

# Compiladores, Editores y Entornos de Desarrollo

Instalación, Configuración y Prueba

Esp. Ing. José María Sola, Ing. Edgardo Joel Peralta, Facundo Salerno & Ariel Silva

v1.0.0

## Abstract

Te contamos que es un compilador y cómo armar tu entorno de desarrollo para C y C++ bajo un entorno Windows, basado en el compilador MinGW y el editor de código fuente Visual Studio Code.

## tl;dr

Primero bajate e instalá el [compilador MinGW](#), durante la instalación asegurate de elegir g++, gcc, y make. Después configurá tu [variable de entorno path](#) para que llegue a la carpeta con los ejecutables. Bajate e instalá [Visual Studio Code](#) y también su extensión para [C/C++](#). Para probar escribí en Code un programa que diga [hola](#), en [C](#) o en [C++](#) y compilalo desde la terminal de Code; si lo escribiste en C compilalo con el comando con `gcc hello`, si está en C++ con el comando `gpp hello`. Para ambos casos, ejecutalo con el comando `a.exe`. Si todo salió bien, vas a ver un saludo.

## Parte I: Introducción

### Compiladores

Un compilador es un programa que genera programas, por eso lo podemos ver como un metaprograma. Dado un texto en un código fuente de alto nivel de abstracción, un compilador genera un código objetivo de bajo nivel; es decir, no solo es un traductor de un lenguaje a otro, si no que baja el nivel de abstracción para que pueda ser ejecutado en una máquina, ya sea real o virtual. Se lo puede ver como una función matemática que dado un string con el programa en alto nivel, genera otro string con un programa en bajo nivel.

### Familia de Compiladores

Cada par *lenguaje fuente* y *lenguaje objetivo* define una *familia de compiladores*.

Por ejemplo, el par (C, [x86](#)) y el par (C++, x86), cada uno define una familia de compiladores diferente. La primera familia, dado un programa en C genera código máquina para la arquitectura x86, la segunda familia también genera para x86 pero dado un código C++. Aunque ambas

familias generan programas para x86, cada familia de compiladores lo hace desde un lenguaje fuente diferente, uno desde C y otro desde C++.

Otros ejemplos son los pares: (C, x86), (C, ARM), (C, PowerPC). Cada uno es una familia de compiladores diferente, aunque mismo lenguaje fuente tienen diferente lenguaje objetivo.

¿Por qué lo llamamos familia de compiladores? Porque dado un par hay infinitas formas de implementar la transformación.

Por ejemplo, dado el par (C, x86) hay infinitas formas de implementar la transformación de lenguaje C a código máquina x86, por lo tanto hay teóricamente infinitos compiladores para ese par. Un compilador en particular es una implementación real de una transformación particular y es un miembro de una familia de compiladores.

Otro ejemplo es la familia (C++, x86) como ejemplo, los compiladores [g++ de GNU](#) y [cl.exe de Microsoft](#), ambos para plataformas Windows, pertenecen a la misma familia de compiladores pero son diferentes ya que, aunque ambos compilan de C++ a x86, lo hacen con métodos y capacidades diferentes.

## Entornos de Desarrollo

Como el resto de los programas, los compiladores corren en el contexto de un sistema operativo; dependiendo del sistema, la integración puede ser más o menos orgánica. El compilador, el editor, el depurador, las bibliotecas, y otras utilidades conforman la *caja de herramientas de desarrollo*, junto con el sistema operativo donde corren, constituyen el entorno de desarrollo.

### Editores

Una actividad principal del desarrollador es escribir código fuente, quizás la principal es leer código fuente. Para eso es necesario poder acceder al texto del programa de forma amigable y eficiente con [editores de texto plano](#) ó, hace ya unos años, [editores de código fuente](#), que son editores de texto especializados para el código fuente. Ejemplos de editores de texto de código fuente son Visual Studio Code, [Notepad++ \(Windows\)](#), [Sublime Text](#), [TextMate \(macOS\)](#), [vi](#), y [Vim](#).

### IDEs

Los [IDE](#) (Integrated Development Environment, Entornos Integrados de Desarrollo) son paquetes de software que proveen una solución todo en uno: desde el editor, pasando por el compilador, depurador, y empaquetador, hasta la ayuda sobre la tecnología de desarrollo y otras herramientas que facilitan el desarrollo. Estos sistemas son muy productivos, pero pueden ser avasalladores al principio y consumir muchos recursos de nuestro sistema.

