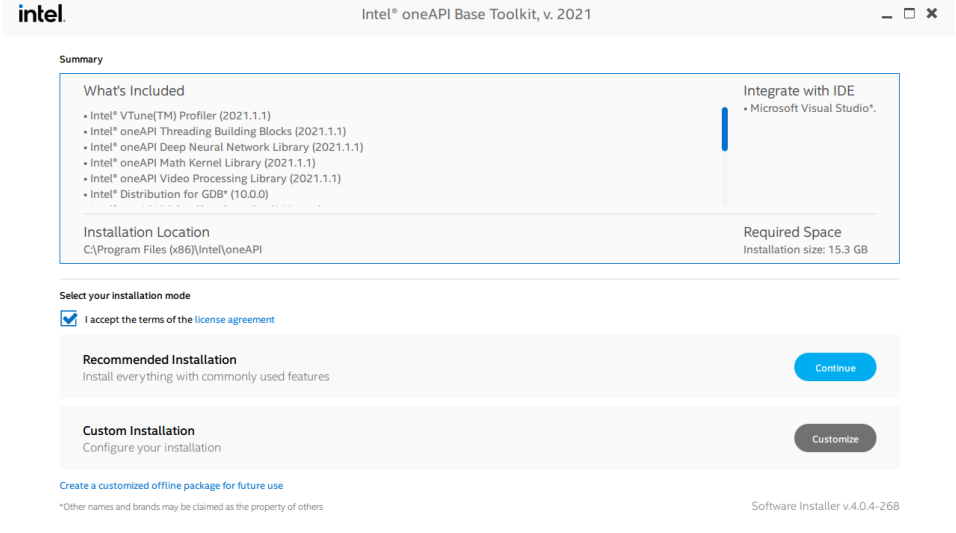

Figure 5.3: First window that appears in the installation, follow the *recommended installation* after accepting the *terms of the license agreement*.

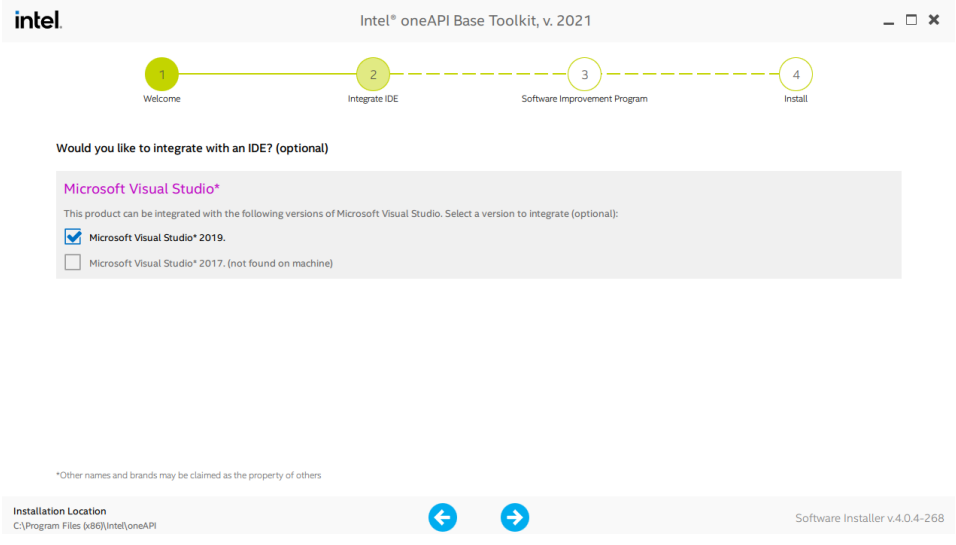

Figure 5.4: Second step of the installation. Make sure the integration with your current version of Visual Studio is marked

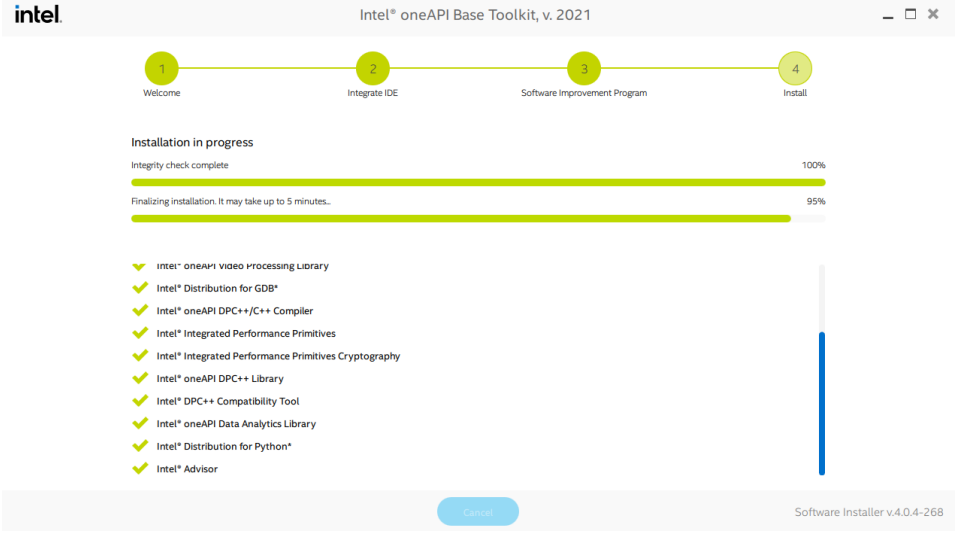

Figure 5.5: Fourth step of the installation, wait until all the tools are installed.

5.2 Create a Fortran project

To create a Fortran project, proceed with the following steps:

1. Open your version of Visual Studio.
2. Click on *File/New/Project...* (Figure 5.6).
3. In the *Intel(R) Visual Fortran* menu select *Console Application* and then click on *Main Program Code*.
4. Choose a name for the project (Figure 5.7).
5. Change the *Location* of the solution, a directory will be created there.
6. Choose the name of the solution, it does not have to be the same as the name of the project.
7. Select the option *Create directory for solution*.
8. Click on *OK*.

5.3 Compile, link and execute the "Hello world" example

In order to check that everything is correctly installed, we will run the easiest example. This is usually called the "Hello world" example. This example is written automatically when we select *Main Program Code*. This program opens the window console, writes the message "Hello world" and after pressing the enter key, it closes the window console.

If we want to execute this Fortran program, a translation to machine code must be done. This compiling and linking process is done automatically by Visual Studio by clicking the right button and is accomplished by the two following steps. Assuming the file is called "p1" for example, first the source file `p1.f90` is translated to an object code `p1.o` and then it is linked to other components to create an executable file: `p1.exe`.

To compile, link and execute the program follow the next steps:

1. Click on *BUILD/Build solution* or click on the corresponding icon. The program is compiled and linked.
2. Click on *DEBUG/Start Without Debugging* or click the corresponding icon. The program is executed.

Important Notice

More project/solution examples with Fortran language can be downloaded from:
https://github.com/jahrWork/Visual-Studio-projects.

5.4 Include new projects and new files

Before learning how to include files in our project (whether new or existing ones), it is interesting to understand the difference between the logical order that is generated in our solution and the real order of the files stored in the hard drive. This concept has been introduced at the beginning of this guide and it is specially important for the case of Fortran. When our project grows, we include files and folders in the solution explorer in order to have all the source codes, images, libraries, etc. organized. However, each file can be stored anywhere in our hard drive, maybe in the folder of a different project/solution, in the desktop, in the cloud, etc.

Think for example in a library that we use for different projects, it can be stored in a generic folder that is not related to the projects at all. It is responsibility of the programmer to have also the files organized in the computer in order to avoid problems when linking the project (e.g avoiding changes of files location). As an example see the Figure 2.17, we see in the Solution explorer a large number of folders but if we open the project we find that all the source codes are stored in the same folder (including also the libraries and modules of dislin).

On one hand, in the case of Fortran, in the solution explorer we find three folders by default: *Source*, *Resource* and *Header Files*. We typically use the Source folder for all the source codes written for the project while the other two are empty. Create the most appropriate structure inside *Source*

folder (by creating new folders for example) and organize the project. The libraries and modules can also be included there.

On the other hand, in the hard drive the structure is different, choose the most comfortable for your purposes. Apart from the `.sln` and the `.suo` files in the solution folder and the `.vfproj` file in the project folder (all essential for the project), we find the folders associated to the results of the build and compilation processes. If we build the project in the Release configuration, a `Release` folder appears with the `.obj` and `.mod` files, the executable, etc. If it is built in Debug mode, the same happens with a `Debug` folder. We suggest to create a `sources` folder inside the project path and store there all the `.f90` real files with the source code.

We can check in the solution location that after creating the solution and project, a `.sln` file, a folder with the name of the project and a `.vfproj` file inside have appeared. If the Solution is already created and we just want to open it we can double click on the `.sln` file or click on *File/Open/Project/Solution...* in the IDE and look for our `.sln` file.

Include another project in that solution:

1. Click on *File/New/Project....*
2. In the *Intel(R) Visual Fortran* menu select *Console Application* and then click on *Empty Project*.
3. Write the name of the new project.
4. In *Solution* select the option *Add to solution*.
5. Click on *OK*.
6. Before closing Visual Studio do not forget to click on *Save All*.

Including files in a project is easy:

1. Right click on the name of the project (in the solution explorer).
2. Click on *Add*.
3. Click on *New Item...* if you are going to start from scratch (or click in *Existing Item...* if you add it from an existing one).
4. Click on *Fortran Free-form File (.f90)*.
5. Write the name of the file.
6. Click on *Add*.

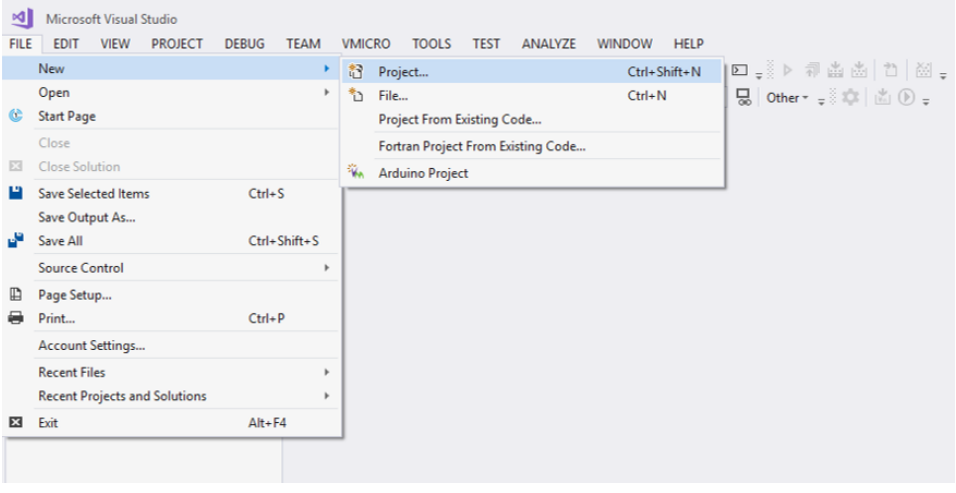

Figure 5.6: First step in order to create a solution with a Fortran project.

Important Notice

- By default, the created file appears in the source folder but we can grab and drop it in the root location of the project so it appears in the same level as Source, Resource and Header folders.
- For those files that we include in our project from a different location (not the folder where we store all the project), we have to bear in mind that Visual Studio will **look for it in the original location the next time** and thus we cannot change location. A better option would be saving a copy in our project folder and include it from there.

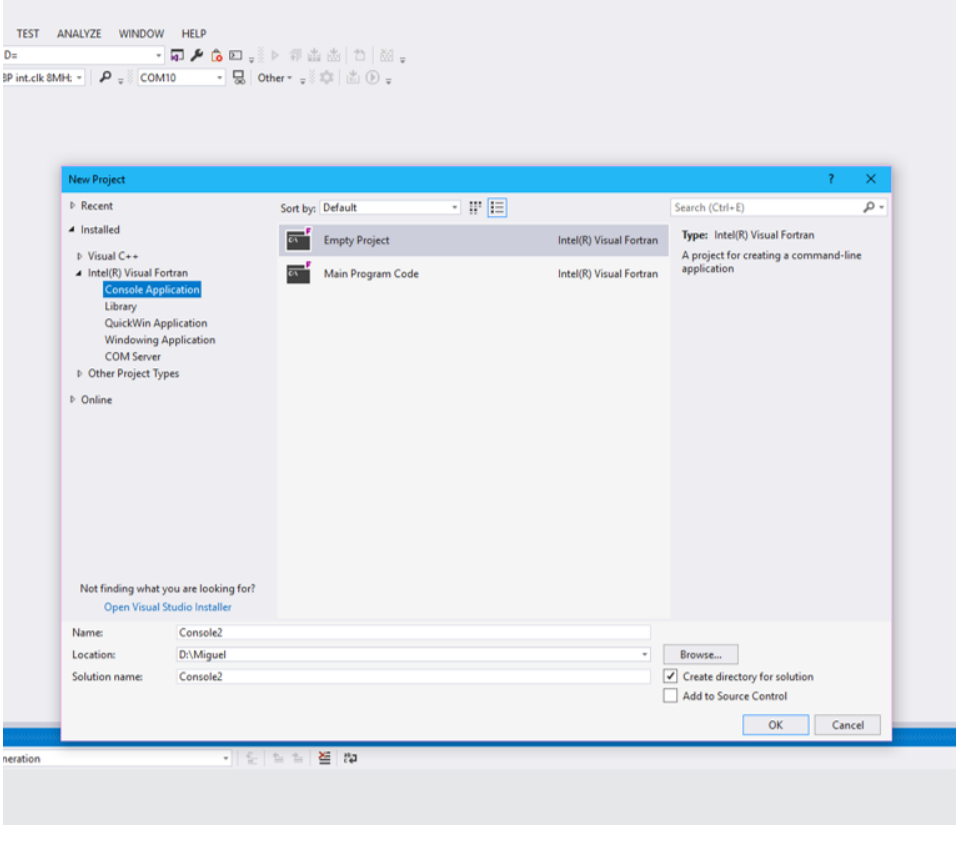

Figure 5.7: Second step in order to create a solution with a Fortran project.

5.5 Configuring compilation options

In the section 2.2 we have seen how to configure general VS settings, however, notice that those options are not related to the **Fortran project configuration**. For each project inside our solution (or any project we create from now on) we have to define some compilation properties. Actually, it is possible to define some properties for every **individual source code file** (by clinking right button on the name of the file and clicking on properties) but we are going to treat here some of the properties that involve **the whole project**.

With the project opened and selected in the solution explorer we have to click on *Project/Properties*, the changes we make now will only affect to

the chosen project. First of all check in this window that the *Configuration* selected is the Active one, which means, if you are running the application in Release mode, then change the properties of the Release mode (and the same if you are running the program in a different configuration). A change applied only to the Debug mode won't affect to the rest of configurations. The configuration (or mode) that you are modifying in the window opened can be checked in the top-left part of the window itself.

