

# Prácticas de Programación 1

Grado en Ingeniería Informática



Escuela de  
Ingeniería y Arquitectura  
Universidad Zaragoza

**Miguel Ángel Latre, Ricardo J. Rodríguez, Rafael Tolosana y  
Javier Martínez**

Área de Lenguajes y Sistemas Informáticos  
Departamento de Informática e Ingeniería de Sistemas



**Universidad**  
Zaragoza

**Curso 2024-25**



Distribuido bajo licencia CC BY-NC-SA. ©Profesorado del DIIS  
<https://creativecommons.org/licenses/by-nc-sa/4.0/deed.es>

# Presentación

Las prácticas de la asignatura **Programación 1** no deben imaginarse como una sucesión de sesiones en laboratorio a las que el estudiante debe acudir *a que le cuenten cosas*. Nada más lejos de la realidad; cada estudiante debe acudir a la sesión de prácticas de laboratorio con el trabajo propuesto en la práctica muy avanzado, deseablemente ya terminado, y aprovechar la sesión para aclarar dudas y para que el profesor supervise su trabajo y le indique, en su caso, cómo mejorarlo.

Las prácticas de **Programación 1** comprenden un conjunto de trabajos prácticos de programación que cada estudiante debe realizar con bastante autonomía a lo largo del cuatrimestre para alcanzar los resultados de aprendizaje que se describen en la guía docente de la asignatura<sup>1</sup>.

La información de este manual está organizada en seis capítulos. Los cinco primeros están asociados a una sesión de prácticas, y el último, al sexta sesión de prácticas y al trabajo obligatorio de la asignatura. Los capítulos se irán publicando conforme avance el curso.

- Cada capítulo comienza con una breve descripción de los objetivos que se persiguen realizando la práctica o el trabajo.
- Continúa con un apartado dedicado a la presentación de **elementos tecnológicos** del lenguaje de programación C++ y de algunas **herramientas** que facilitan el trabajo del programador. Este apartado hay que leerlo, comprenderlo y, lo más importante, lo que allí se describe hay que saber aplicarlo y utilizarlo a partir de ese momento.
- Finaliza el capítulo con la descripción de los **trabajos de programación** que cada estudiante debe desarrollar y se dan pautas sobre cómo hacerlos. Como se ha dicho anteriormente, el trabajo asociado a una práctica no se circunscribe a la sesión de dos horas asociadas a ella. El trabajo debe iniciarse en cuanto se publique el enunciado, para poder plantearse el objetivo de concluirlo durante la sesión asociada a la práctica y que pueda ser supervisado por un profesor, recibiendo las indicaciones y sugerencia que permitan, en su caso, mejorarlo.
- Cada estudiante debe tener claro que el trabajo de prácticas es una parte fundamental del estudio y aprendizaje de la asignatura y que **le corresponde adquirir la suficiente autonomía como para ser capaz de realizar trabajo por sí mismo**. El profesor le puede ayudar, pero esta ayuda es inútil si él no ha trabajado previamente lo suficiente.

Este manual es una guía inicial a las prácticas de la asignatura. Cada estudiante deberá acostumbrarse a acudir frecuentemente y consultar las siguientes fuentes de información, a las que se puede acceder a través del curso Moodle de la asignatura:

- En la sección de **Programar en C++** se presentan los enlaces a las páginas web con la documentación de las bibliotecas predefinidas en C++.

---

<sup>1</sup>[https://sia.unizar.es/documentos/doa/guiadocente/2022/30204\\_es.pdf](https://sia.unizar.es/documentos/doa/guiadocente/2022/30204_es.pdf)

- En la sección de **Código fuente del curso** se facilitará código y datos para el desarrollo de algunos de los trabajos propuestos en estas prácticas. Este material podrá ser descargado y copiado.

**A programar se aprende programando.** De ahí la importancia de que cada estudiante empiece a programar desde el primer día del curso. **Programar** es comprender el problema de tratamiento de información a resolver. **Programar** es analizar ese problema, dedicándole el tiempo que sea necesario, hasta decidir cómo abordar su resolución. **Programar** es escribir **en una hoja de papel** el algoritmo a aplicar para resolverlo. **Programar** es trasladar ese algoritmo a código en un lenguaje de programación determinado (en esta asignatura, C++), editarlo, compilarlo y ejecutarlo. **Programar** es someter nuestros programas a un completo juego de pruebas hasta que tengamos la convicción de que nuestro código no solo está libre de burdos errores, sino que su comportamiento satisface todas las especificaciones planteadas de partida para resolver el problema.

Todas estas tareas que acabamos de señalar son parte del trabajo de un programador y su realización en estas prácticas es responsabilidad personal de cada estudiante.