## Integración con Sistema Operativo

En [sistemas operativos tipo-Unix](#), como por ejemplo [GNU/Linux](#) y [macOS](#), hay una integración orgánica entre el sistema y las herramientas desarrollo basadas en un compilador C. Esto se debe a que la relación entre [Unix](#), [creado por Ken Thompson](#) y [Dennis Ritchie](#), y el Lenguaje C, creado por Ritchie, es muy estrecha, y se remonta al origen de ambos con el compilador de C llamado cc (C compiler). En esos sistemas es común que haya pre-instalado un compilador de C/C++;

aunque en macOS requiere la instalación a demanda de [Xcode o sus Command Line Tools](#). Los compiladores más populares para estos sistemas con [gcc](#) y [clang](#).

En [Windows](#) la historia es diferente, aunque existe una gran cantidad de compiladores libres y comerciales para ese sistema operativo, no tienen integración orgánica como sí ocurre en sistemas tipo Unix. Pero hay una opción que es [MinGW](#), un paquete de software que *porta* las herramientas de desarrollo de entornos tipo-Unix y las disponibiliza en Windows.

## OK, Todo Bien ¿PeroCuál Compilador Instalo?

La respuesta rápida es: "**cualquiera**, siempre y cuando compile para [C18](#) en el caso de C y para [C++17](#) o superior en el caso de C++."

Lo interesante es que explores, pruebes y decidas por vos mismo. Pero, para facilitar los primeros pasos, en la próxima parte de este texto te contamos cómo instalar y dejar funcionando un entorno desarrollo basado en el set de herramientas MinGW y el editor de código fuente Visual Studio Code corriendo sobre el sistema operativo Windows.

Si tenes un GNU/Linux, probablemente ya tengas instalado gcc o clang, así que MinGW no es necesario. Es también probable que estés acostumbrado al uso de [la línea de comando](#) y tengas un editor de texto preferido, pero de todas maneras también puedes instalar [Code para GNU/Linux](#) o un IDE como [Eclipse CDT](#), que también está disponible para Windows y macOS.

Si tenes macOS, el estándar de facto es el [Xcode](#), un IDE muy popular para esta plataforma. Recordá que macOS requiere que explícitamente [descargues las herramientas de desarrollo para línea de comando](#). Pero también puedes armar tu propio entorno basado en Code, en un editor de texto, o en otro IDE como Eclipse CDT.

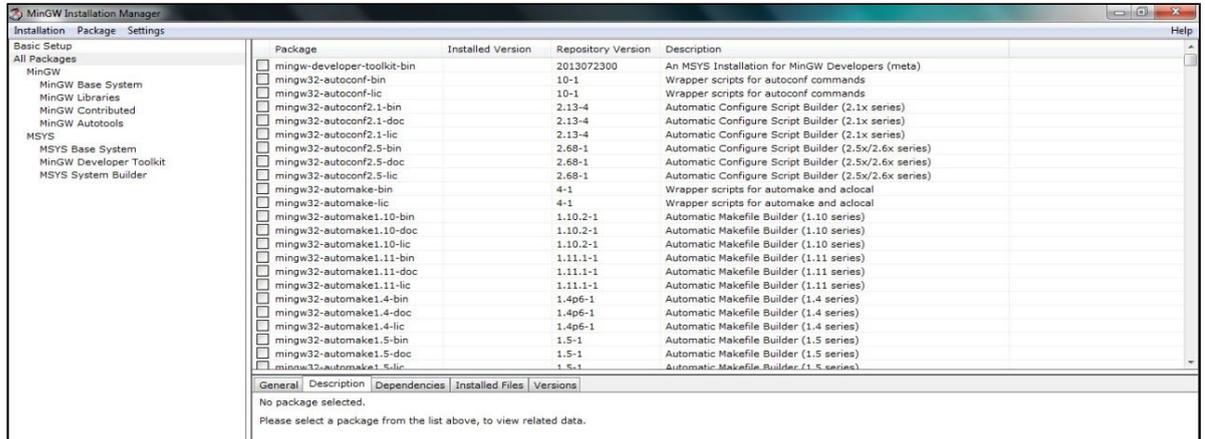
## Parte II: Armado de Entornos de Desarrollo

### Entorno de Desarrollo: Windows - MinGW - Visual Studio Code

Entorno de desarrollo está formado por el sistema operativo Windows, el compilador y herramientas provistos por MinGW y el editor de código fuente Visual Studio Code.