Important Notice

- If we have selected a specific file in the solution explorer (and not the whole project) the properties shown in the window opened are the **individual file properties**. Hence, take care of what is marked in the solution explorer.
- In addition, notice that a change in the properties of the project will change all internal files **except those properties of files that we have changed individually**.

  For example, if we mark ***8 (/real_size:64)*** in *Fortran/-Data/default Real KIND* in a `example.f90` file and we change the same option in the project configuration (let's say to ***16 (/real_size:128)***), all files are modified but `example.f90` does not. What's more, a red spot appears in the symbol of the file in the solution explorer notifying this condition. Furthermore, if the option does not have the compiler's default value, the value itself appears in bold in the configuration window.

Now let's take a look at some important compiler options that can be found in the window *Project/Properties*:

1. Default Real KIND
2. Stack
3. Heap arrays
4. Automatic Reallocation
5. Traceback Information
6. Runtime Error Checking
7. Treat Warnings As Errors
8. Warn For Non-standard Fortran
9. Compile Time Diagnostics

1. In *Fortran/Data/**Default Real KIND*** we can change the default value to "8 (/real_size:64)". Then, when we write in our code

   ```
   real :: x
   ```

   the default kind of the x will be 8 bytes (double precision) and we do not have to specify

   ```
   real(kind=8) :: x
   ```

   every time we declare a variable in double precision. When we use this trick, we get used to write only `real :: x` in our programs and not mix different precisions in the same code.

   Using this compilation option, when we want to run our program with simple precision, we just have to change the value and the whole program will be executed with simple precision. The following example shows both ways of executing the same program and demonstrates that the results can be different depending on the compiler options.

   Execute the following code changing the *Default Real KIND* value in the compiler options, first with the value by default (simple precision) and then forcing the code to be executed in double precision.

```
program ExampleManual1


    implicit none

    real(kind=4) :: x
    real :: y


    write(*,*) 'Declaration of x with - real(4):: x'
    write(*,*) 'Maximum value', huge(x)
    write(*,*) 'Minimum value', tiny(x)
    write(*,*) 'Round_off', epsilon(x)
    write(*,*) 'Significant digits', precision(x)

    write(*,*) 'Declaration of y with - real :: y'
    write(*,*) 'Maximum value', huge(y)
    write(*,*) 'Minimum value', tiny(y)
    write(*,*) 'Round_off', epsilon(y)
    write(*,*) 'Significant digits', precision(y)



end program ExampleManual1
```

In the first case you are going to obtain this:

```
Declaration of x with - real(4):: x
Maximum value  3.4028235E+38
Minimum value  1.1754944E-38
Round_off  1.1920929E-07
Significant digits           6
Declaration of y with - real :: y
Maximum value  3.4028235E+38
Minimum value  1.1754944E-38
Round_off  1.1920929E-07
Significant digits           6
```

While in the second one, the results are:

```
Declaration of x with - real(4):: x
Maximum value  3.4028235E+38
Minimum value  1.1754944E-38
Round_off  1.1920929E-07
Significant digits           6
Declaration of y with - real :: y
Maximum value  1.797693134862316E+308
Minimum value  2.225073858507201E-308
Round_off  2.220446049250313E-016
Significant digits          15
```

Important Notice

- Notice how in both cases the `x` is a simple precision real number since we are forcing the program to declare this variable as `kind=4`. Then, the upper limit for this variable is around `E+38` as the `huge(x)` function returns, the lower limit is around `E-38` according to `tiny(x)` function and we can trust the first 6 significant digits of the number (`epsilon(x)` calculates this value).
- However, the behaviour of the `y` is really different. In the declaration of the variable we have not specified the kind of the real variable. As said before, in the first execution (*Default Real KIND* 4), the compiler treats `y` as simple precision and in the second execution (*Default Real Kind 8*) the variable is treated as double precision. Hence, the limits for `y` in the second case are those related to double precision (`E+308`, `E+308` and 15 significant digits). In conclusion, the same code can return different results depending on the compilation options.

2. **Stack Overflow** can occur in our software for many reasons. It happens when the stack memory overflows and occupies other memory regions, then the stack pointer exceeds the stack bound [7] (concepts like call stack can be found in [8]). Since the problem is to allocate more memory on the stack than the maximum that fits (too large local arrays or infinite recursion for example) one solution is to extend the size of the stack. We can change the configuration by clicking on *Linker/Command Line/Additional Options:* and writing "/STACK:100000000".

3. **Heap allocation** is another method to avoid stack overflow in our programs. The heap is one of the three memories used by a Fortran application, it is dynamically allocated, bigger than the Stack in size but also slower. However, if your program needs from large automatic arrays (those whose size is defined by routine arguments) or makes arithmetic with large arrays (temporary arrays are normally stored in the stack) it can be a good idea to automatically store those arrays in the Heap, instead of stack. In order to do that click on *Fortran/Optimization/Heap Arrays* and write a value in kilobytes for the minimum temporary/automatic array size to be allocated in heap.

If `0` is written, then all those arrays are affected. Notice that writing */heap-arrays0* in *Fortran/Command Line/Additional Options* is another way to do the same.

4. In *Fortran/Command Line/Additional Options:*, in order to enable **automatic reallocation**, write "/assume:realloc_lhs". This option decides if using current Fortran Standard rules or old Fortran 2003 rules in relation to the automatic reallocation, which is described as:

> "Tells the compiler that when the left-hand side of an assignment is an allocatable object, it should be reallocated to the shape of the right-hand side of the assignment before the assignment occurs. This is the current Fortran Standard definition. This feature may cause extra overhead at run time. The option standard-realloc-lhs has the same effect as assume realloc_lhs."
>
> (https://software.intel.com/en-us/node/678222)

5. In *Fortran/Run-time* the option ***Generate Traceback Information*** is by default deactivated. Set it as *Yes* so extra information will be placed in the object files and in the case of appearing a severe error in the run time, you will be capable of locate the source of error because source file, routine name and line number correlation will be displayed. This function is independent of the Debug option.

6. In the same section, *Fortran/Run-time*, fix the option ***Runtime Error Checking*** to *All* so the compiler checks at run time all the available conditions. These conditions are for example related to pointers (check if some allocatable objects are not allocated) or uninitialized variables. Some conditions in the code that could be ignored by the compiler, in this case are considered as Warnings and the compilation shows them. Then, it forces the programmer to make a clean code and take control of those inappropriate conditions.

7. In the section *Fortran/Diagnostics* activate ***Treat Warnings As Errors***. This option changes some Warnings to Errors so the compilation will fail until they are solved. This function helps to maintain a clean

code and avoids to accumulate bugs in a program. Even when those warnings could let the program be compiled, they can also be conceptual mistakes and they can become a problem if a lot of ignored warnings are accumulated in the projects.

8. In the same section, *Fortran/Diagnostics*, the option ***Warn For Non-standard Fortran*** makes the compiler to return warnings when there are language elements that are not contained in the Fortran standard. These standard warnings do not affect compilation but they help the programmer to modernize the programming style, ensure that other programmers understand the code, comply with common rules and avoid problems with different compilers. In our case, we want to be adapted to the Fortran 2008 standard so fix that value in the compilation option.

9. Set the option *Fortran/Diagnostics/**Compile Time Diagnostics*** to *Show All* and the compiler will warn about a pile of interesting conditions to consider. For example, it will issue information about: variables declared in the code but not used after that, source code lines that exceed the maximum column width or statement functions that are never called in the program.

5.6 Configuring a graphic library: DISLIN

DISLIN is a plotting library for Fortran and C languages created by the Max Planck Institute (MPS). It is a high-level plotting library for displaying data and allows a quick plot of results when we are making a lot of tests with the code or debugging (graphic debugging) our program. It can be called from the main program or subroutines and "contains routines and functions for displaying data as curves, bar graphs, pie charts, 3D-colour plots, surfaces, contours and maps". In order to use DISLIN libraries in our project, we have to add some files first. Follow these steps:

1. Download the DISLIN distribution package required for our machine. Open the web:

   https://www.dislin.de/win64.html

   and choose Intel Fortran compiler package.

2. Unzip the downloaded file and look for:

   `disifl_d.lib` and `dislin_d.f90`

   It is assumed that our calculations are done in double precision and that is why double precision files (`*_d.lib, dislin_d.f90`) are selected.

3. Create a folder named DISLIN and locate those files on it.

4. Include this folder and all the items in our Visual Studio project following the process mentioned in section 5.4.

5. To avoid errors configure the project to: *Platform: x64* and *Configuration: All Configurations* by clicking with right button in your solution (in the solution explorer) and opening:

   Properties/Configuration Properties.

6. Also, in the project properties:

   Properties/Fortran/Data/Default real kind to 8.

7. *Properties/Fortran/Libraries/* : Multithreaded.

8. *Properties/Linker/Input/Additional Dependencies* : user32.lib, gdi32.lib.

9. Revise *IgnoreDefaultLibraryNames* : empty, none.

Now our dislin example should work, take note of the subroutines used in the plotting example below in order to use it in future codes. The example allows to plot a sine graph in a very simple way. It is advisable to write a `read(*,*)` line at the end of the program to avoid that the command line gets closed after finishing.

```
program Graph

       use dislin
       implicit none

    call sine_graph

contains

 subroutine sine_graph

  integer, parameter :: N = 100
  real :: x(0:N), y(0:N)
  integer :: i;
  real :: PI = 4 * atan(1.);
  real :: a, b;

   a = 0
   b = 2 * PI

   x = [ (a + (b-a) * i / N, i=0, N) ]
   y = sin(x)

   call scrmod("reverse")
   call metafl('XWIN')
   call qplot( x, y, N+1)

 end subroutine

end program
```

Important Notice

In case that you comment the `use dislin` statement that enables the use of DISLIN libraries (because you are not going to use it for example) do not forget to reverse last two steps of the DISLIN configuration. A list of errors could appear while building the code if you do not change the configuration.

5.7 Fortran FAQ

In this section some general concepts and good practices are going to be explained through Frequently Asked Questions.

1. What are Release and Debug execution modes?
2. What are Compile, Build and Start without debugging?
3. What are Static and Dynamic libraries? How can I create a `.lib` file?
4. What should I know about files and formats in Fortran language?

1. **What are Release and Debug execution modes?**

   Release and Debug are two possible modes for executing our program, each of them has its own default configuration that allow us to build and run the code in a different way. We can create more modes with different configurations for the solution or for the project. For example, we could create one mode where the option *Default Real KIND* of Fortran compiler is 4 and other where it is 8. In the first one a variable declared as a real number will be considered of simple precision by default, in the second case it will be treated as a double precision real number. By changing between both modes, we would run the code with the two configurations. In order to define these modes we can access the Configuration Manager by clicking on *Build/Configuration Manager* or by deploying the selector of Configuration (where the mode *Release* or *Debug* are shown). There we can create new modes or edit those we have. Once a mode is selected, the configuration can be modified in the properties page of the project (click with the right button in the name of the project and click on *Properties*).

   Regarding the Debug mode, it will allow you to run the code without turning on the optimiser. A lot of information will be included in the build files so we can check our program step by step, it can be useful for example for fixing bugs. However, if we are developing Numerical Simulations and related programs, we will use another kind of debugging: graphic assisted debugging. Checking errors in the code starts with the printing of results and the validation of the program module by module. That is why we typically include Dislin libraries in our program, to check results quickly and decide if we have programmed correctly our simulation, later we will save the numerical results in order to plot them with another tool.

2. **What are Compile, Build and Start without debugging?**

   We have seen what Debug and Release modes are and we now understand why one of the first IDE configurations to make is changing the command that executes the program, from the *Start with Debugging* to this one (see section 2.4). This command goes directly to the execution of the code using the Release mode (or the selected one, but not debugging). However, we are going to take a closer look at the different options mentioned.

   Compile: Once our project is created inside a solution and the first codes are written, we want to compile the program file by file in order to check that everything is correct. In this step the compiler comes into play, it checks and warns us about all the errors that exists in the program. In order to compile your piece of code open its file and click on *BUILD/Compile* in the IDE. You can also press `Ctrl+F7` or click in the following symbol (that should be in your toolbars[1]). Notice that you are compiling only the file opened and not the whole project.

   Build: With this command Visual Studio will compile and link the source files of your program. You can build only the project that you have opened or you have selected in the Solution Explorer with the option *BUILD/Build "Name of the project"* or clicking in the symbol

   You can also build the whole solution, with all the projects involved, by clicking in *BUILD/Build Solution* or clicking in the symbol

   In this case it does not matter which project you have selected since the whole solution is going to be compiled and linked.

[1] If you can not see the symbol go to the section 2.4 and modify the aspect of your IDE to show all the interesting shortcuts.

Important Notice

- **Build** command is going to perform an incremental build. This means that if Visual does not think it needs to build anything in the project (because the files did not changed since the last build), then it won't build anything. It is going to compile only the files that have changed and hence, the process is quicker. This is the most common used command to build our program. If here is no changes in the files, and you try to build everything two times, a message will appear saying that all the code is up-to-date.
- If **Clean** command is used, whether *BUILD/Clean "Name of the project"* or *BUILD/Clean Solution*, Visual Studio is going to delete all the compiled and intermediate files that exist (only for one project or the whole solution respectively). Then, if you want to build the program again, we have to wait for all the files and codes to be compiled and linked. We can use this if you are interested in starting a new compilation of all the source files. When you clean all the solution, the projects involved are cleaned one by one.
- **Rebuild** option, whether it is for only one project or the whole solution, will clean and then build the project/solution from scratch. The difference between using this and clicking first in Clean and later in Build is that Rebuild (in the case of a solution) will clean and build one project, later clean and build another one, etc. instead of cleaning all of them and then building all the solution.

Start Debugging/Start without Debugging: When you want to execute your program the command *Start without Debugging* is used. It **saves** the modified files, **compiles**, **links** and **executes** all the project selected in the Solution Explorer. In other words, it makes all the previous steps necessary to execute the program. Notice that it uses the build option (compile and link) so nothing will change if there is no changes in the files.

There are two options for this task; attach the debugger to the execution of the application or not. In the first case you will be able to pause the execution by breakpoints or use debugging tools such as

Watch Window or Diagnostic Tools. However, if you just want to run your application and see the results, it is not necessary to use this integrated debugger, there are different ways of performing a debugging of the code when needed.

If *Start without Debugging* option is used Visual Studio will launch your application without attaching the Debugger. Both ways of executing the code can be found in the menu *DEBUG* of the IDE and more specifically the option *without Debugging* can be found clicking on *DEBUG/Start without Debugging*, pressing `Ctrl+F5` or pressing the following icon of your toolbars[2].