*Javier Martínez y los profesores de Programación 1  
Departamento de Informática e Ingeniería de Sistemas  
de la Universidad de Zaragoza*

# Práctica 1: Desarrollo en C++

## 1.1. Objetivos de la práctica

La primera práctica de la asignatura persigue los siguientes objetivos:

- Aprender a utilizar los puestos de trabajo disponibles en los laboratorios de programación.
- Aprender a utilizar un IDE (*Integrated Development Environment* o Entorno de Desarrollo Integrado) para el desarrollo y puesta a punto de programas escritos en C++.
- Estudiar el comportamiento de algunos programas C++ elementales, las instrucciones utilizadas en ellos y programar en ellos algunas pequeñas modificaciones.

**Es esencial que todos los estudiantes que cursan por primera vez la asignatura asistan y completen el trabajo propuesto en esta práctica, ya que lo que se aprende en ella deberá ser utilizado a partir de ese momento de forma continuada.**

Cada estudiante deberá desarrollar individualmente el trabajo propuesto en el apartado 1.3 *Trabajo a desarrollar en esta práctica*. Para ello conviene que, antes de participar en la sesión de prácticas, haya leído atentamente el guion completo de la práctica, haya comenzado a preparar el trabajo descrito en el apartado 1.3 y haya estudiado el contenido del apartado 1.5 *Manipuladores para dar formato a los datos de salida*.

## 1.2. Tecnología y herramientas

### 1.2.1. Los puestos de trabajo en los laboratorios del DIIS

Las prácticas de Programación 1 pueden realizarse tanto en los equipos del laboratorio del Departamento de Informática e Ingeniería de Sistemas (DIIS) como en los equipos personales de cada estudiante.

Cada uno de los puestos de trabajo de los laboratorios del DIIS consta de un computador personal conectado a la red de comunicaciones de la Universidad de Zaragoza.

Si el computador está apagado y lo ponemos en funcionamiento, nos permite seleccionar el sistema operativo con el que deseamos trabajar. En diferentes asignaturas utilizaremos normalmente uno de los sistemas operativos siguientes:

- **CentOS**. Sistema operativo derivado de Linux RHEL (*Red Hat Enterprise Linux*). Para trabajar con él hay que disponer de una cuenta de usuario. Cada estudiante dispone de una cuenta abierta por el administrador del sistema, una vez ha formalizado su matrícula en los estudios. En la primera



sesión de prácticas, se dará más información acerca de dicha cuenta, así como las credenciales necesarias para conectarse a la misma (nombre de usuario y contraseña).

Una vez cargado el sistema operativo CentOS, el usuario debe identificarse mediante su nombre de usuario (*username*) y su contraseña (*password*). Por ejemplo:

```
username: a333444
password: miClave856
```

- **Windows 10.** Se puede trabajar en él a través de una cuenta de usuario propia, como en el caso anterior, o a través de una cuenta de «invitado», siendo esta última la opción que utilizaremos en esta asignatura. La cuenta de usuario de invitado se abre automáticamente al encender un equipo y elegir Windows como sistema operativo. Si se desea iniciar sesión con la cuenta de usuario propia, hay que cerrar la sesión de invitado y proceder a identificarse mediante su nombre de usuario (*username*) y su contraseña (*password*). Debe hacerse a través del dominio DIISLAB, anteponiendo la secuencia «DIISLAB\» al nombre de usuario. Por ejemplo:

```
usuario: DIISLAB\a333444
contraseña: miClave856
```

El **nombre de usuario** y la **contraseña** para acceder a ambos sistemas operativos anteriores se pueden obtener a través de la pseudotarea «Credenciales de «hendrix» del curso de Moodle correspondiente a la asignatura. La contraseña de acceso a nuestra cuenta es **personal y debe mantenerse en secreto** para evitar un mal uso de nuestra cuenta por terceros, del cual seríamos corresponsables.

### 1.2.2. Una sesión de trabajo con el sistema operativo Windows

En los equipos del laboratorio, trabajando en la cuenta de invitado, una vez cargado el sistema, seleccionamos el icono [Conectar Unidades de Prácticas](#) y nos identificamos con el nombre de usuario y contraseña de nuestra cuenta. Si accedemos al equipo a través del explorador de ficheros y carpetas de Windows, veremos el efecto de la acción anterior: se han conectado dos unidades de red (denominadas «Y:» y «Z:») que están configuradas para acceder al servidor de docencia del DIIS.

Nos interesa especialmente la unidad «Z:», ya que en ella trabajaremos en esta asignatura y en ella podremos crear nuevas carpetas, borrarlas cuando queramos y almacenar en ellas ficheros.