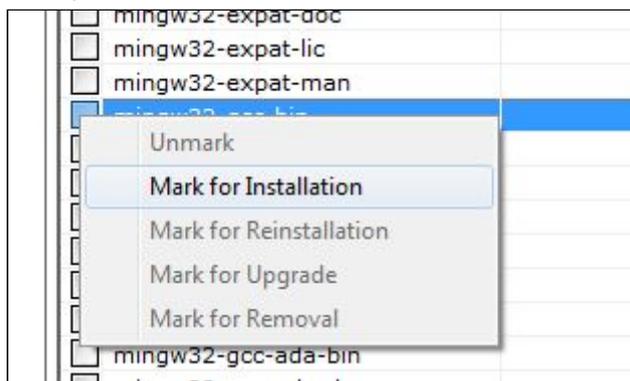
# 1. Instalación y Configuración de MinGW

1.1. Primero [descargamos MinGW](#) y lo instalamos. Una vez instalado, tenemos que abrir su Installation Manager, el cual se suele abrir sólo al finalizar la instalación:

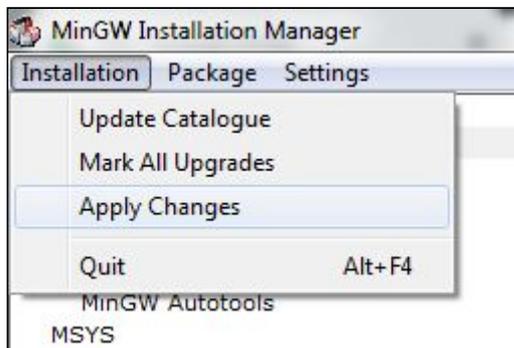


1.2. Seleccionamos de la lista de paquetes:

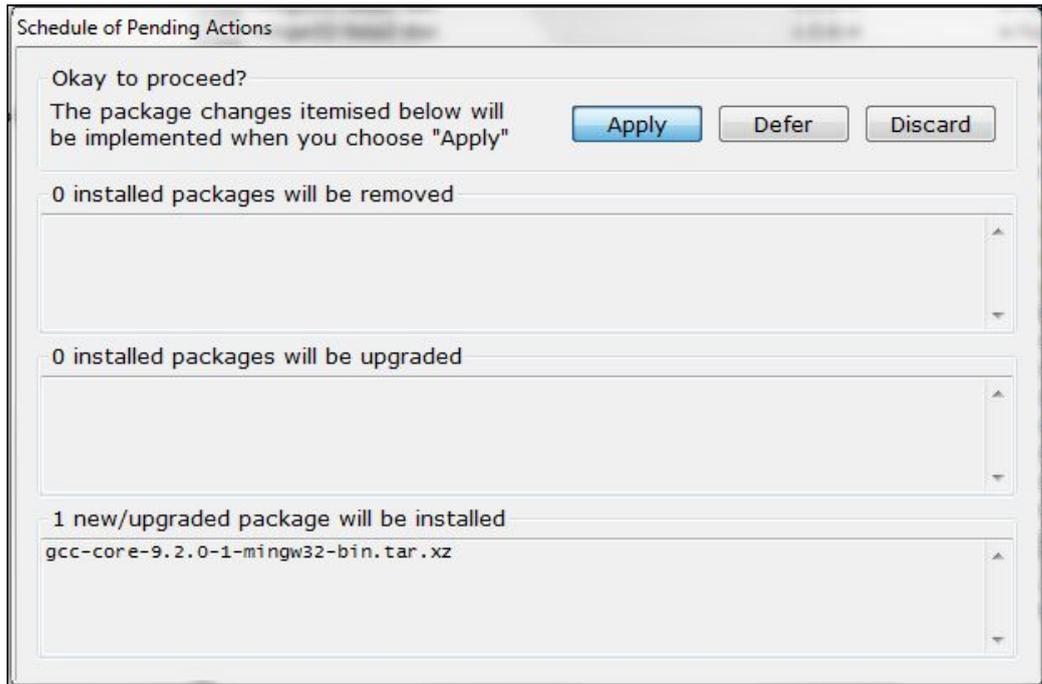
- ✓ mingw32-gcc-bin
- ✓ mingw32-g++-bin
- ✓ msys-make-bin