3. **What are Static and Dynamic libraries? How can I create a `.lib` file?**

   Let's say we have some modules already validated and we want to create a library with them, the first thing to do is to decide the kind of library to use.

   On one side, in a **Static Library** the code written is compiled and located inside the executable file once it is created, we could go to another computer and run that executable file (the program) without problems. Then, we have a fast program since it has all the necessary code inside and does not have to look for part of the program "outside". However, it also has some disadvantages, for example programs become heavier and if we find a bug in the library, we will have to recompile the whole program.

   On the other side, **Dynamic libraries** are not stored in our executable file, they are added as an external file. It becomes a lighter program and fixing bugs is easier as soon as it is repaired for all programs once we change just one file. However, we have to drag all libraries when moving the executable to another computer and the execution will be slower because of the search that the program has to perform when it needs those codes.

   In conclusion, both have advantages and disadvantages and one or the other will be used depending on the situation. Each of them is

[2]See previous footnote

created, compiled and linked in a different way. Let's see how to create a static library and a full program that uses the library through an example:

(a) Create a new solution (or recycle one of the previous sections). Then, in this solution, create a new project as it is explained previously in this chapter. This project will contain the call to the subroutines stored in the library. Do not forget to choose a proper name for the project.

(b) Include an empty file called `ExampleManual2.f90` in the project and copy the following code:

```
program ExampleManual2

    use ModuleExample

    implicit none



    call  HelloWorld
    call  WritePI


    read(*,*)


end program
```

Notice that the program (called `ExampleManual2`) is going to `use` a module named `ModuleExample` and is going to `call` to the subroutines `HelloWorld` and `WritePI` contained in that module.

(c) Now create a new project in the same solution (it could be done in a different solution), this time with kind *Static Library*. In order to do that click on:

File/New/Project.../Installed/Intel(R) Visual Fortran

and then:

Library/Static Library

Take care of choosing a proper name for the library project (it will be the name of the `.lib` file), in this case let's use the same name as the module `ModuleExample.lib`.

(d) Add to this Library project an empty source file called `ModuleExample1.f90` and paste the next example code, this will be the module code stored in our library:

```
module ModuleExample


    implicit none

    contains


subroutine HelloWorld

    write(*,*) 'Hello World!'

end subroutine


subroutine WritePI

    real :: PI

    PI = 4 * atan(1.)


    write(*,'(f14.7)') PI

end subroutine



end module
```

(e) Then save all and set the library project as startup project (click with the right button on its name in the solution explorer and click on *Set as StartUp Project*). Compile the file `ModuleExample.f90` and build the whole library project in the Release mode. Check that `ModuleExample.lib` and `ModuleExample.mod` appear in the Release folder of the project.

(f) Add to the main project both library files with the Solution Explorer in the **Source folder** using *right click in the project name/Add/Existing Item.../* and looking for the files (see section 5.4).

There are **two options**: in the **first one** you can copy both files

in the folder of the main project and then include them using the solution explorer. In this case if you make a change and rebuild the library, you will have to copy again the files in the folder of the main project.

The **second option** could be including the files directly in the solution explorer and linking them to the original folder (the folder of the library). Just Add them in the same way, but this time look for the files in the folder of the library project, in Release. This last option lets you rebuild the library whenever you want and the main project will be accessing always the latest version of the library. However, you must remember that the location of the library cannot change since the main project could not find the files.

(g) Set the main project as startup project, build it and execute it without debugging.

Important Notice

- Remember that the name of the library project is shared with the name of the `.lib` file that you will have to include in the main project. At the same time the name of the `.mod` file is the same as the name of the module stored in the library.
- Remember that the source file of the library (`ModuleExample1.f90`) is not stored in the folder of the library by default. It is stored in the folder of the main project.

4. **What should I know about files and formats in Fortran language?**

We have read in the introduction of this manual about source and object codes and how the program we write needs from both. The first one is what we write with Fortran language and the second one is the translation that the compiler performs in order to make it understandable for computers. The **source code** written is stored in plain text files and in the case of Fortran, those files have the extension `.f90`, `.f` or `.for`.

In order to execute our program, we use the Intel® Fortran Compiler which is in charge of translating the code to machine language. It works developing two sub processes. The first process verifies that the source code is well written (fulfilling all the syntactic and semantic Fortran rules) and once finished, it creates an intermediate code called **object code** (with extension `.obj` in Windows OS). The second sub process consists in linking the object code with other codes stored in libraries. The extensions used for this process are `.dll` for shareable library files and `.mod` for module files. The module file is created if a source file being compiled defines a Fortran module, which means, it uses the MODULE statement. Finally, the compiler optimizes the code and converts it in an executable program (`.exe` in Windows).

The `.mod` files stores the interfaces of the modules that we have compiled, `example.mod` contains the necessary information about the modules that have been defined in the program (`example.f90`) and they are created with the `.obj` file also, when compiling the project. Actually, a `.mod` file is created for each module defined in our source (`.f90`) file and a `.obj` file will appear for the whole source code. The module interfaces share the name with the modules and the object file has the same name as the source file. We can define one module in each file and assign the same name to the module and the source file (it is not a requirement but it helps to organize everything). This can be broaden in [10], Lionel [4] and [5]. Regarding the history behind the file extensions of the source codes in Fortran, te information can be broaden in Conic-Jacob [1] or Lionel [3].

Regarding the Visual Studio files, the kind of files are shared with all the projects and solutions of Visual Studio. The first important kind of document is `.sln` which is the format where Visual stores our solution, the projects associated and some configuration. With this file, the Visual interface opens the projects associated. Together with the solution, it appears a configuration file with extension `.suo` that has been previously described in the introduction of this guide.

Apart from these files, we can find folders with the different projects associated inside our solution folder. The file with extension `.vfproj`, which makes reference to an Intel Fortran project file, stores everything needed to open the projects. More information about formats can be found in the official documentation of Intel Fortran Compiler [9], in the manual [2] or in [6]. Figure 5.8 summarises some of the extensions used.

Figure 5.8: List with common file extensions used in Intel Fortran projects.

File Extension	Type	Contents
.asm	Intermediate	Assembly file, passed to the assembler
.exe .dll .lib	Output	Executable, dynamic- link library, or library files
.fi .fd	Source	Header files
.for .f .fpp	Source	Fortran source files (fixed format)
.f90	Source	Fortran source file (free format)
.def	Source	Linker
.idl	Source	Microsoft IDL (non-Fortran)
.ilk	Intermediate	Incremental link file
.map	Output	Map file; output from the linker
.mod	Intermediate	Module file; created if a source file defines a Fortran module
.obj	Intermediate	Object file; passed to the linker
.pdb	Output (Debug)	Program debug database file
.tbl	Output (MIDL)	Type library; passed to Resource
.rc	Resource	Resource file (non-Fortran)
.res	Intermediate	Resource file; passed to the linker
.sln .suo	Solution	Visual Studio* solution file and solution options file
.vfproj .icproj .vcproj	Project	Intel® Fortran, Intel® C++, and Microsoft Visual C++* project files

Chapter 6

Web Projects

6.1 Installing tools

In order to obtain the essential tools for creating website projects with Visual Studio, we have to open the Visual Studio Installer as an administrator and click on *Modify*. We must select in the *Workloads* (in the tab reserved for Web and Cloud) the tools related to "ASP.NET and web" with the predefined optional installations selected (see Figure 6.1). After that, we click on *Modify* and wait for the process to finish the download and installation.

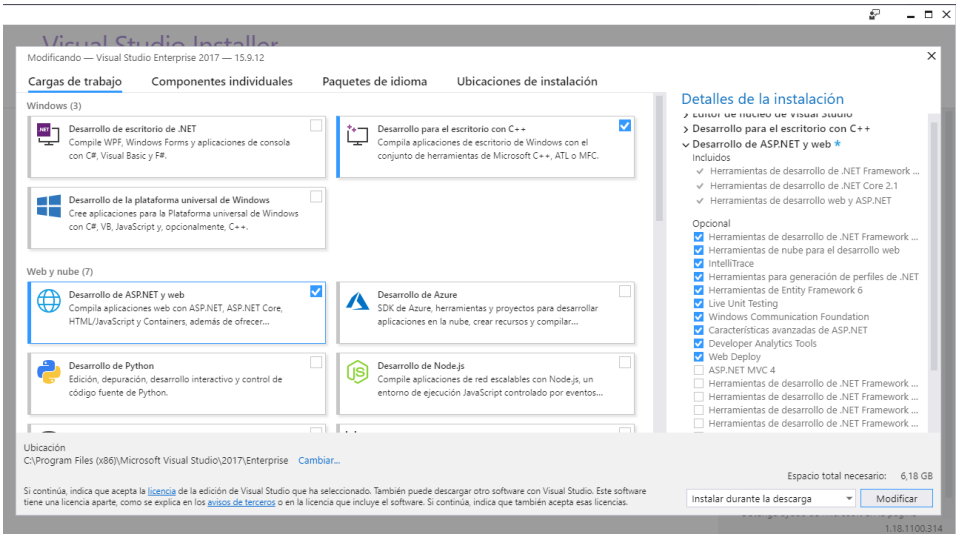

Figure 6.1: Main window of Visual Studio Installer with the Workload related to the web programming tools selected.

6.2 Create Website HTML/CSS/JS project

In this section the basic tools for creating web projects with Visual Studio are treated (HTML/CSS/JS). First of all, we have to open Visual Studio and create a solution with a similar template to the one we want, an "ASP.NET Empty Web Site".

1. Open your version of Visual Studio.
2. Go to *FILE/New/Project....*
3. In the tab *Installed/Visual C#/Web/Previous Versions* click on *ASP.NET Empty Web Site* (see Figure 6.2).
4. Choose a name for the project. For example choose "myWebSite".
5. Change the *Location* of the solution, a directory will be created there. In this case it is stored in the Desktop but choose the right location for yours.
6. Choose the name of the solution ("myWebSiteSolution" for example), it does not have to be the same as the name of the project.

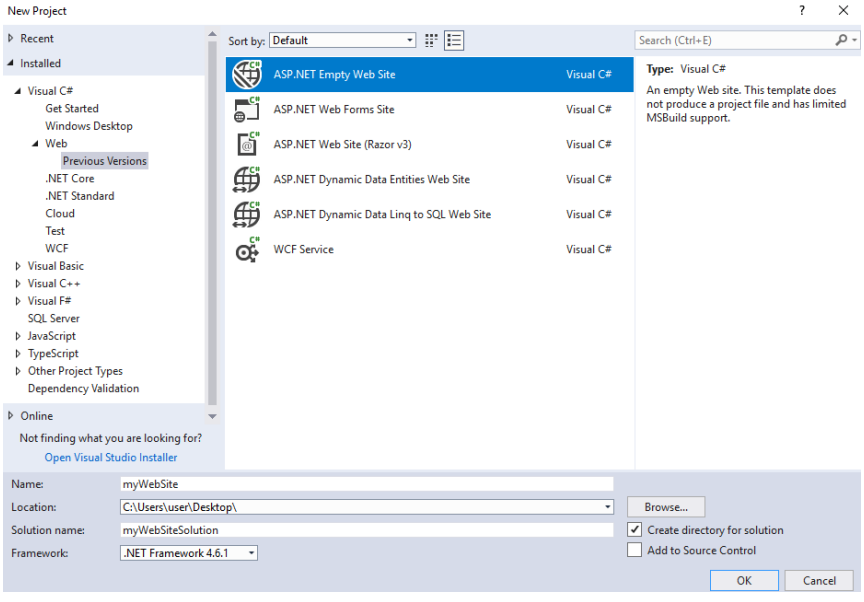

Figure 6.2: "New Project" window of Visual Studio with the project template selected, the name of both the project and the solution are already written.

7. Select the option *Create directory for solution.*

8. Click on *OK.*

The project is created with the default files and folders (see Figure 6.3) but we prefer to use the structure explained in this guide: `index.html`, CSS folder, JS folder and images folder if needed:

1. Firstly, click with the right button on the name of the project (in the Solution Explorer/myWebSite) and go to *Add/Add New Item....*

2. In the tab *Installed/Visual C#* click on *HTML Page* and call it "index.html" (see Figure 6.4).

3. Then, click on *Add* and a basic template will be created in your project (Figure 6.5).

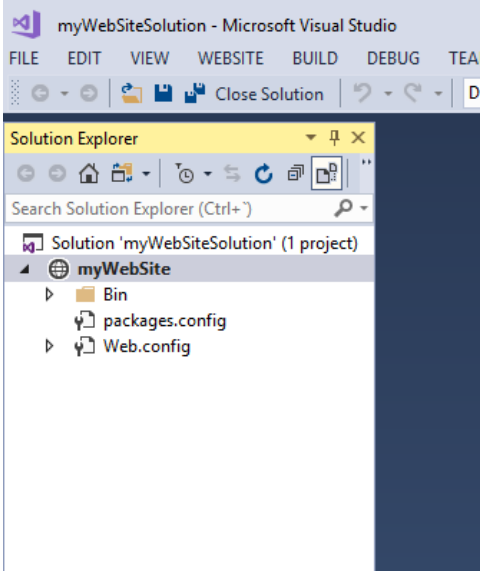

Figure 6.3: Initial aspect of the Solution and Project created for the Website.

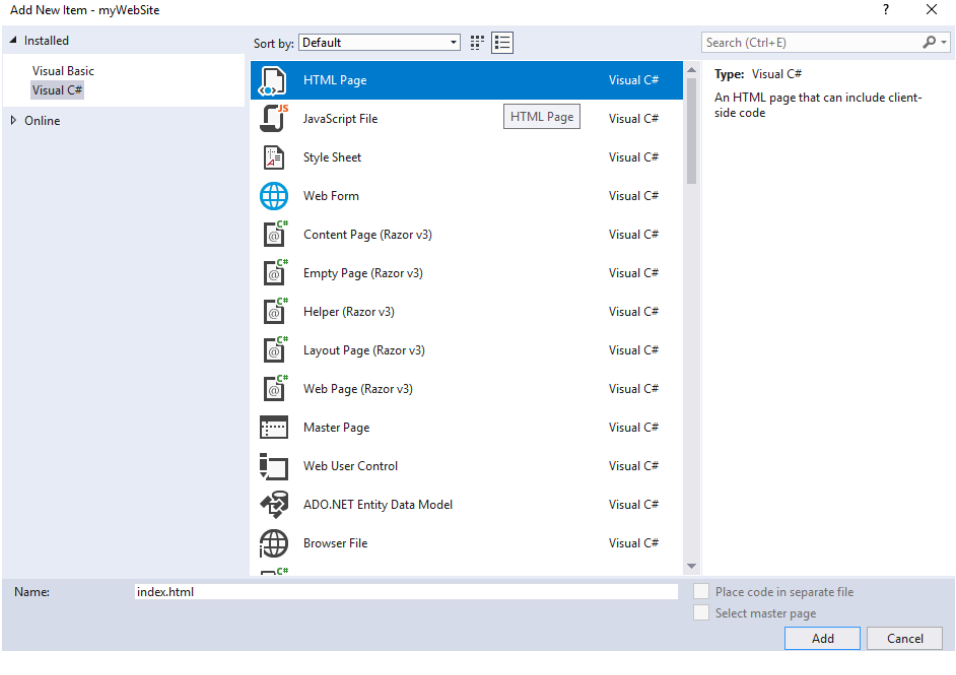

Figure 6.4: Adding `index.html` to our website using the *Add New Item* window of Visual Studio.