La unidad «Z:» se almacena en un clúster de computadores Unix denominado hendrix, integrado por los computadores hendrix01 y hendrix02, conectados a la red informática de la Universidad de Zaragoza. De este modo, las carpetas y ficheros que creemos y almacenemos en la unidad «Z:» podrán ser accedidos desde cualquier otro puesto de trabajo de los que dispone el DIIS y se conservarán allí hasta que decidamos borrarlos.

El clúster hendrix realiza funciones de servidor de los ficheros de muchos centenares de estudiantes. Dado que su capacidad de almacenamiento es finita, cada usuario tiene limitada la cantidad de información a almacenar. Ello significa que cada usuario debe hacer una gestión adecuada de lo que se almacena, debiendo eliminar sistemáticamente aquellos ficheros que no sean necesarios para trabajar en el futuro.

Al terminar de trabajar con los equipos y antes de abandonar el puesto de trabajo es necesario desconectar las unidades de red, a través del icono del escritorio [Desconectar Unidades de Prácticas](#). Si hubiéramos iniciado sesión en nuestra propia cuenta, deberíamos cerrar dicha sesión.

Desde el menú [Inicio](#) son accesibles diversos programas de aplicación que podemos ejecutar.

En las prácticas de esta asignatura solo vamos a tener que ejecutar dos aplicaciones:

- Un navegador web para consultar diversas páginas web, especialmente las siguientes:
  - El curso Moodle de la asignatura Programación 1.
  - La web <https://github.com/prog1-eina/>, que contiene todo el código fuente utilizado en las clases de teoría, las soluciones a los problemas planteados en clase y el código de partida de las distintas prácticas.
  - La web <https://www.cplusplus.com/> con amplia y diversa documentación sobre el lenguaje C++.
  - Desde la página anterior se puede acceder al manual de referencia de la biblioteca estándar C++ en la dirección <https://www.cplusplus.com/reference/>.
- El entorno de desarrollo integrado Visual Studio Code (más información de este entorno en <https://code.visualstudio.com/>)<sup>2</sup>.

Cuando hayamos concluido nuestra sesión de trabajo deberemos salir de nuestra cuenta de trabajo o desconectar las unidades de red que hayamos conectado. Hay que evitar dejar la sesión personal abierta o nuestra unidad de red conectada para impedir que alguien pueda hacer un uso indebido de ellas.

### 1.2.3. Organización de los ficheros relacionados con la asignatura

Una **carpeta** (directorio) denominada «programacion1» alojará todos los ficheros que se desarrollen y generen en la asignatura y, en particular, en la realización de sus prácticas. Elige una ubicación para esta carpeta en tu equipo y utilízala para tener todo tu código organizado. Esta ubicación puede ser cualquiera, siempre que en su ruta de acceso completa desde el directorio raíz (por ejemplo, «C:\ » en Windows o «/» en Linux) no haya ninguna carpeta intermedia cuyo nombre contenga espacios en blanco o caracteres acentuados. Si utilizas los equipos del laboratorio, ubica la carpeta «programacion1» en la unidad «Z:» Evita trabajar en carpetas temporales o en la carpeta en la que se almacenan por defecto los archivos descargados de internet. A partir de la práctica 3, comenzaremos a reutilizar código de prácticas anteriores y, si no gestionas correctamente la ubicación de tu código fuente, tendrás problemas para reutilizar el código.

De forma orientativa, la organización que tendrá la carpeta «programacion1» al finalizar el curso se muestra a continuación:

```

programacion1..... Carpeta asociada a la asignatura
├── practica1..... Carpeta asociada a la 1.ª práctica
│   ├── bienvenida.cpp.....Código fuente del programa «bienvenida»
│   ├── eco-numerico.cpp.....Código fuente del programa «eco-numerico»
│   ├── ...
│   ├── potencias.cpp.....Código fuente del programa «potencias»
│   └── angulos.cpp.....Código fuente del programa «angulos»
├── practica2..... Carpeta asociada a la 2.ª práctica
│   ├── fecha.cpp.....Código fuente del programa «fecha»
│   └── ...
├── ...
└── practica6
    └── ...
  
```

<sup>2</sup>Puedes utilizar el entorno de desarrollo que desees, aunque los profesores tendremos más facilidad para ayudarte a resolver problemas habituales con el entorno si utilizas uno con el que estemos familiarizados.

Por el momento, nos vamos a limitar a crear, en algún lugar adecuado de nuestro disco duro (o en la unidad «Z:»), la carpeta «programacion1» y, dentro de ella, la carpeta «practical», que por el momento estará vacía de contenido.