1.3. Una vez seleccionados, hacemos clic en: Instalación > Aplicar cambios:



1.4. Pulsamos "Apply", y esperamos a que termine:



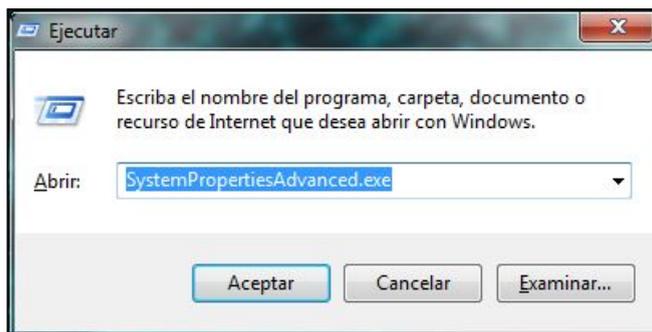
Una vez finalizado ya podemos cerrar las ventanas.

## 2. Path a MinGW

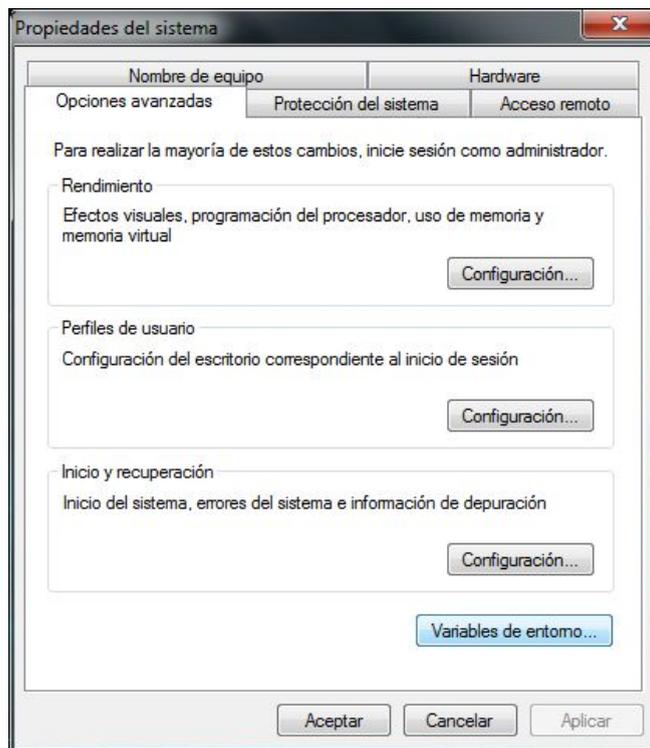
2.1. Necesitamos indicarle a Windows donde se instalaron las herramientas de MinGW para que sean accesibles siempre. Para eso existe una configuración que son las variables de entorno. Apretamos las teclas Win + R



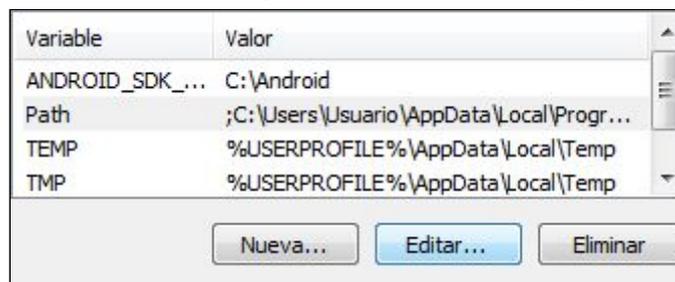
escribimos 'SystemPropertiesAdvanced.exe' y le damos a Enter:



- 2.2. Aparece la ventana "Propiedades del sistema", en la solapa "Opciones avanzadas", pulsamos "Variables de entorno".



- 2.2.1. En **Windows 7**, elegimos la variable "Path" y pulsamos "Editar":



Agregamos al final de lo que haya el siguiente texto, sin las comillas y comenzando con un punto y coma:

`";C:\MinGW\bin;C:\MinGW\msys\1.0\bin"`.