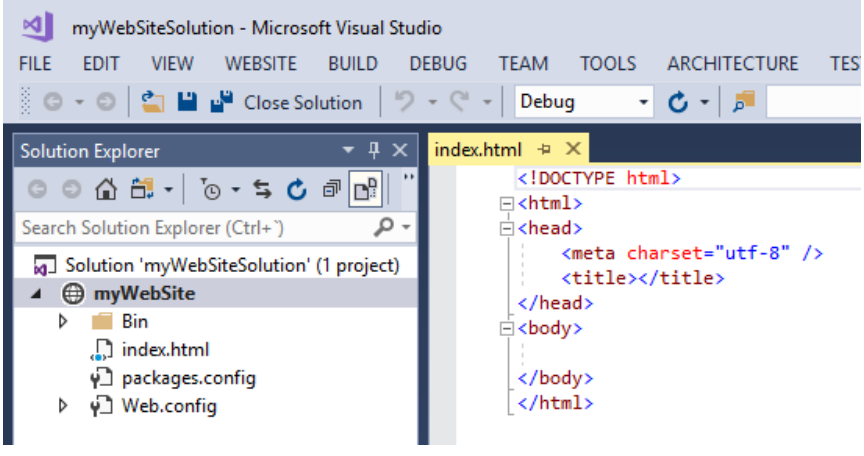

Figure 6.5: Template of the `index.html` file (created by default), here the website is going to be built.

Now we are going to create the desired folder structure, right click on the name of the project once again and go to *Add/New Folder*, give it the name "css". Repeat the process twice with the names "js" and "img", the final aspect of your project should be like the seen in Figure 6.6.

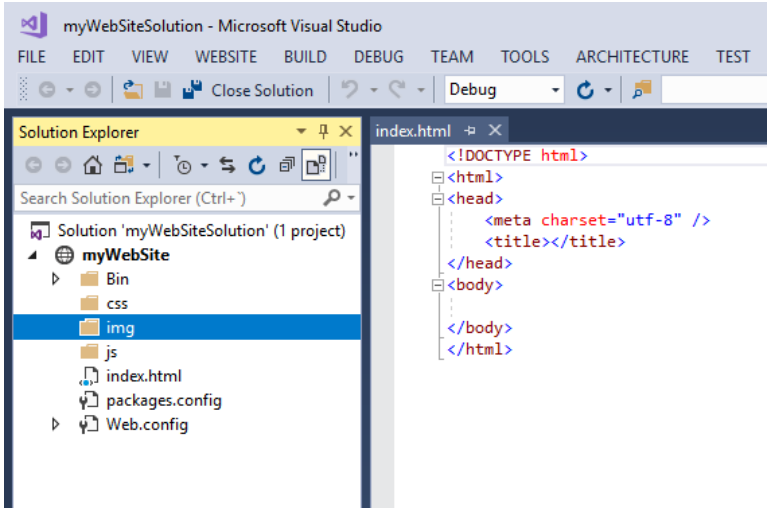

Figure 6.6: Final folder structure of the project, the files "css", "js" and "img" are not included by default.

6.3 Execute the "Hello world" example

Now we are going to "write" our website starting by a simple example. **In the first step**, we have to make our initial `index.html` template looks like the Figure 6.7. Do not forget to save the changes each time you make a modification in the webpage.

If we want to open our website we can go to the location of the project, open the folder "myWebSite" and double click on our `index.html` file (or just drag it into our browser: Chrome or Firefox for example). However, we want to use Visual Studio for this project so we are going to Build and Start our website using the IDE.

The Figure 6.8 shows the toolbar *Build* where the Build and Start commands are. We have to click first on *Build* and later in *Start* and Visual Studio will automatically show the output. In addition, a Chrome window (or the default browser) will be opened with our result. If we do not have the mentioned toolbar shown in the environment (highly recommended) we can also click on *BUILD/Build Web Site* and later on *DEBUG/Start without Debugging*. The result of our webpage is shown in the Figure 6.9.

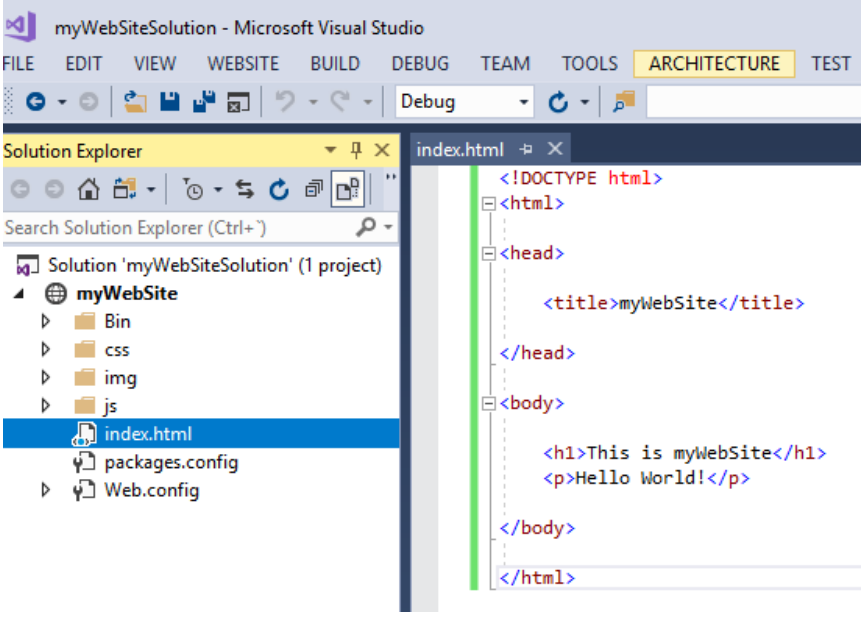

Figure 6.7: First example of website, the `index.html` file has been modified.

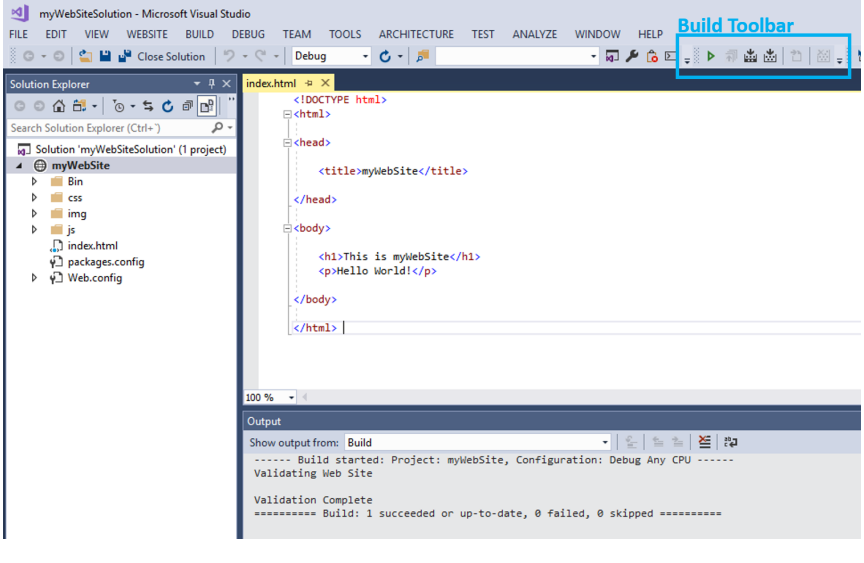

Figure 6.8: The window Output shows results of the execution. The *Build* toolbar appears remarked in the upper part of the IDE.

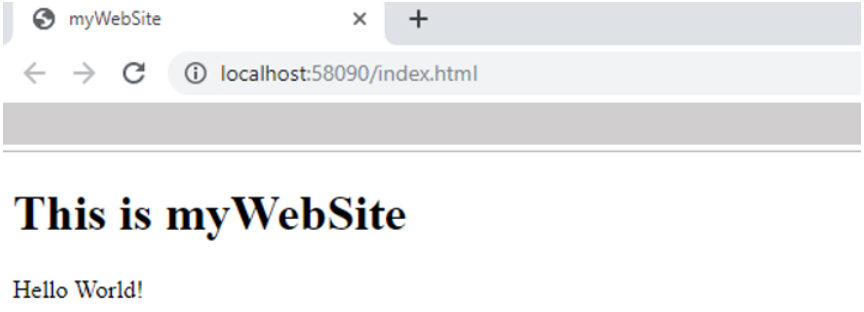

Figure 6.9: Result of "myWebSite" project presented in Google Chrome browser.

The next step is to include some style to the page. It can be done in the main HTML file (including <style>...<\style> tags and defining inside properties) or using external style sheets, which is preferable. In order to include the mandatory reference to the external file, it is used the <link> element in the head section.

First of all, include the file in the project by clicking with the right button in the `css` folder (in the Solution explorer) and going to *Add/Add New Item...*. There, in *Installed/Visual C#*, select *Style Sheet* and name it `Style.css`. Finally, add the new sheet to the project.

Now, include in the head of the html file the line seen in the Code 6.1. While the `rel` attribute specifies the relation between the main document and the linked one, the `type` command is used to specify the media type of the external file. The `href` command at the end of the line defines the location and name of the file.

```
<link rel=''stylesheet'' type=''text/css'' href=''css/Style.css''
```

Listing 6.1: Line of code included in the HTML file in order to link the css style sheets.

The code is now ready to accept some style in our dedicated file. In order to complete this basic example, we are going to change three things: the body background, the colour of the header in the text and the colour of the paragraphs. Open the sheet `Style.css` in Visual Studio and copy there the code seen in 6.2. As can be checked, there are different colours

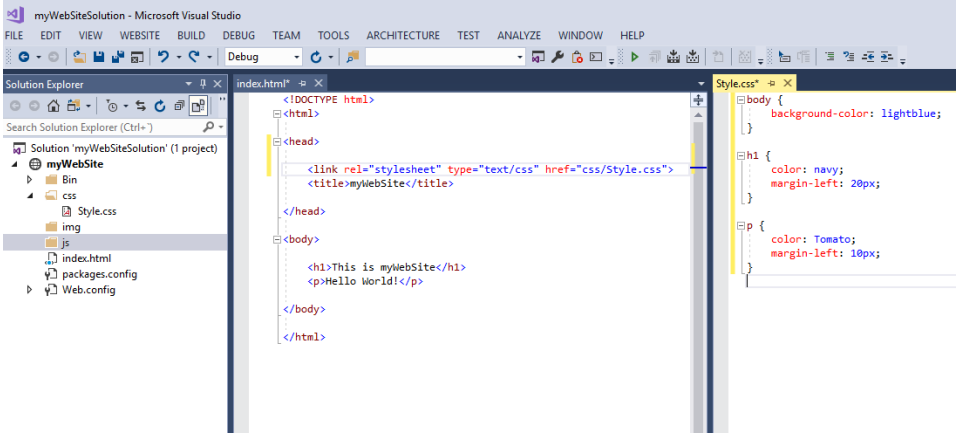

Figure 6.10: Result of myWebSite project, main files opened and structure of the project seen in the Solution Explorer.

to choose (you can even define them with the RGB code). In this example we are changing first the background colour, later the colour and alignment of the header (called h1, included in the HTML file) and finally the same parameters for the "Hello world!" paragraph. When finished, press on Build and later in Start, Visual will show the result. It should be like the Figure 6.10 and the final webpage like the Figure 6.11.

```
body {
background-color: lightblue;
}

h1 {
color: navy;
margin-left: 20px;
}

p {
color: Tomato;
margin-left: 10px;
}
```

Listing 6.2: Code to include in the `Style.css` created in the project in order to give style to the webpage.

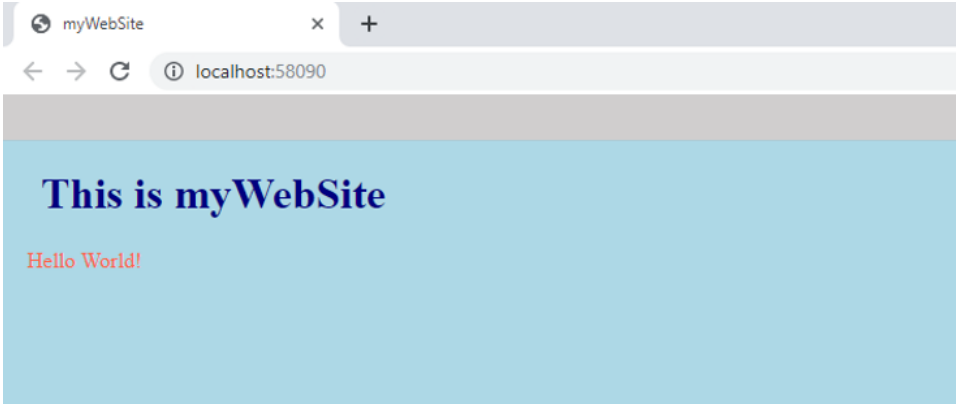

Figure 6.11: Result of myWebSite project after including style, using Google Chrome browser.

The third step of this example involves a similar process, now in order to include a JavaScript file in the project. Once again, the content is defined in a external file (it could also be written in the HTML with the tags <script>...<\script>). Hence, include the file by clicking with the right button in the `js` folder and going to *Add/Add New Item...*. There, in *Installed/Visual C#* select *JavaScript File* and name it `JavaScript.js`, adding finally the file with *Add* button.

Include at the end of the body of the html file the line seen in the Code 6.3. It makes use of the same tags but in this case the `src` attribute tells the browser where to find the script. External scripts are practical when the same code is used in many different web pages. Notice that the style is charged at the beginning of the html file in order to have the style orders defined before the content appears. For a similar reason, the modification of the contents (js file) is not uploaded until the content is already charged in the website, and that is why the code is located at the end of the body.

```
<script src="js/JavaScript.js"></script>
```

Listing 6.3: Line to include at the end of the HTML body in order to link the JavaScript file.