### 1.2.4. Visual Studio Code. Un entorno de desarrollo integrado

Un entorno integrado de desarrollo (en inglés *integrated development environment* o *IDE*) es un programa informático que integra un conjunto de herramientas que facilitan diseño, escritura, ejecución, depuración y mantenimiento de programas. Dichas herramientas posibilitan la realización de las siguientes tareas:

- Gestionar nuestros proyectos de programación.
- Editar los ficheros fuente con el código de nuestros programas.
- Compilar el código de nuestros programas.
- Informar de errores en el código de nuestros programas.
- Depurar nuestros programas hasta que su comportamiento sea el deseado.
- Ejecutar nuestros programas cuantas veces sea preciso.

A partir de este momento vamos a utilizar el entorno Visual Studio Code para desarrollar programas escritos en C++. Puedes consultar su guía de instalación en Moodle, en la sección «Tutorial para la instalación de Visual Studio Code». Si trabajas en Windows o macOS, con casi total seguridad, tendrás que instalar un compilador de C++. En el tutorial de instalación de Visual Studio Code tienes también los enlaces a los tutoriales para instalar el compilador.

## 1.3. Trabajo a desarrollar en esta práctica

### Tarea 1

Se debe crear, si no se ha hecho ya, una carpeta (directorio) denominada «programacion1». Crea dentro de ella un directorio denominado «practical». Descomprime en «practical» los ficheros de extensión «.cpp» del repositorio de GitHub <https://github.com/prog1-eina/practical/>. Quita el sufijo «-master» que añade GitHub al nombre del directorio contenido en el fichero ZIP. Asegúrate de que te queda una estructura como la siguiente:

```
programacion1
├── practical
│   ├── angulos.cpp
│   ├── bienvenida.cpp
│   ├── eco-numerico.cpp
│   └── potencias.cpp
```

Desde Visual Studio Code, abre el directorio correspondiente a la carpeta «practical» a través de la orden **File** > **Open Folder...**. En el panel izquierdo de Visual Studio Code, verás los cuatro ficheros correspondientes al código fuente con el que trabajaremos en esta práctica. Haciendo doble clic en cualquiera de ellos, se abrirán en el editor.

Abre el fichero «bienvenida.cpp». Ejecuta el código, tal y como se explica en el tutorial de instalación de Visual Studio Code.

Modifica el código del programa, para provocar un error de sintaxis. Por ejemplo, elimina uno de los dos puntos que separan `std` de `cout`, dejando algo como esto:

```
std:cout
```

Intenta volver a ejecutar el programa. Observa cómo el editor señala un error en el uso de `cout` y cómo, en la pestaña **Terminal** del panel inferior, el compilador indica que hay un problema en la función `int main()`, relativo al ámbito de declaración de `cout`. Observa también que el compilador indica que el error está en la línea 15, reproduce luego la línea completa donde está el error y termina indicando en la línea siguiente, utilizando un acento circunflejo (^) a modo de flecha, el lugar donde comienza a tener problemas sintácticos el programa. Habitualmente, los errores sintácticos estarán donde apunta esa flecha o antes.

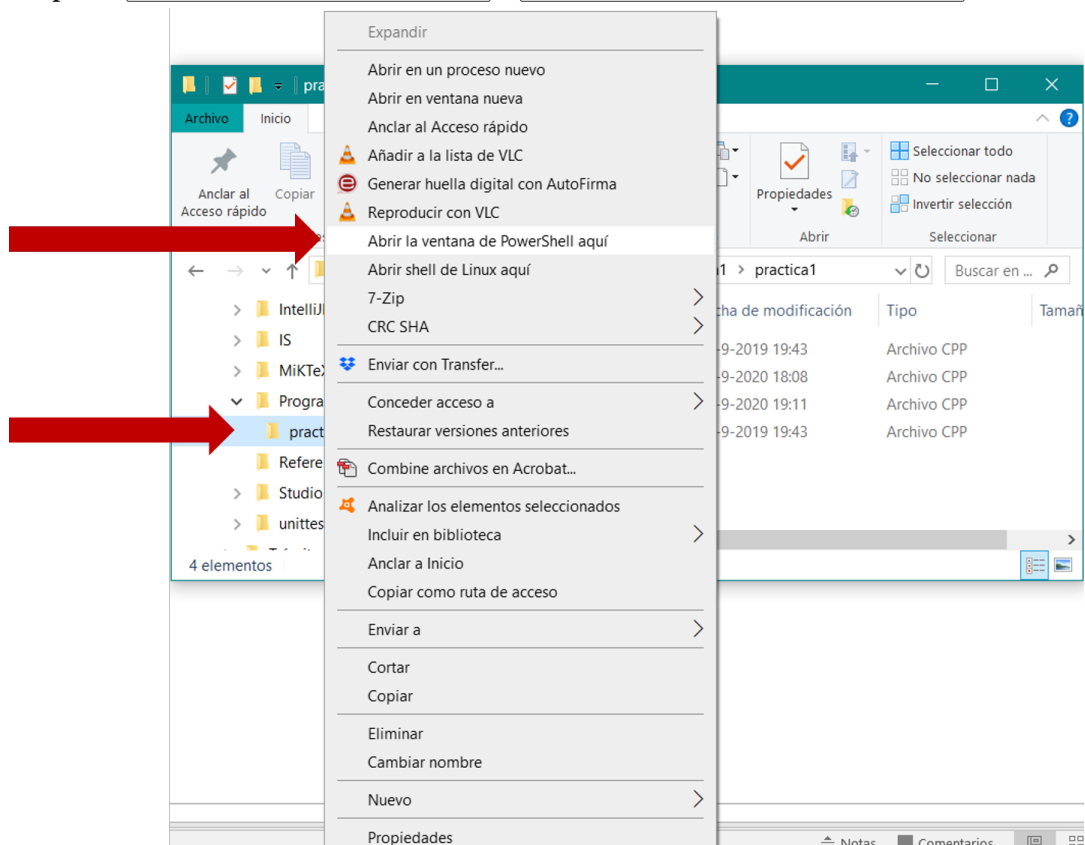
Déjalo con errores. Lo corregiremos en la siguiente tarea.