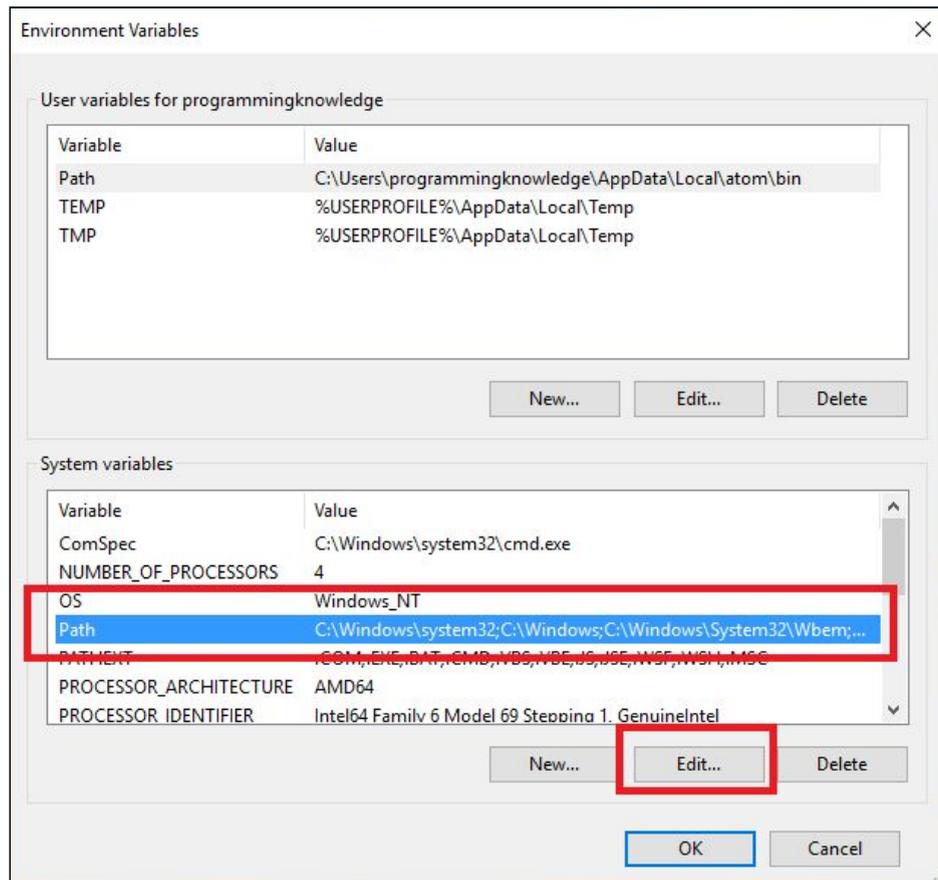
Si elegiste instalar MinGW en otra carpeta, tenes que cambiar el texto:



También podría darse la situación en la cual no exista la variable 'Path' aun. En este caso, simplemente la creamos haciendo clic en 'Nueva'.

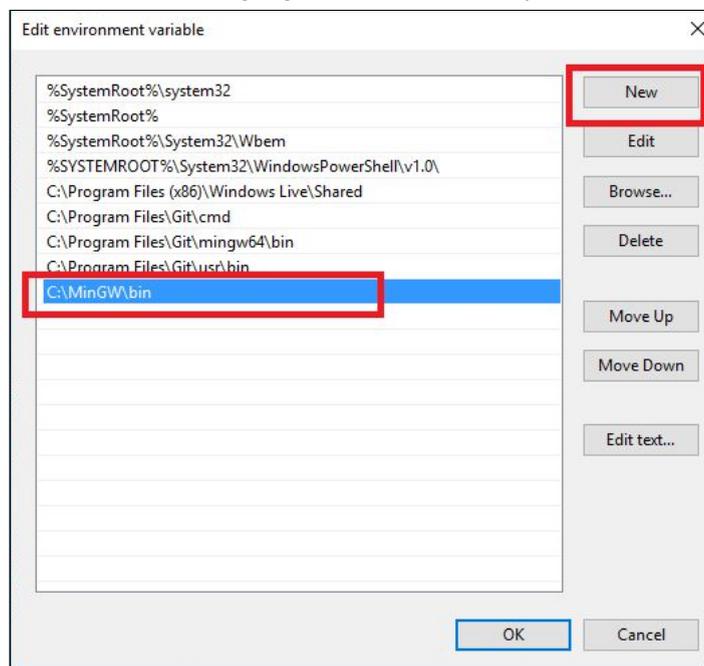
Guardamos todos los cambios pulsando en Aceptar.

- 2.2.2. Si tenemos **Windows 10**, el procedimiento es análogo. Seleccionamos la que dice "Path" y pulsamos "Editar":



Agregamos los dos paths, pulsamos "New" y escribimos sin las comillas: "C:\MinGW\bin".