We have to write the JavaScript file and some elements to modify in the content of the webpage. In this example, we include a button with the text "I'm a button" written, and in the HTML file a function programmed for

that button. At the same time, we include an identifier for the element to be modified when clicking in the button, let's say for example the paragraph "Hello world!". This identifier (HWid) is used to specify (in the function) the element that has to change. Hence, the HTML file should be changed to be like the seen in the Code 6.4. Notice how the button has the attribute type and the function (called in this case "myFunction") has the order to make when clicking (*onclick*).

```
<!DOCTYPE html>
<html>

<head>

<link rel="stylesheet" type="text/css" href="css/Style.css"
    >
<title>myWebSite</title>

</head>

<body>

<h1>This is myWebSite</h1>
<p id="HWid">Hello World!</p>
<button type="button" onclick="myFunction()">I'm a button</
    button>

<script src="js/JavaScript.js"></script>

</body>

</html>
```

Listing 6.4: HTML file with a new button included, an identifier in the paragraph element and the link to the `.js` file.

Finally, open in Visual Studio the file `JavaScript.js` and copy there the code seen in 6.5 in order to define the function. When finished, press on *Build* and later in *Start* and Visual will show the result. It should be like the Figure 6.12 and the webpage, after pushing the button created, should be like the Figure 6.13.

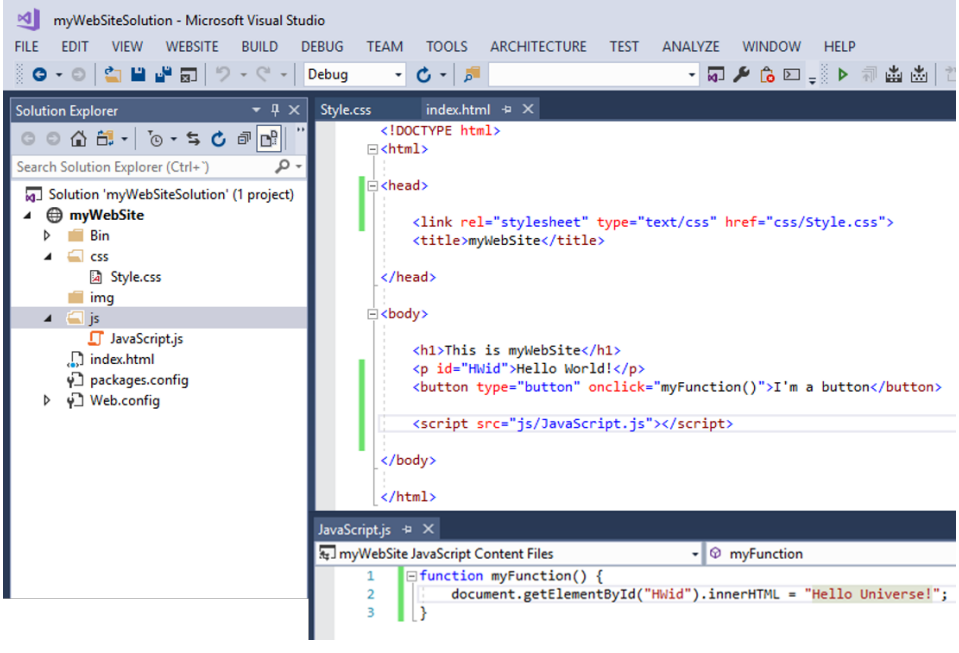

Figure 6.12: Result of myWebSite project, index and script opened and the structure of the project seen in the Solution Explorer.

```
function myFunction() {
document.getElementById("HWid").innerHTML="Hello Universe!"
}
```

Listing 6.5: *myFunction* code defined in the JavaScript file, it changes the text found with the identifier "HWid".

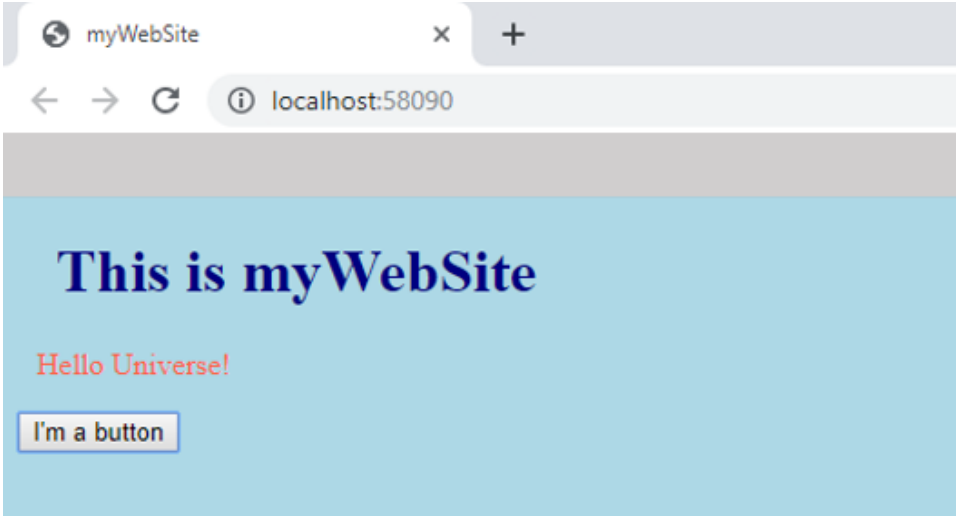

Figure 6.13: Result of myWebSite project after including style and a script, using Google Chrome browser.

6.4 Web Application: Fundamentals

A website usually consists of many files: text content, code, style sheets, media content, etc. When building a website, it is necessary to assemble these files into structure on your local computer, make sure they can talk to one another, and get all the content looking right before it is uploaded to a server. A common distribution of folders and files (extracted from https://developer.mozilla.org) in a web project involves:

1. **`index.html`:** This file generally contains your homepage content (the text and images that people see when they first go to the site). Using your text editor, you can create a new file called `index.html` and save it just inside your test-site folder.
2. **images folder:** This folder contains all the images that you use on your site.
3. **styles folder:** This folder contains the CSS code used to style the content (for example, setting text and background colours).
4. **scripts folder:** This folder contains all the JavaScript code used to add interactive functionality to the site (e.g. buttons that load data when clicked).

In conclusion, mainly three types of files contain the code necessary to create a web project: **HTML, CSS** and **JS**. The first one contains the static template of the webpage, the code that is used to structure the content and the webpage itself. Here it is decided if the webpage shows paragraphs, bullet points, images, tables, etc. and at the same time the content is linked.

HTML (Hypertext Markup Language) is a markup language, hence, the file is constructed using a series of elements (tags are used to create the elements) that define every instruction we want to apply to an specific piece of content. For example, the enclosing tags can make a word hyperlink to somewhere else, italicize words and more. There are tags for every function or behaviour we want to define.

CSS and JS files are linked to the main HTML file using a file path so one file (main HTML file) knows where another one is. For example, the header of the `index.html` contains different routes to the style files (CSS), in addition, the body of the main HTML file also contains routes to images, videos and rest of the material that the webpage has.

CSS (Cascading Style Sheets) is the code used to provide style to the content referred in the HTML file, this means that here we are going to decide whether the words are black or red, the decoration around the images or the aspect of the different elements mentioned in the HTML file. CSS is a *style sheet language*, we can select individually elements in the main file and change the style. In addition, we can apply the same configuration to a specific group of elements that share the same aspect.

JS makes reference to JavaScript, a programming language that adds interactivity to the website. With these files we can decide what the webpage shows when a button is pressed or define some dynamic styling. These scripts have to be applied to the main HTML file and the possibilities are huge.

Once the different files are written in our computer and the project has been finished, all the information is uploaded to the server, a hardware and software tool that is activated at all times (in the case of a public website for example) waiting for the clients. If this server is accessible through the internet, it will be waiting for a client all over the world. When a user writes the specific IP address in its browser (or the URL assigned to

our server and port), then the software part of the server will receive the request and answer. If the access is permitted, then the server will send over the internet the webpage to the computer of the client, where the browser will interpret all the information and show the website. This is a general description since there are many different ways of receiving the webpage or the web application depending on the amount of information that is processed in the client's computer or in the server. In addition, there are many protocols and applications involved in this operation.

6.4.1 HTML Structure

In this section a brief introduction to HTML is presented and some concepts related to the structure of an HTML file are treated. HTML is mainly the combination of Hypertext and Markup language. While Hypertext defines the link between the web pages, Markup language is used to define the text document and the tags that define the structure. The first version made of HTML was *HTML 1.0* but the first standard version was *HTML 2.0*, in 2014 HTML5 was released as a stable W3C (World Wide Web Consortium) Recommendation, meaning the specification process is complete.

HTML uses predefined tags and elements, both are read by the browser and define the properties of the content that is displayed, all the tags used have to be closed when the effect must finish. Otherwise, the property of the specific effect will continue until the end of the page. In a general case, the extent of an element is indicated by a pair of tags: the "start tag" ( ) and the "end tag" ( ), an example is shown in Code 6.6. The text content of the element is placed between these tags. The start tag may also include attributes or specific routes (in the case of including images for example) within the tag.

```
<p> This is a paragraph. </p>
```

Listing 6.6: Tags used in HTML with the predefined function of including paragrahps in the webpage.

One of the simplest examples of HTML can be seen in the Code 6.7. It contains elements (head, title or body in this case) which are used to build the blocks of webpages, there are a lot more elements and tags that can

be used for many different purposes. The first declaration (<!DOCTYPE html>) is used to specify the HTML version, in this case the last one, HTML5. The next tag (<html>) wraps all the content (is closed at the end) and is called HTML root element. Then, two main blocks are defined, the head and the body. While the head (<head>... </head>) contains all the HTML metadata (all the information that is not going to be visible i.e. title, page CSS, etc.), the body (<body>... </body>) includes all the data that the webpage has, from texts to links and everything is organized with the elements and tags mentioned (this information is visible).

```
<!DOCTYPE html>

<html>
<head>

<title>Page Title</title>

</head>
<body>

<h1>This is a Heading</h1>
<p>This is a paragraph.</p>
<p>This is another paragraph.</p>

</body>
</html>
```

Listing 6.7: Main structure of an HTML file.

We could write now the Code 6.7 in the notepad, save it with the name `index.html` and open the file with the browser of our computer. The browser would interpret it and show the result as a common internet webpage. The browser also takes into account the HTML version written at the start of the code since not all the browser support all the versions, it must be remarked that creating webpages involves that we always depend on the browser used by the user.

6.4.2 CSS Concepts

CSS is a simply designed language that tries to simplify the process of making webpages presentable. It allows applying styles to webpages and enables the possibility of doing the style page independent of the HTML. When tags like <font> and colour attributes were added to the HTML 3.2 specification, the development of large websites became complicated because fonts and colour information were added to every single page. The code became a long process. Then, W3C created CSS and the style formatting was removed from the HTML page.

The CSS syntax consist in rules that the browser interprets and then applies to the specific element in the document that is affected by the style change. In every rule-set there are a selector and a declaration block; the selector points to the specific element that is going to show the style and the declaration block contains the style. It uses the name of the property to be changed and the value, both separated by semicolons. Finally, a CSS declaration always ends with a semicolon and declaration blocks are surrounded by curly braces.

There are a lot of properties and values to apply to the design of the web, some examples are: colour, text-align, opacity, font size or font-weight. Depending on the element selected there are different options available to modify. An essential part of CSS code is the selector of the element to change, there are Selectors used to "find" (or select) HTML elements based on their element name, id, class, attribute, and more. The **Universal selector** simply matches the name of any element type so we will define the colour (for example) of every element in the document. On the contrary, we can choose that the property only affects to the elements with an specific **Name** like for example the paragraphs called ... in the example of the previous section. The **id selector** uses the id attribute of an HTML element (we can add it to the different elements used) to select a specific element. The id of the elements should be unique so we would be selecting one unique element. It uses a hash (#) character followed by the id in order to select the element. The **class selector** selects elements with a specific class attribute (we can also add the class to the elements of our webpage). In this case it uses a (.) character, followed by the name of the class. The Code 6.8 shows some examples of the notions explained.

```
p {
color: red;
text-align: center;
}

#para1 {
text-align: center;
color: red;
}

.center {
text-align: center;
color: red;
}
```

Listing 6.8: Code to include in the css file created in order to give style to the webpage.

6.4.3 JavaScript Concepts

In the beginning, JavaScript was only used in the client side (in the browser), but node js have made possible to run also JavaScript on the server side. Today, JavaScript is everywhere (Desktop/Server/Mobile) and, as it has been said, it is used to program the behaviour of the webpages. JavaScript can change content of the HTML, change attribute values, styles, show or hide elements and more.

In the HTML file, the code of JavaScript must be written between <script> and </script> tags and any number of scripts can be written in the whole document, in the body, the head, or in both. External scripts are really useful when the same code is used in different web pages and also when developing a multilayer project (all the documents are separated conceptually in different layers). In this case we reference the external file (called `ExternalScripts.js`) as it is shown in the Code 6.9 and the reference can be placed in the head or in the body. One of the advantages of this kind of structure is that the HTML and JavaScript files are easier to read and maintain.

```
<script src="ExternalScripts.js"> </script>
```

Listing 6.9: Example of the inclusion of JavaScript files in our main HTML.

Similar to other programming languages, a function in JavaScript is a block of code that is executed when it is called. It can be programmed that an *event* (like the click in a button) calls a function and makes some changes in the webpage. *Objects* in JavaScript are the most important data-type in this programming language since the whole language is formed by these blocks. The objects are similar to variables but can contain many values, each value goes together with a name (the pairs name and value are called *properties*). At the same time, the objects can have *methods* which are the actions that can perform. These methods are functions stored as properties. In the Code 6.10 (extracted from https://www.w3schools.com) an object called "person" is declared containing both properties and a method. Notice that the word "this" refers to the owner of the method, in this case the object "person".

```
var person = {
firstName: "John",
lastName : "Doe",
id       : 5566,
fullName : function() {
return this.firstName + " " + this.lastName;
}
};
```

Listing 6.10: Example of object called "person", declared containing both properties and a method.

6.5 Installing Node.js tools

In order to use Node.js with Visual Studio we need the specific workload for the IDE and the JavaScript runtime environment installed in the computer. To install the workload we have to open as usually the Visual Studio Installer (as an administrator) and click on *Modify*. We must select in the *Workloads* (in the tab reserved for Web and Cloud) the package *Node.js Development* with the predefined optional tools selected (see Figure 6.14). Then, we click on *Modify* and wait for the process to finish the download and installation.

Now, we are going to install the LTS version of the runtime (long-term support, stable release of software that is maintained by the owners for long periods of time).