## Tarea 1b. Compilación desde el terminal

Por una vez, vamos a compilar y ejecutar el código del programa «bienvenida.cpp» directamente desde la línea de comandos la asignatura.

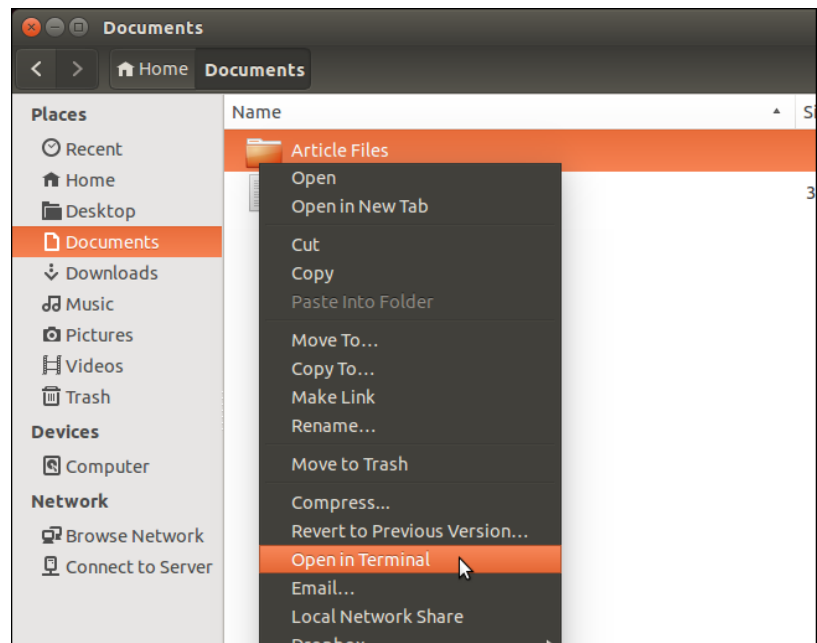
Para ello, sigue los siguientes pasos:

1. Abre un terminal y úbicalo en el directorio «practica1» donde está el fichero «bienvenida.cpp».
- En Windows, la forma más simple de conseguirlo es utilizar el Explorador de archivos para localizar el directorio «practica1» (posiblemente lo sigas teniendo abierto tras realizar la tarea 1). Abre el menú contextual extendido haciendo clic con el botón secundario del ratón en la carpeta «practica1» mientras mantienes pulsadas las teclas **Ctrl** + **Mayús** y selecciona la opción **Abrir la ventana de PowerShell aquí** o **Abrir la ventana de Símbolo de sistema aquí**:



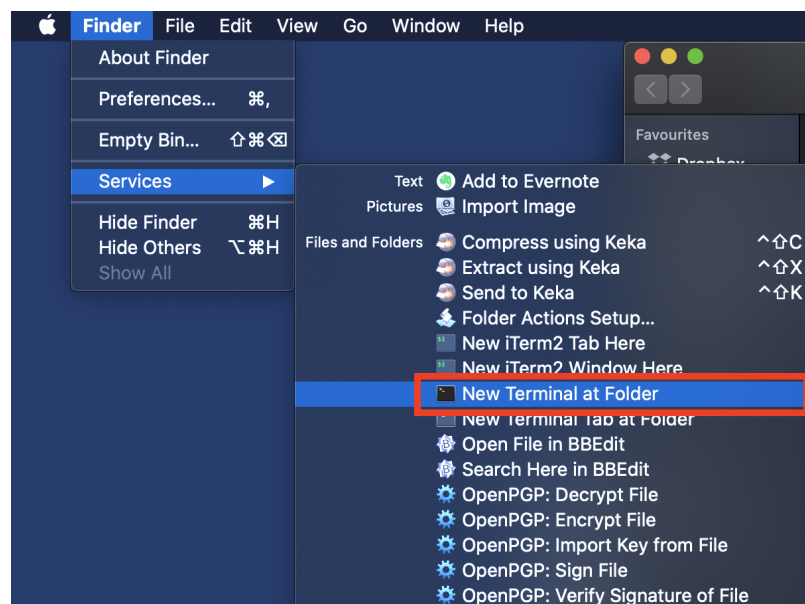


- En Linux, dependiendo del administrador de archivos que utilices, la forma puede variar. Por ejemplo, con el administrador de archivos de GNOME, existe la opción de menú `Open in Terminal` en el menú contextual de un directorio:



En cualquier caso, siempre puedes abrir una ventana del terminal y navegar en ella hasta el directorio «practical».

- En un sistema macOS también es bastante sencillo. Teniendo abierta una ventana del navegador de ficheros (llamado *Finder*) en la carpeta de interés (donde deseas abrir una terminal), simplemente acude al menú `Finder > Services > New Terminal at Folder`.



- En cualquiera de los tres sistemas operativos anteriores, siempre puedes abrir una ventana del terminal y navegar en ella hasta el directorio «practical». Aprenderás a desenvolverte mejor en una terminal en la asignatura Programación II.

<sup>3</sup>Fuente de la imagen:

<https://www.howtogeek.com/192865/how-to-open-terminal-to-a-specific-folder-in-ubuntu-file-browser/>

2. En el terminal, escribe la orden

```
g++ bienvenida.cpp -o bienvenida
```

Con esta orden, estás pidiendo al programa g++ que procese el fichero de código fuente «bienvenida.cpp» y que genere un ejecutable denominado «bienvenida» (en Linux y macOS) o «bienvenida.exe» (en Windows; la extensión «.exe» la añade automáticamente el compilador en este caso).

Comprueba como el compilador sigue indicando que hay errores en el código de «bienvenida.cpp» y que la salida del compilador es exactamente la misma que la que aparecía en la pestaña «Problems» del panel inferior de Visual Studio Code.

3. Corrige el error que hemos introducido en la tarea 1. Puedes usar cualquier editor de texto, incluido Visual Studio Code. Eso sí, asegúrate de guardar los cambios que hagas.

4. Vuelve a ejecutar la orden `g++ bienvenida.cpp -o bienvenida`

Si todo es correcto, el compilador g++ no dirá nada, y habrá generado el fichero «bienvenida» o «bienvenida.exe», dependiendo del sistema operativo.

5. Ejecútalo escribiendo `./bienvenida` si estás en Linux, macOS o Windows con PowerShell. Escribe solo `bienvenida` si estás en Windows con el Símbolo del sistema.

Verás que aparece el texto «Bienvenidos a UNIZAR».

## Tarea 2

Modifica el programa del proyecto «bienvenida» para que, al ser ejecutado, escriba lo siguiente en la pantalla, en dos líneas distintas:

```
Bienvenidos a UNIZAR
Bienvenidos a Prog1
```

## Tarea 3

Basándote en el código del fichero «eco-numerico.cpp», modifícalo para que pida dos datos enteros y escriba su suma en la pantalla:

```
Escriba un entero: 43
Escriba otro entero: -16
Su suma es: 27
```

Por convenio, cuando en los enunciados mostremos el contenido de la pantalla tras la ejecución de un programa, el texto que aparezca en negrita y subrayado no lo habrá escrito el programa, sino que será el eco de lo escrito por el usuario a través del teclado.

## Tarea 4

Modifica el programa anterior para siga pidiendo dos datos, pero invitando al usuario a hacerlo en una única línea:

Escriba dos enteros separados por un espacio: 26 32  
Su suma es: 58

Prueba a «ser mala persona» al introducir los datos y comprueba cuándo consigues que el programa funcione de acuerdo con las especificaciones, cuándo no y cuándo el programa termina generando un error. Prueba introduciendo datos numéricos reales (con decimales) y datos no numéricos, como texto. Prueba también añadiendo espacios en blanco extras o añadiendo líneas en blanco a discreción.

## Tarea 5

El objetivo de esta tarea y de la siguiente es practicar y aprender a presentar resultados después de conocer qué manipuladores para la escritura en la pantalla hay disponibles en C++ y cuál es el efecto de cada uno de ellos. Estos manipuladores se presentan en la última sección de este guion de prácticas (sección 1.5). Se recomienda consultar la documentación sobre ellos disponible en el manual de referencia de la biblioteca predefinida en C++, accesible desde la página web de la asignatura. En este caso, te bastará con utilizar adecuadamente el manipulador `setw`.