Repetimos para agregar "C:\MinGW\msys\1.0\bin":

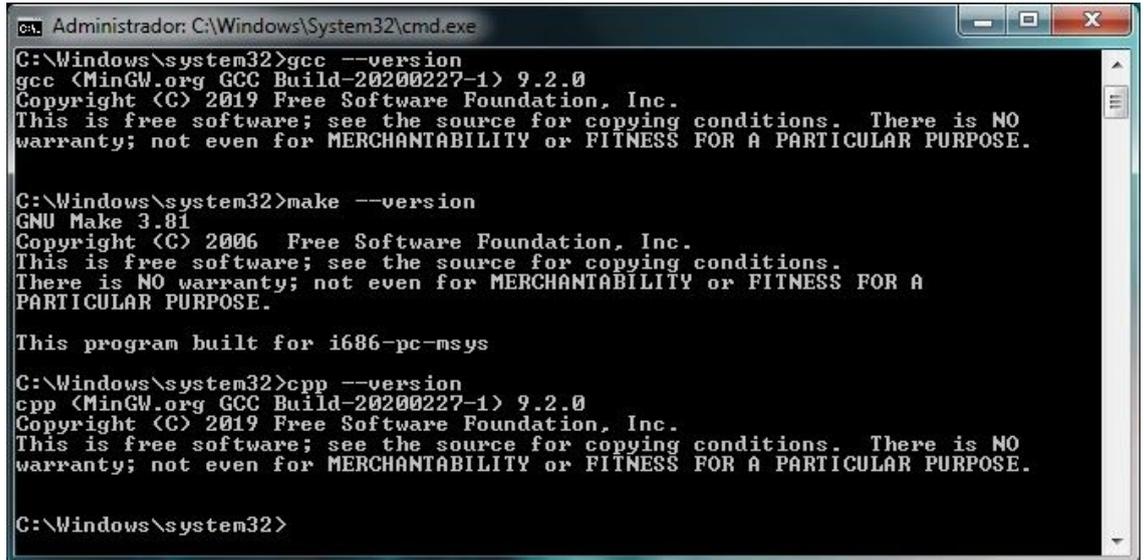


Guardamos todos los cambios pulsando "OK".

### 3. Comprobación de Instalación

Realizamos los siguientes pasos para verificar la correcta instalación:

- 3.1. Abrimos una ventana de línea de comando. En el Menú Inicio escribimos "cmd" y pulsamos Enter.
- 3.2. Aparece la línea de comandos. Escribimos sin las comillas "gcc --version" pulsamos Enter, y repetimos con "make --version" y Enter, y con "cpp --version" Enter. Deberíamos obtener un texto similar al siguiente:



```
Administrador: C:\Windows\System32\cmd.exe
C:\Windows\system32>gcc --version
gcc (MinGW.org GCC Build-20200227-1) 9.2.0
Copyright (C) 2019 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

C:\Windows\system32>make --version
GNU Make 3.81
Copyright (C) 2006 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.
There is NO warranty; not even for MERCHANTABILITY or FITNESS FOR A
PARTICULAR PURPOSE.

This program built for i686-pc-msys

C:\Windows\system32>cpp --version
cpp (MinGW.org GCC Build-20200227-1) 9.2.0
Copyright (C) 2019 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

C:\Windows\system32>
```

## 4. Instalación y Configuración de Code

Visual Studio Code, o simplemente VS Code ó Code, es un editor de código fuente multiplataforma con el cual vamos a crear y modificar nuestros programas y desde el cual también podemos compilar nuestro programa y correr el ejecutable resultante.

4.1. Descargamos e instalamos [Visual Studio Code](#).

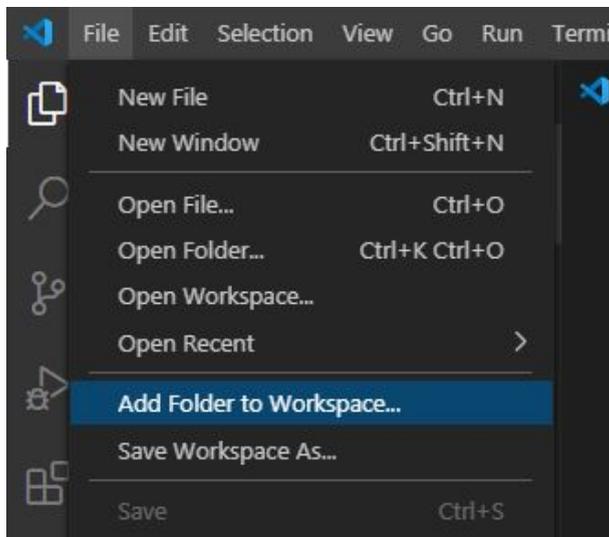
4.2. Descargamos e instalamos la [extensión para C/C++](#).