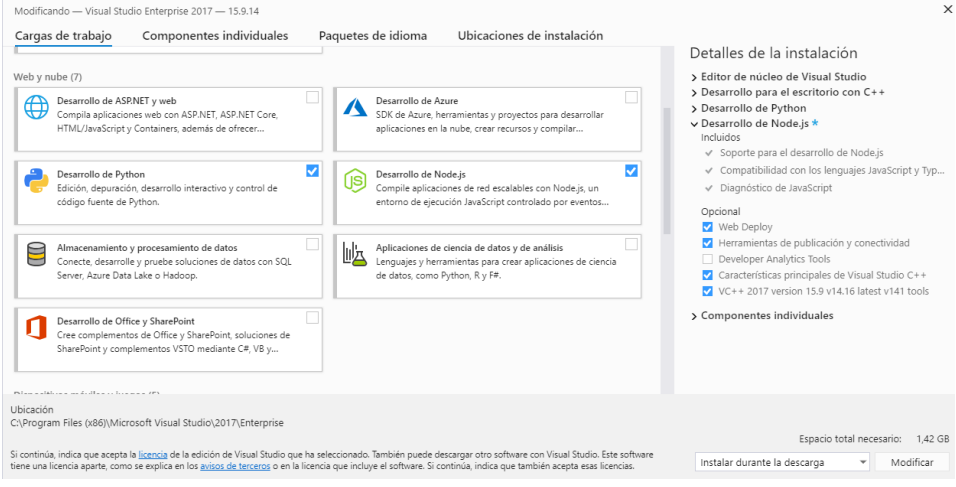

Figure 6.14: Main window of Visual Studio Installer with the Workload to install chosen.

1. Download the installer from the Node.js website: https://nodejs.org/en/download/.
2. Choose the *recommended version for most users* installer for your computer (typically Windows 64-bit) and download the installer.
3. Double click on the file that you have obtained.
4. Click on *Next*, accept conditions and click again on *Next*.
5. Maintain the same location for the installation and click twice in *Next*.
6. Finally, click on *Install* and accept the administrator warning (Visual Studio should automatically detect the installed runtime.).

6.6 Create a Node.js project

To create a Node.js project, proceed with the following steps:

1. Open your release of Visual Studio.
2. Click on *File/New/Project...*.

3. In the *TypeScript* menu of the *Installed* tab select *Node.js* and then click on *Blank Node.js Console Application*.
4. Choose a name for the project.
5. Change the *Location* of the solution, the directory will be created there.
6. Choose the name of the solution, it does not have to be the same as the one of the project.
7. Select option *Create directory for solution*.
8. Click on *OK*.

6.7 Execute the "Hello world" example (Node.js)

You can check in the solution directory chosen that a folder with the project has been created, inside, the project is represented by a `.njsproj` file. In the solution explorer you will find a npm node which shows the already installed npm packages (NPM is a package manager for Node.js with thousands of free packages (or modules) to download and use).

In order to check that everything is correctly installed we run the "Hello world" example. This example is written automatically when we select this kind of project so you can open the file `app.ts` (double-clicking on this name in the solution explorer) and check that is written something similar to the code 6.11.

```
console.log('Hello world');
```

Listing 6.11: "Hello world" example written in a Node.js project.

After building and starting the project this program opens the console and shows ”Hello World” written. Follow the next steps:

1. Click on *BUILD + “Name of the project”* or click on the corresponding icon.

2. Click on *DEBUG/Start Without Debugging* or click the corresponding icon.

Important Notice

If the console is immediately closed after printing “Hello world” in your computer, you have to force the process to wait after finishing the execution. Click on *DEBUG/Options...* and look for “NodeJS Tools”, there tick the box *Wait for input when process exits normally.*

6.8 Debugging a web page

Generally, the main file of a web page needs to be opened by a server. Node.js allows to run a local server very easily. Hence, in order to debug a web page before it is hosted in a special server, the following steps can be accomplished:

1. Install `node.js`.
2. Install a http server by the following command:
   `>npm install http-server -g`
3. Execute the commnad:
   `>http-server`
4. In chrome browser type:
   `//http://localhost:8080/index.html`
   and press F12.

The last command will start up a http server in the local-host.

6.9 Cordova project

Cordova allows us to create Phone Apps starting with our previously created Web project. In order to obtain the essential tools for creating Cordova projects with Visual Studio follow these steps:

1. Open Visual Studio installer as an administrator and click on *Modify*.
2. Select in the *Workloads* (in the tab reserved for smartphones and games) the tools related to *Development for Mobile Phones Devices with JavaScript* with the predefined optional installations selected (GIT for Windows is not needed) (see Figure 6.15).
3. In this case we also need from the *Individual Components* tab the *Google Android Emulator (API level 27)* and the *Installation of Android SDK (API level 27)* (Figures 6.16 and 6.17).
4. Click on *Modify* and wait for the process to finish downloading and installing.

Important Notice

This process has been tested with Visual Studio 2017, in the newer versions of the IDE we do not know if Microsoft will continue offering the Cordova tools.

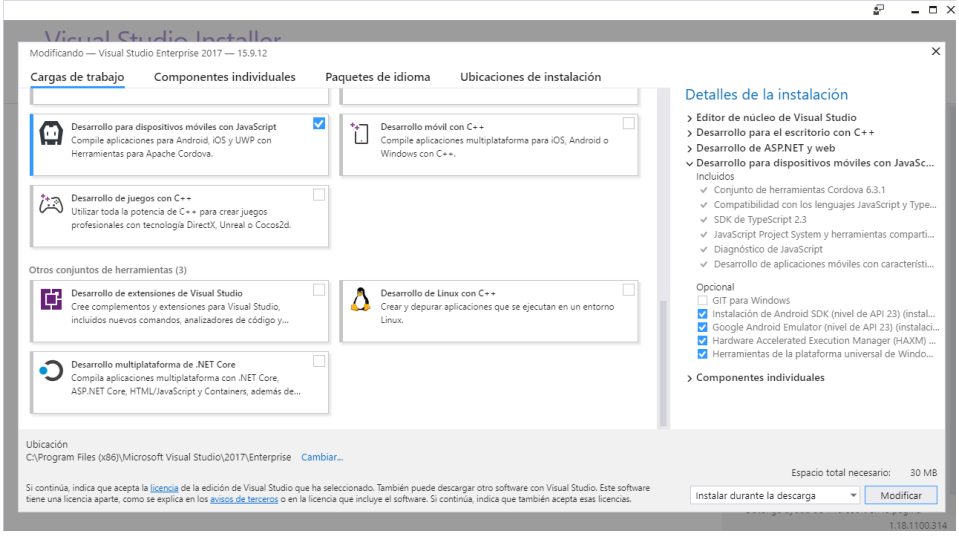

Figure 6.15: Main window of Visual Studio Installer with the Workload to install chosen, it installs Cordova programming tools.

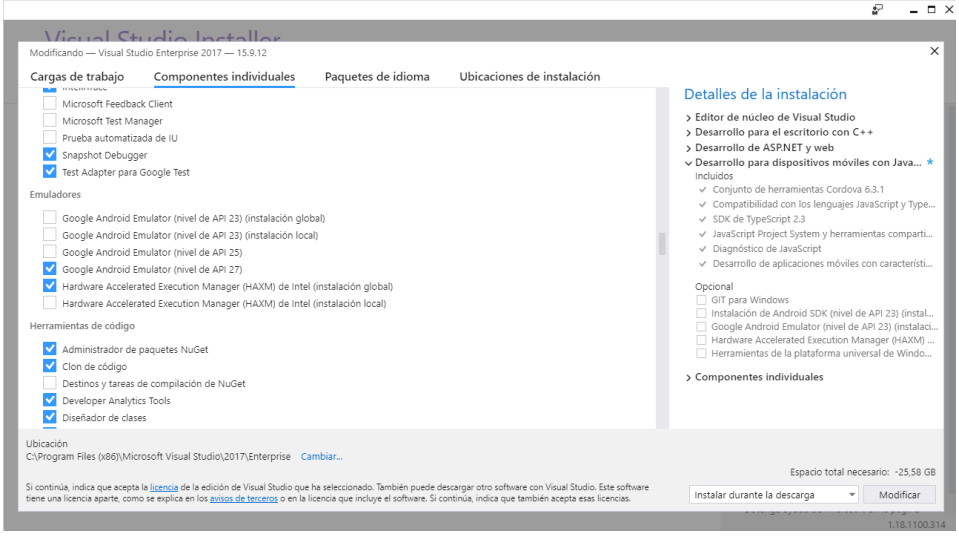

Figure 6.16: Main window of Visual Studio Installer with the Individual Components to install chosen. The right version for Android applications is selected.

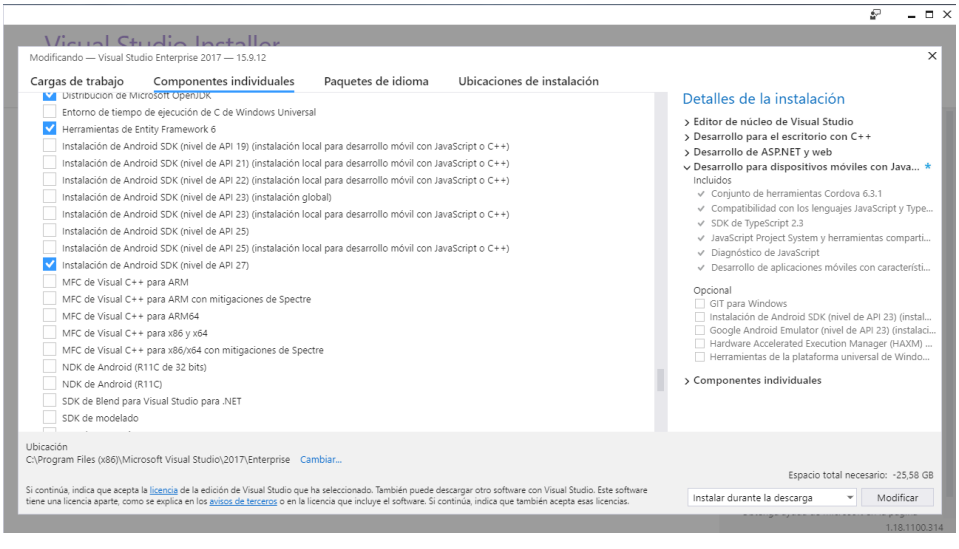

Figure 6.17: Main window of Visual Studio Installer with more Individual Components to install in order to obtain Cordova programming tools. The right version for Android applications is selected.

The next step is to create a project with Cordova, follow these steps:

1. Click on *File/New/Project...* or press Ctrl+Shift+N (Figure 6.18).
2. Choose *Blank App (Apache Cordova)* in the tab *Installed/JavaScript/Mobile Apps*. Name the project "HelloWorld" (this will be the name of the app), choose the location of the project and name the solution: "CordovaProjects" for example (Figure 6.19).
3. Click on *OK*.
4. The aspect by default of the solution and project is seen in Figure 6.20.

Now we can simulate and execute the HelloWorld example starting with the webpage created in the previous section:

1. Select all the files inside the `www` folder and delete them.

2. Click with the right button of your mouse on the folder `www` and click on *Add/Existing Item...*, navigate to the location of the files created in the section 6.3 (HTML/CSS/JS) and add to the project all the files and folders. The final aspect of the project opened is shown in the Figure 6.21.

3. Deploy the menu with the solution configurations (Debug is shown by default) and open the Configuration Manager.

4. In order to simulate the result of our App, deselect *Deploy* when the Configuration *Debug* and Platform *Android* are selected (Figure 6.22).

5. Close the Configuration Manager and check that the configuration of the solution is marked as **Debug - Android - Simulate in Browser LG G5**. Click on the execution button and the result will appear in your default browser (Chrome, Firefox, etc. Figure 6.23).

6. If the results of the simulation are correct, the .apk file can be generated in order to install the App in your Android phone.

7. In the same place where the simulation is chosen, now choose *Device* (Figure 6.24). Execute the project and a `.apk` file will appear in the folder of the project, in the path

   platforms\android\build\outputs\apk.

8. Transfer that file to your smartphone and execute it (Figure 6.25).

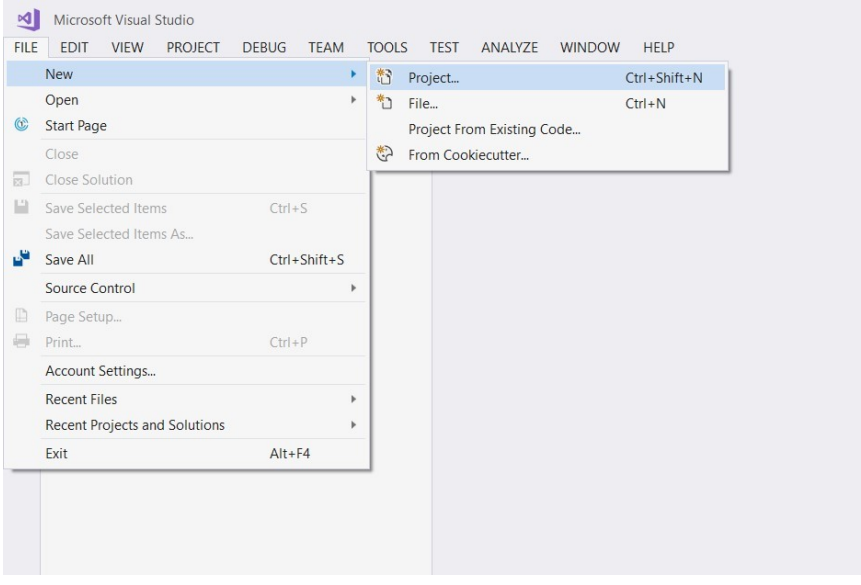

Figure 6.18: First step in order to crate a Cordova project.

Important Notice

When building and executing the project, the first time, an error like the next one could appear:

Error: Failed to run "javac -version", make sure that you have a JDK installed.
You can get it from: http://www.oracle.com/technetwork/java/-javase /downloads.

If it is your case, install or update the JDK (Java Development Kit) of Java. We have fixed this problem by installing not only the release 12.0.2 of the JDK but also installing the update 8u221 for the previous release of the JDK (Java SE 8).

In order to install those complements open the website and go through the page looking for the Download button of the Oracle JDK of the version *Java SE 12.0.2* and the JDK of *Java SE 8u221*. Click first on one button and download and install the file `jdk-12.0.2_windows-x64_bin.zip`. Later, click on the second one and download and install `jdk-8u221-windows-x64.exe`.

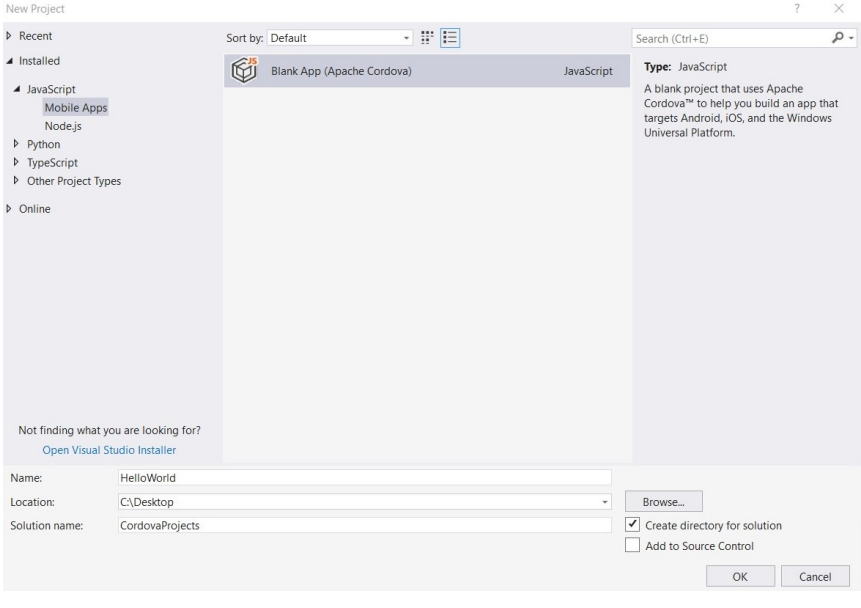

Figure 6.19: Empty Cordova project, do not forget to choose a Name for the project (name of the app), a Solution Name and the location to store the folders and files.