Trabajaremos primero con el código del fichero «`potencias.cpp`». Este programa adolece de un grave defecto: *la presentación de resultados por pantalla es penosa*. Compruébalo ejecutándolo y modifícalo después hasta que la salida que muestre sea como la que se muestra a continuación:

x	x^2	x^3
===	===	===
1	1	1
2	4	8
3	9	27
4	16	64
5	25	125
6	36	216
7	49	343
8	64	512
9	81	729
10	100	1000
11	121	1331
12	144	1728
13	169	2197
14	196	2744
15	225	3375
16	256	4096
17	289	4913
18	324	5832
19	361	6859
20	400	8000
21	441	9261
22	484	10648
23	529	12167
24	576	13824

**Tarea 6**

Vamos a trabajar ahora con el código del fichero «angulos.cpp». De nuevo, la presentación de los datos en la pantalla es muy mejorable. Modifica el programa hasta que la salida que muestre sea como la que se muestra a continuación:

Grados	Radianes	Seno	Coseno
=====	=====	=====	=====
0	0.0000	0.0000	1.0000
10	0.1745	0.1736	0.9848
20	0.3491	0.3420	0.9397
30	0.5236	0.5000	0.8660
40	0.6981	0.6428	0.7660
50	0.8727	0.7660	0.6428
60	1.0472	0.8660	0.5000
70	1.2217	0.9397	0.3420
80	1.3963	0.9848	0.1736
90	1.5708	1.0000	0.0000
100	1.7453	0.9848	-0.1736
110	1.9199	0.9397	-0.3420
120	2.0944	0.8660	-0.5000
130	2.2689	0.7660	-0.6428
140	2.4435	0.6428	-0.7660
150	2.6180	0.5000	-0.8660
160	2.7925	0.3420	-0.9397
170	2.9671	0.1736	-0.9848
180	3.1416	0.0000	-1.0000

Las tareas anteriores pueden intentarse antes de acudir al laboratorio a la sesión correspondiente a esta práctica. Durante la sesión podrá completarse y, en su caso, mejorarse el trabajo previo realizado.

## 1.4. Resultados y entrega de la práctica

Como resultado de esta primera práctica, cada estudiante dispondrá en su máquina de una carpeta denominada «practical», ubicada dentro de otra denominada «programacion1». En la carpeta «practical» se localizarán los siguientes ficheros:

- Fichero «bienvenida.cpp», cuyo código fuente en C++ corresponde a la modificación solicitada en la tarea 2.
- Fichero «eco-numerico.cpp», cuyo código en C++ corresponde a la modificación solicitada en la tarea 4.
- Fichero «potencias.cpp», cuyo código en C++ corresponde a la modificación solicitada en la tarea 5.
- Fichero «angulos.cpp», cuyo código en C++ corresponde a la modificación solicitada en la tarea 6.

Deberán haberse subido a Moodle los ficheros «bienvenida.cpp», «eco-numerico.cpp», «potencias.cpp» y «angulos.cpp», con carácter general, **antes del sábado 28 de septiembre a las 18:00**.

Sin embargo, los estudiantes de los **subgrupos de prácticas 33 y 34 del grupo 413** (con prácticas en viernes A), deberán haber subido los ficheros solicitados antes del **antes del jueves 26 de septiembre a las 12:00**. Para esta entrega, se consideran estudiantes de los subgrupos de prácticas 33 y 34 a aquellos estudiantes apuntados en Moodle en dichos subgrupos y a los estudiantes del grupo 413 que no estén apuntados todavía en ningún grupo. Se toma como referencia el estado de la tarea «Grupos de prácticas» de Moodle en el momento de publicar este guion.

## 1.5. Manipuladores para dar formato a los datos de salida

Un *manipulador C++* es una función que se aplica al flujo de datos enviados hacia un dispositivo de salida (pantalla o fichero) para dar formato a los datos que el programa presenta por él o que se aplica al flujo de datos procedente de un dispositivo de entrada (teclado o fichero) para dar formato a los datos que llegan al programa desde él.

La utilización de manipuladores va a ser necesaria en los programas solicitados en las tareas 5 y 6 de esta práctica.

Un resumen de algunos de los manipuladores de uso más común se presenta a continuación. En las tareas mencionadas anteriormente, no se va a requerir la utilización de todos ellos, sino solo un pequeño subconjunto. De todas formas, se incluyen en este resumen todo ellos para que os sirvan como referencia en el futuro.

En el resumen aparecen agrupados por funcionalidad, aunque su utilización requiere la inclusión de dos bibliotecas distintas que no se corresponde con esta agrupación. La mayor parte de estos manipuladores están definidos en la biblioteca <iostream>, **pero** tres de ellos (setw, setprecision y setfill), requieren hacer uso de la biblioteca <iomanip>.