## 5. Escribimos Nuestro Primer Programa: "Hello, World!"

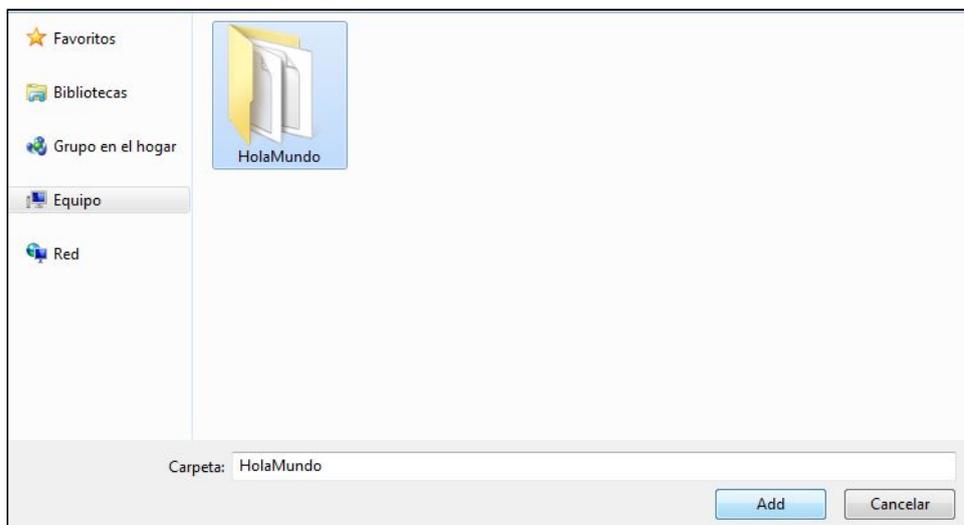
Escribimos un programa que diga "Hola, Mundo" para comprobar que nuestro entorno de desarrollo está operativo.

5.1. Creamos una carpeta en cualquier lado, en este ejemplo se llama HolaMundo. En esta carpeta va a estar el código fuente, el programa ejecutable resultado de la compilación y otros archivos de trabajo.

5.2. Abrimos Code y agregamos la carpeta recién creada y vacía a nuestro espacio de trabajo en Code mediante "File > Add Folder to Workspace".



5.3. Nos aparece una ventana donde buscamos la carpeta que recién creamos, la seleccionamos y la agregamos:



5.4. Escribimos nuestro primer programa:

5.4.1. Para lenguaje C lo vamos a guardar en hello.c, con este texto:

```
#include <stdio.h>
int main(void) {
    printf("Hello, World!\n");
}
```

5.4.2. Para lenguaje C++ lo vamos a guardar en hello.cpp, con este texto:

```
#include <iostream>
int main() {
    std::cout << "Hello, World!\n";
}
```

## 6. Compilación y Ejecución de Nuestro Primer Programa: "Hello, World!"

Hay varias formas de compilar. Presentamos cuatro formas y sus alternativas para C y C++: [make sin makefile](#), invocación explícita al compilador, [make con makefile](#), y Tasks de Visual Studio Code.

### 6.1. Alternativa 1 -- Make sin makefile:

6.1.1. Desplegamos el menú "Terminal" y elegimos "New Terminal Window".

6.1.2. Escribimos "make hello" y pulsamos Enter. Esto es independiente de si el lenguaje es C o C++ ya que make invoca a gcc o a g++ en función de la extensión del archivo. El estándar del lenguaje a utilizar en la compilación es el que gcc establece por omisión.

6.1.3. Escribimos "./hello.exe" y pulsamos Enter. Si todo está correcto debe aparecer el saludo.

### 6.2. Alternativa 2 -- Compilador invocado a explícitamente:

6.2.1. Desplegamos el menú "Terminal" y elegimos "New Terminal Window".

6.2.2. Para C escribimos  
"gcc hello.c -std=c18 -o hello.exe"  
y pulsamos Enter.

6.2.3. Para C++ escribimos  
"g++ hello.c -std=c++17 -o hello.exe"  
y pulsamos Enter.

6.2.4. Escribimos "./hello.exe" y pulsamos Enter. Si todo está correcto debe aparecer el saludo.

### 6.3. Alternativa 3 -- Make con Makefile:

6.3.1. Escribimos un makefile que es la receta para *make*ear programas. Para eso creamos un archivo llamado "Makefile", con la M en mayúscula, en la misma carpeta que el archivo fuente y con el siguiente contenido.