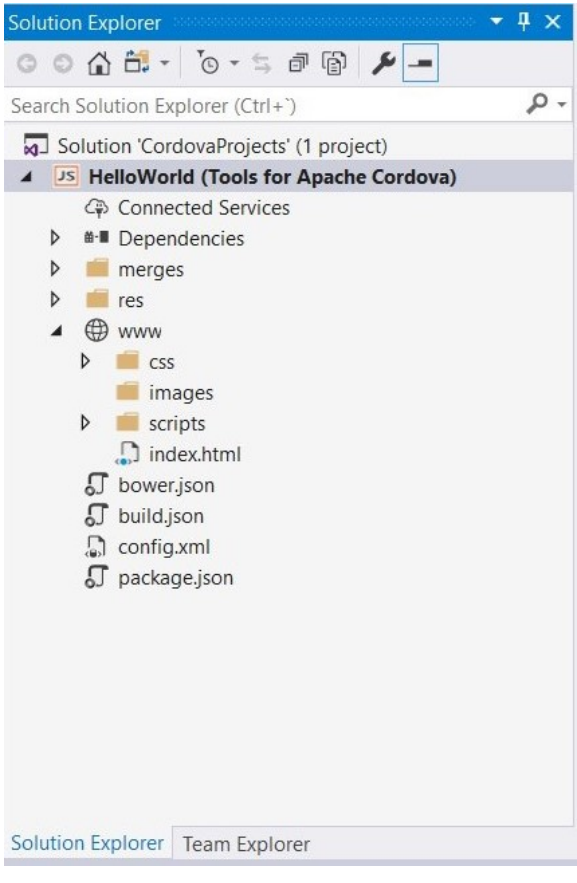

Figure 6.20: Initial aspect of the project, the folder `www` is where all the files of the HTML/CSS/JS project will be stored.

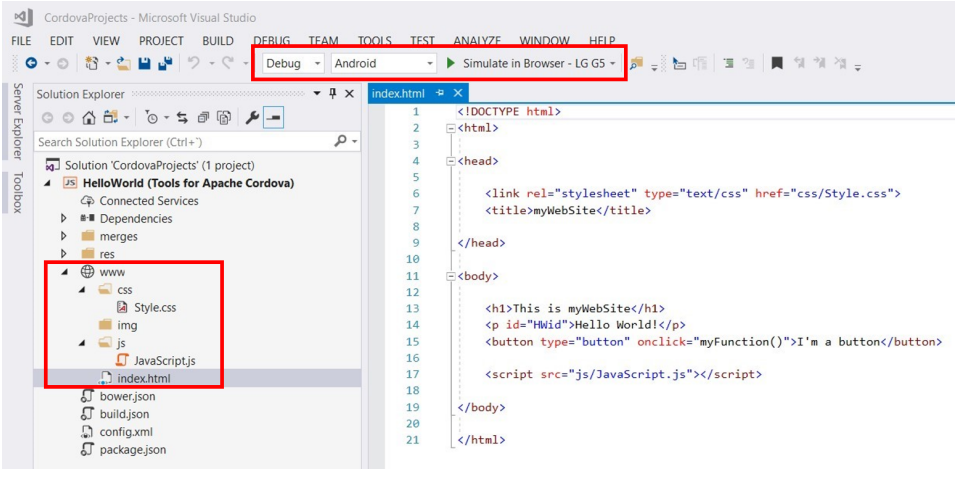

Figure 6.21: Solution explorer with our files already included, the `index.html` that we created in the previous sections is displayed.

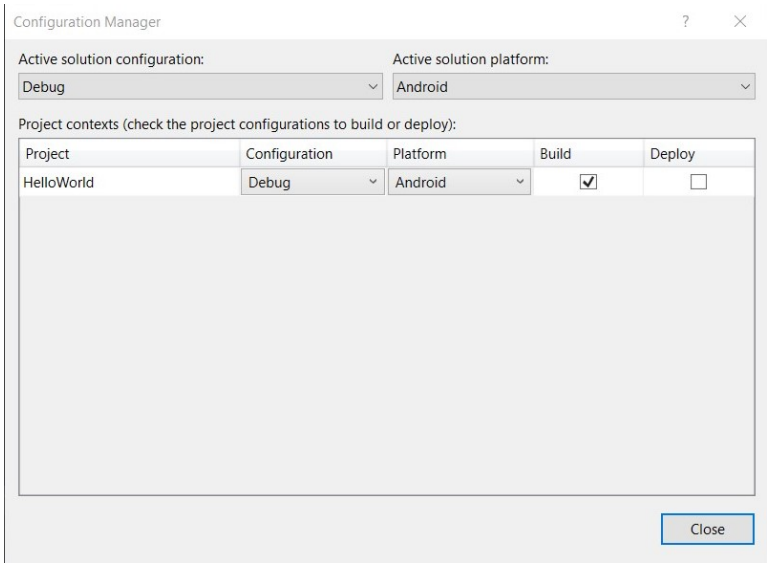

Figure 6.22: Configuration Manager of the solution, here we deselect the Deploy option that allows to transfer the `.apk` file to the smartphone automatically when it is connected.

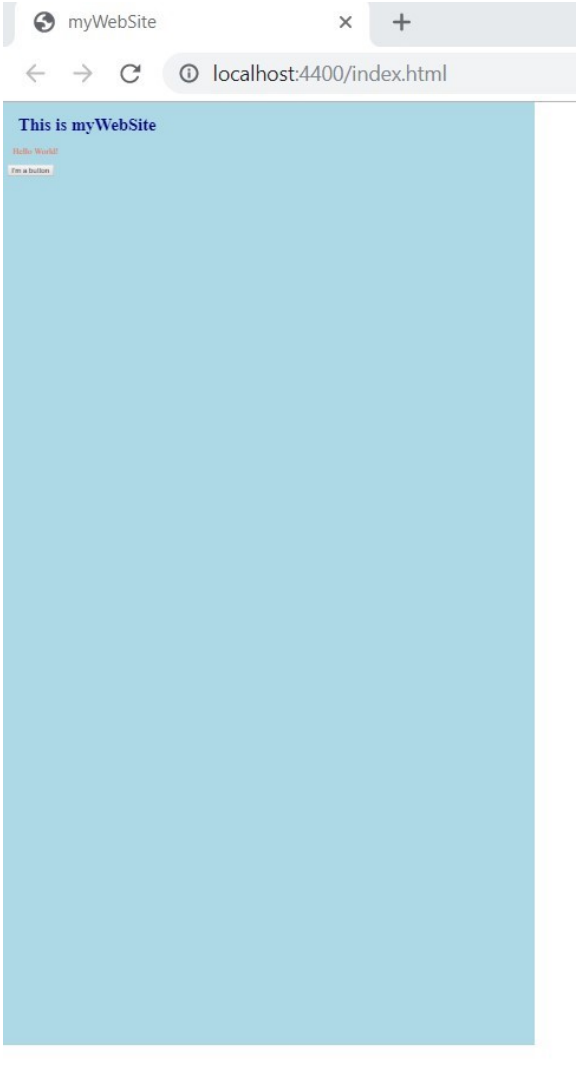

Figure 6.23: Simulation of the App in the browser, in this case simulating a LG G5 smartphone. The result has the same aspect ratio as the smartphone screen.

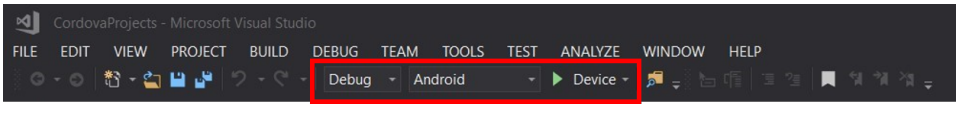

Figure 6.24: Configuration to choose in order to generate the `.apk` file that will be installed in the Android smartphone.

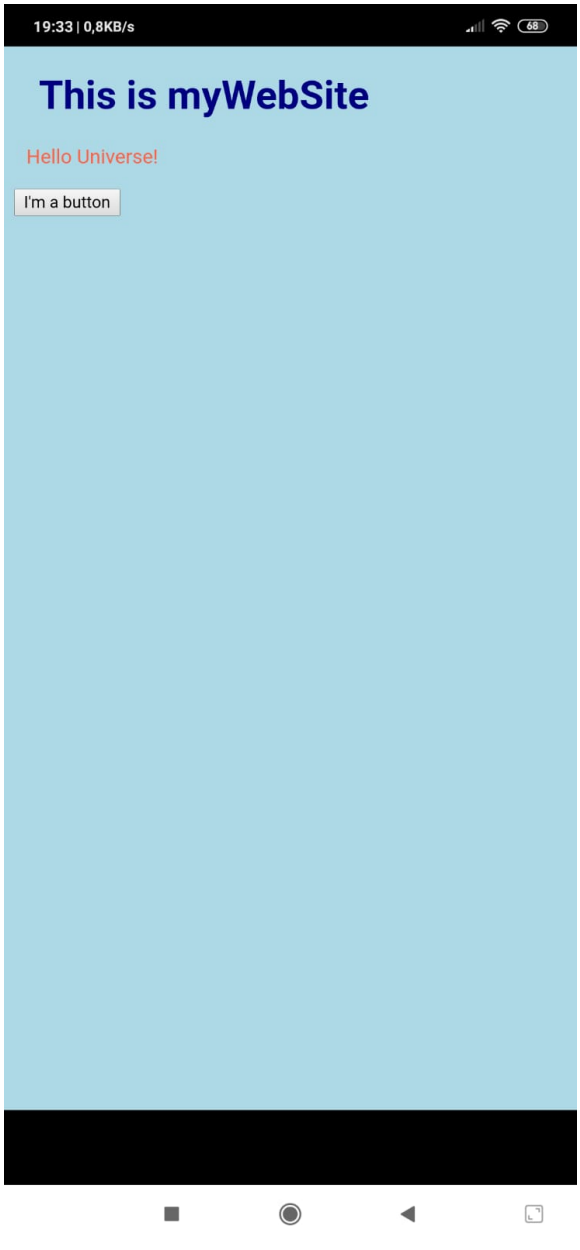

Figure 6.25: Phone Application installed in the smartphone through the `.apk` file.

Chapter 7

Git and GitHub

7.1 Introducing Git

Version Control Systems (VCS) are in charge of registering all the modifications made on a bunch of files, saving copies of the files, recording what changes have been made, at what time, by who, etc. The following actions can be carried out by a version control system such as Git:

1. Old versions of the project can be recovered in the future.
2. The evolution on the project can be tracked.
3. It is easier to collaborate on the same project with a group of people.
4. Incompatible or simultaneous changes in the same files can be managed by conflict resolution tools.

Git is a software to track changes in computer files what makes it ideal for software development (individual and among groups). Once it is installed in the computer any project can be managed by Git, that project is then called repository. When some project is tracked by Git, a `.git` hidden folder and some other files (i.e. the file `.gitignore`) appear in the repository. The version control system can be used from the command prompt

once `git` is installed. Those `git` commands can be also be invoked from graphical interfaces such as: Visual Studio, GitHub or GitHub Desktop.

It is important to distinguish between the local repository (your personal computer) and the remote repository such as GitHub. GitHub is a provider of Internet hosting for software development and version control using Git. It is a web-based hosting service that allows to store files in the cloud in a private or public repository. The next sections covers the creation of local and remote repositories as well as procedures to work with collaborators and teams by developing software or modifying files.

7.2 Create a remote GitHub repository

First of all, a GitHub account must be created to store remote repositories. Go to *github.com/join* and create a GitHub account associated to your email. There are different ways to create a remote repository depending on the graphical user interface you use:

1. **From the GitHub web page.** Once our GitHub account has been created, login in the GitHub web page with our credentials and click + button close to your photo to create a repository. Follow the steps to create the basic files of the repository such as `.gitignore` file, and `README.md` file to start with.

2. **From Visual Studio.** Open some Visual Studio project that you are working with. Click on *GIT/Create Git Repository*, select the GitHub account, choose the repository name and create and push. Browse through your GitHub web page to see your new repository.

3. **From GitHub Desktop.** Open GitHub Desktop and sign in with your GitHub account (*File/Options/Accounts*). Create a new repository (*File/New repository*) by selecting its name, a `REAMDE.md` file and a `.gitignore` template. Click on *Create repository* and then *Publish repository*. Browse through your GitHub web page to see your new repository.

7.3 Clone a remote repository

There are three different ways to clone a remote repository so a local repository is created. Go to any GitHub public repository and from the green button *Code* click on:

1. **Open with GitHub Desktop.** This option opens the GitHub Desktop application and allows to select the local path where this repository will be stored. Click on *Clone* to create the local repository.

2. **Open with Visual Studio.** This option opens Visual Studio and allows to select the local path where this repository will be stored. Click on *Clone* to create the local repository.

3. **Download zip.** This option downloads a zip file that can be decompressed in any selected folder.

7.4 Workflow and nomenclature in Git

Before dealing with the recommended workflow to work with Git, it is important to define nomenclature and important concepts to understand the proposed protocol. These concepts are shown graphically in figure 7.2. Once files are under `Git`, they are `TRACKED` which means that their modifications are registered every time a `COMMIT` is performed. Once theses changes are validated, they are uploaded to the cloud by a `PUSH` order. On the other hand, when the most recent version of the remote repository is needed, a `PULL` order will allow to download files from the remote repository to the local repository.

Sometimes, auxiliary or executable files do not need to be tracked because they are automatically generated from sources files and their changes depend only on the source files. By that reason, those files are ignored by `Git` by specifying in the `.gitignore` their specific extension such as: `*.exe` or `*.aux`.

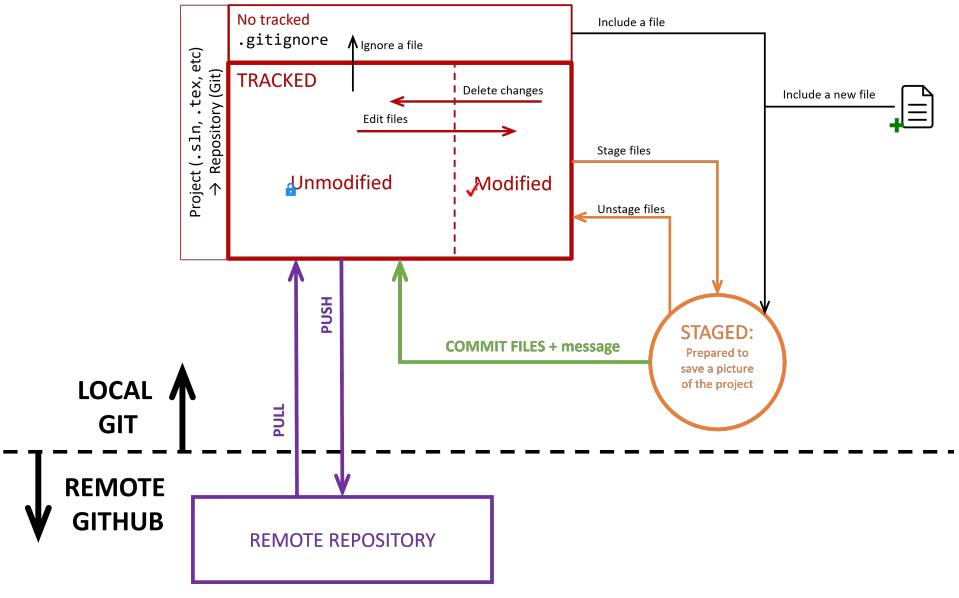

Figure 7.1: Main states of Git and GitHub workflow.

7.5 How to work with Git

Once a remote repository has been created, later improvements or modifications should be carried out with a pristine protocol in order to be efficient in developing software. Some specific rules are proposed to define this protocol. To modify some part of a software code, decide the name of the specific task you are going to deal with and create a new branch based on the newest version of the master branch. Even if you are the owner of the repository, do not make changes in the master branch. Use your new branch to validate your changes. Once these changes are validated, try to merge this branch with the master branch to consolidate those changes. The following procedure should be carried out:

1. **Modify**. Make changes of sources files, compile and test.

2. **Commit changes**. Once your changes are clear enough to solve or improve your specific task, commit changes. The frequency of different commits depend on the expertise of the programmer or developer but it is always safer to increase the frequency of commits if you don't fell comfortable with those changes.

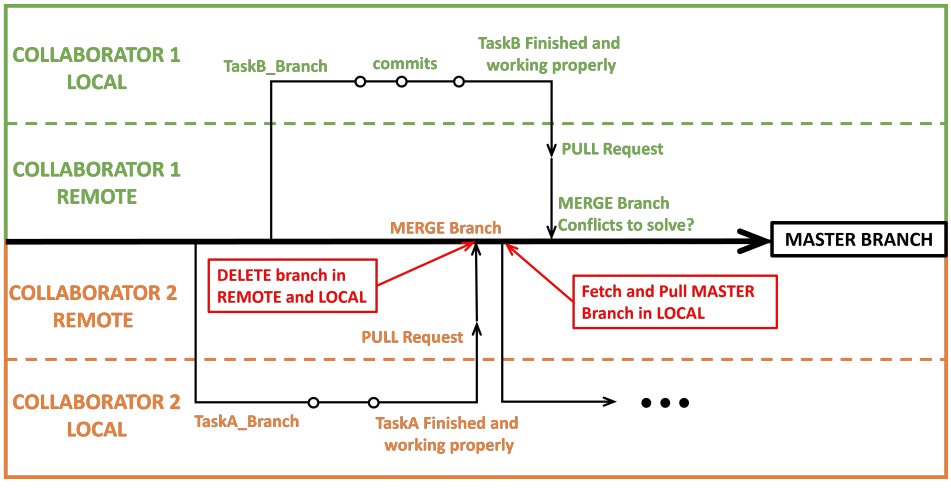

Figure 7.2: Recommended Workflow when collaborating with GitHub.

3. **Push changes**. Once a bunch of committed changes are validated, upload to changes to GitHub with a push order. It allows to have the more recent version of your branch uploaded to GitHub if some collaborator want to check it or to give you feedback.

If collaborators are going to work in some repository, the owner of the repository could invite people to collaborate (*settings/manage access*). Even if you are not invited to collaborate, you can create a fork of any public repository and to propose later a pull request.

7.6 How to merge two branches

When the developer decides his task is finished and ready to be shared, he should merge his branch with the master branch. If only one person is working or if the developer is working only in one specific task at a time, no conflicts are going to arise when trying to merge different branches (notice at this point the importance of having the branch you work on based on the most updated version of the main branch). However, if two collaborators are modifying the same file, conflicts should be resolved. This process is shown graphically in figure 7.2 where two collaborators are working on two different branches or tasks. As it was mentioned before, it is a good practice not to

modify the master branch and to create specific branches with a succinct label describing the purpose of the task or branch. The following steps represent the proposed methodology to work with collaborators:

1. Collaborators 1 and 2 create branches b_1 and b_2 respectively based on the same master branch.
2. Imagine that collaborator 2 finishes his task before collaborator 1 and merges his branch b_2 with the master branch. He will not find conflicts because no one has merged with the master branch after branch b_2 was created.
3. Imagine that collaborator 1 finishes his branch and tries to merge. If both collaborators are working on the same files, collaborator 1 could find conflicts when trying to merge branch b_1 with the master branch because at the time he tries to merge, master branch is different from what it was previously based on. Hence, when working in teams possible conflicts could arise and they should be resolved.

There are ways to merge and to resolve possible conflicts of two different branches: *(i)* From GitHub and *(ii)* From Visual Studio.

7.7 Merging and resolving conflicts from GitHub

From from GitHub or Visual Studio create a pull request to pull changes of some specific branch to be merged with the master branch. Now, from GitHub web page *create the pull request* and *merge* accepted changes if no conflicts appear. If conflicts appear, GitHub allows to open with a web editor those files associated with conflicts. By deleting manually those parts of files which are not accepted, conflicts are resolved. Once this is done, *Mark as resolved* to accept changes by a *Commit merge*. The following steps summarize the merging process, from GitHub, of two branches with conflicts:

1. Create pull request
2. Open the web editor and solve conflicts.
3. Mark as resolved.
4. Commit merge.

7.8 Merging and resolving conflicts from VS

Select *Git/Manage branches* and from the master branch (it has to be active), merge by clicking on the specific branch with the right button and choose *merge branch into master*. If conflicts appear click on *Resolve conflicts* and then look for Git changes where a list of files with changes with conflicts will appear. Click right button on each file and select **merge**. Three windows will appear showing the incoming modification file, the current master file and the desired result or merge file. By selecting different ticks or differences between the modification file and the master file, the result file is built. Once you agree with the result file, click on **Accept merge** and commit the merge. The following steps summarize the resolving process when merging two branches with conflicts:

1. Select master branch.
2. Select *Git/Manage branches*.
3. Click right button on the branch to merge and click on *Merge this branch into master branch*.
4. Click on *Resolve conflicts* (in blue).
5. Select *Git changes* to identify a list of files with conflicts.
6. Click right button on the file presenting conflicts and select **merge** .
7. Select differences or ticks between the two branches to merge and validate the result.
8. Accept merge.
9. Commit the merge result.
10. Push to the remote repository.

Once conflicts are resolved and some specific branch is merged with the master branch, it is mandatory to delete the local and the remote specific branch which was used to work in this task. If a new modification is going to be carried out, a new branch should be created based on the most recent update of the master branch.

7.9 How to revert specific commits

There are three different stages from which modifications or changes can be reverted:

1. Before changes are committed.
2. After changes are committed.

If changes are not still committed, select *Git changes* and a list of files that have been modified is shown. Click right button on the file we want to revert the change and select *Undo Changes* to revert to the last change or select *View History* to recover some specific version of this file.

If changes are already committed, select *Git/View Branch History* to inspect different commits of changes and proceed with the following steps:

1. Double click in any commit to revise the commit.
2. Click *Revert* (in blue).
3. Select *Git changes*. Click right button on the file and select **merge**.
4. Select differences by observing the result.
5. Accept merge.
6. Commit the merge result.
7. Push to the remote repository.

7.10 Create issues

Issues are really useful for a bunch of common situations when collaborating within a team: ask for some help or give feedback to your collaborators, make a change request before closing a pull request through a merge or used them as an indicator of where the work is being developed. These *issues* are retained as part of the timeline of the project so they can also be used to revise previous steps of the development. Furthermore, creating issues could be used as a planning for the future work, so the requirements

of a branch can be captured in an *issue* in such a way that it is closed when all the needs are covered and merged in the main branch.

By default *issues* are enabled in the GitHub repository, however, the administrator could disable this option. In case the team considers useful to use them as a communication tool between the partners it can be interesting to define a template ensuring that all basic information is covered when reporting problems, changes, etc.

With the repository of GitHub opened (and already signed in with your GitHub account), to create an issue you can click on *Issues* and click on the green box *New issue*. You can also launch the issue from GitHub Desktop by clicking on *Repository/Create issue on GitHub*. Then, follow these steps:

1. Give a title and description to the issue, for example, which is the error found? How to reproduce it? What changes are needed?
2. Define some extra information:
    - Which collaborators should be assigned to this issue to solve it? Click on *Assignees* on the right part of the window and select people so they became responsible of the task to be performed (they also receive an email notifying this issue).
    - Define some labels associated to the issue in *Labels* so they are classified and collaborators can filter them.
    - Link the issue to a Pull Request so it will be automatically closed when merge is definitely performed, to do that, click on *Linked pull requests*.
    - Add mentions to people through @Name so they receive an email notifying the mention.
    - Create references to other issues with # issueID.
3. Click on the green box *Submit new issue* to create it.
4. When the bug is fixed, the problem solved, the merged performed, etc. you can close the *issue* by clicking on *Close issue*.

From now on the issue appears in the repository for all the collaborators to check it, specially those declared as responsible of the task. Once created, more comments can be attached by collaborators to discuss how to solve

the problem and even *milestones* can be added so the evolution of the task is tracked.

It is responsibility of the team to maintain the issues updated with the evolution of the project. For example, an issue should be closed if the task is finished or the problem solved.

7.11 GitHub classroom

GitHub classroom, by means of the called *organizations*, allows to create classrooms and teams, manage access to repositories, create assignments for students, collaborate with other people and more.

Once an *organization* is created (maybe grouping all students of one subject or maybe different subjects taught by a group of professors) some *classrooms* can be added. In each classroom a group of students is included using their GitHub accounts and their institutional emails. Teams among the students can be created by the teacher, administrator of the organization and classrooms, or by the students themselves at the time of accepting the assignment. They all will receive the assignments created for each classroom. However, not all the assignments has to be created for a team, they can also be an individual task for each student.

An assignment, whether it is individual or for groups, can be public or private, this option must be selected while creating the assignment. Notice that a public assignment allows students to see other classmates/groups work. In addition, an optional starter repository could be included to guide the students.

Once the assignment is created, the available link should be shared with all the students so they can accept the invitation for the assignment and start working. Then, for each student or team that accepts the invitation, a copy of the template repository is made for each student or group. A list of accepted assignments is available for the teacher so the progress of each one can be tracked. Furthermore, a chat is opened between the teacher and the student to facilitate the communication, share comments, ask questions or feedback, etc.

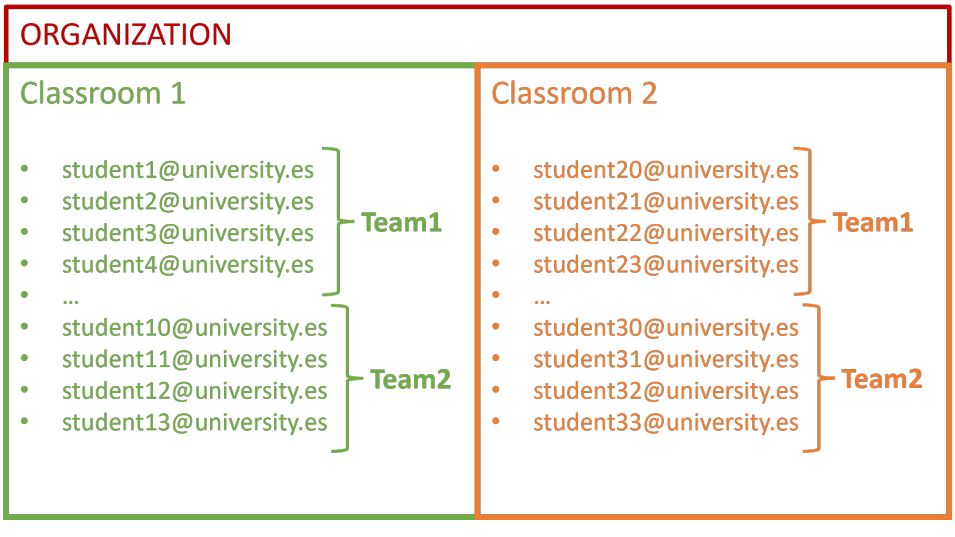

Figure 7.3: Basic structure for grouping students with GitHub Classroom.

References

[1] D. Conic-Jacob. Fortran file extensions. http://degenerateconic.com/fortran-file-extensions/. [Online; accessed 27-April-2018].

[2] Intel Software Development Tools. *Intel® Fortran Compiler User and Reference Guides-Document Number: 304970-006US*. Intel.

[3] S. Lionel. Source form just wants to be free. https://software.intel.com/en-us/blogs/2013/01/11/doctor-fortran-in-source-form-just-wants-to-be-free, . [Online; accessed 27-April-2018].

[4] S. Lionel. Visual fortran newsletter articles. https://software.intel.com/en-us/forums/intel-visual-fortran-compiler-for-windows/topic/275071#comment-1548440, . [Online; accessed 29-April-2018].

[5] Oracle. Sun studio 12: Fortran user's guide-4.9 module files. https://docs.oracle.com/cd/E19205-01/819-5263/aevog/index.html. [Online; accessed 28-April-2018].

[6] B. B. C. Society. What are the differences between .mod, .obj, .lib, .exp, .exe, .dll and .def? http://www.fortran.bcs.org/2003/porting/part_3/slide_06.html. [Online; accessed 27-April-2018].

[7] Wikipedia. Stack overflow. https://en.wikipedia.org/wiki/Stack_overflow, . [Online; accessed 01-May-2018].

[8] Wikipedia. Call stack. https://en.wikipedia.org/wiki/Call_stack, . [Online; accessed 01-May-2018].

[9] I. D. Zone. Understanding files associated with intel® fortran applications (windows*). https://software.intel.com/en-us/node/680056, . [Online; accessed 27-April-2018].

[10] I. D. Zone. Using module (.mod) files. https://software.intel.com/en-us/fortran-compiler-18.0-developer-guide-and-reference-using-module-mod-files, . [Online; accessed 29-April-2018].