Para C:

```
SOURCE = hello.c
BIN = hello.exe
OBJ = hello.o
CC = gcc
CFLAGS = -std=c18
```

```
$(BIN): $(OBJ)
```

```
$(CC) $(OBJ) -o $(BIN) $(CFLAGS)
```

```
run: $(BIN)
      $(BIN)
```

```
$(OBJ): $(SOURCE)
        $(CC) -c $(SOURCE) -o $(OBJ) $(CFLAGS)
```

Para C++:

```
SOURCE = hello.cpp
BIN = hello.exe
OBJ = hello.o
CXX = g++
CXXFLAGS = -std=c++17
```

```
$(BIN): $(OBJ)
        $(CXX) $(OBJ) -o $(BIN) $(CXXFLAGS)
```

```
run: $(BIN)
      $(BIN)
```

```
$(OBJ): $(SOURCE)
        $(CXX) -c $(SOURCE) -o $(OBJ) $(CXXFLAGS)
```

- 6.3.2. Desplegamos el menú "Terminal" y elegimos "New Terminal Window".
- 6.3.3. Escribimos "make hello" y pulsamos Enter.
- 6.3.4. Escribimos "./hello.exe" y pulsamos Enter. Si todo está correcto debe aparecer el saludo.

#### 6.4. Alternativa 4 -- Tasks de Visual Studio Code

Esta alternativa requiere más configuración pero aprovecha el entorno Code y permite ser más productivo. Además, requiere definir un Workspace, tema el cual se trata más abajo. Se basa en definir tasks (tareas) que son accesibles desde Code.

- 6.4.1. Escribimos un makefile para C o C++ según la alternativa 6.3.1.
- 6.4.2. Elegimos el menú "Terminal" y seleccionamos el ítem "Configure Tasks". Nos debería aparecer un nuevo archivo llamado `tasks.json`, borramos todo su contenido y lo reemplazamos por el siguiente:

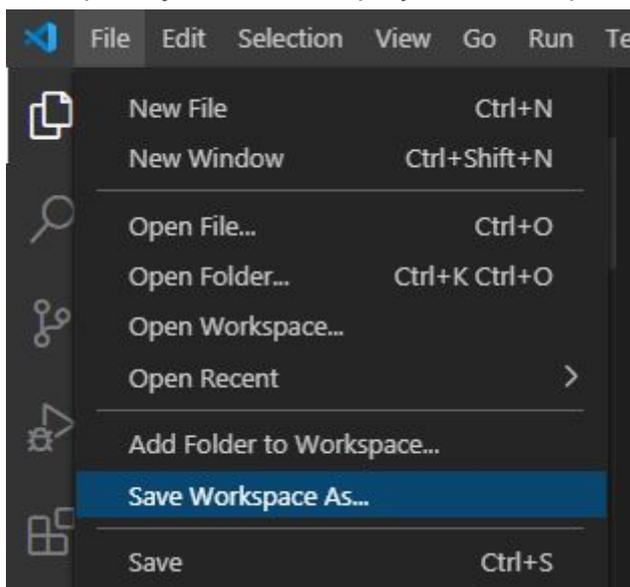
```
{
  "version": "2.0.0",
  "tasks": [
    {
      "label": "Build",
      "type": "shell",
      "command": "make",
      "group": {
        "kind": "build",
        "isDefault": true
      }
    },
    {
      "label": "Run",
```

```
        "type": "shell",  
        "command": "make run",  
    }  
  ]  
}
```

- 6.4.3. Guardamos el archivo.
- 6.4.4. Compilamos mediante Terminal > Run Build Task.
- 6.4.5. Por último, corremos el programa mediante Terminal > Run Task > Run.

## 7. Workspace (Espacio de Trabajo)

Code tiene el concepto de Workspace para representar el set de carpetas y archivos con los cuales estamos trabajando. Nombrar nuestro workspace nos permite volver al estado en el que dejamos nuestro proyecto más rápidamente:



Esta acción crea un archivo con el nombre que hayamos elegido y con extensión "code-workspace". Cada vez queramos abrir ese workspace, podemos abrir ese archivo.