- Relativos a la gestión de líneas y del búfer asociado al flujo de salida:

**flush:** Cuando se escribe en un flujo de salida (como cout, la pantalla, o en un fichero), no está garantizado que lo escrito se vuelque inmediatamente en el mismo. Por motivos de eficiencia, el propio programa C++ o el sistema operativo pueden almacenar los datos que se han ido escribiendo en el flujo en un búfer intermedio que se vuelca de forma real cuando se determina que se puede hacer de forma eficiente.

El manipulador flush fuerza a que el contenido del búfer asociado al dispositivo de salida al que se envía se vuelque inmediatamente en el dispositivo. El uso concreto de este manipulador se explicará con más detalle más adelante en el curso, en los temas relativos a ficheros.

**endl:** vacía el búfer asociado al dispositivo de salida y finaliza la línea en curso.

Por ejemplo, en la instrucción

```
cout << "Bienvenidos_a_UNIZAR" << endl;
```

el manipulador endl implica que el texto Bienvenidos\_a\_UNIZAR<sup>4</sup> aparecerá en la pantalla (incluso en el caso de que se hubiera quedado almacenado en el búfer intermedio) y finaliza la línea de la pantalla en la que se ha escrito. La siguiente operación de escritura en cout se producirá a partir del inicio de la línea inmediatamente inferior.

- Manipuladores relativos a la presentación de datos de forma tabular:

**setw(*n*):** establece en *n* el número mínimo de caracteres con el que presentar el siguiente dato en el flujo. Si el dato a representar tiene una longitud en caracteres inferior a la establecida por *n*, se completará por defecto con espacios en blanco por la izquierda. En caso contrario, se presentará el dato completo, aunque utilice más de *n* caracteres.

Por ejemplo, la instrucción

```
cout << setw(5) << 43;
```

escribe en la pantalla \_ \_ \_ 43, con tres espacios a la izquierda para completar los cinco espacios establecidos en el manipulador.

En cambio, la instrucción

```
cout << setw(4) << 18243;
```

escribirá 18243, utilizando todos los caracteres necesarios para escribir el dato, incluso si el número de caracteres necesarios (5) es superior al establecido (4).

El manipulador setw se utiliza, sobre todo, para escribir datos en forma tabular a lo largo de varias líneas. Este es el único manipulador que solo afecta al siguiente dato que se va a escribir en el flujo de salida.

setw es uno de los tres manipuladores descritos en esta sección cuya **definición se encuentra en la biblioteca <iomanip>**.

**setfill(*c*):** establece el carácter *c* como carácter de relleno cuando se utiliza setw.

Por ejemplo, la instrucción

```
cout << setfill('*') << setw(5) << 43;
```

escribe en la pantalla \*\*\*43.

Como el resto de los manipuladores (excepto setw), su efecto sobre un flujo de salida perdura hasta que se vuelve a utilizar setfill. Si se quiere volver a la situación por defecto, es necesario utilizar setfill(' ').

setfill es otro de los tres manipuladores descritos en esta sección cuya **definición se encuentra en la biblioteca <iomanip>**.

<sup>4</sup>En este ejemplo y los que siguen, se utiliza el símbolo *espacio visible* ( \_ ) para visualizar y poder contar los espacios en blanco presentes en una cadena de caracteres o que se escriben en cout.

**right:** presenta los datos de forma que, cuando se utiliza en conjunción con `setw`, los caracteres que no sean de relleno se alinean a la derecha del campo. Este es el comportamiento por defecto.

Por ejemplo,

```
cout << right << setw(9) << "UNIZAR";
```

escribe en pantalla `___UNIZAR`.

**left:** presenta los datos de forma que, cuando se utiliza en conjunción con `setw`, los caracteres que no sean de relleno se alinean a la izquierda del campo. Es, por lo tanto, el manipulador complementario a `right`.

Por ejemplo,

```
cout << left << setw(9) << "UNIZAR";
```

escribe en pantalla `UNIZAR___`.

- Manipuladores relativos a la base en la que se muestran los datos enteros que se escriben en un flujo de salida o a la base de la que se leen de teclado:

**dec:** modifica la base de lectura o escrita y la establece en base decimal (base 10).

Por ejemplo, la instrucción

```
cout << dec << 43;
```

escribe en la pantalla `43`.

**oct:** modifica la base de lectura o escrita y la establece en base octal (base 8).

Por ejemplo, la instrucción

```
cout << oct << 43;
```

escribe en la pantalla `53`, que es como se escribe el número decimal 43 en base 8.

**hex:** modifica la base de lectura o escrita y la establece en base hexadecimal (base 16).

Por ejemplo, la instrucción

```
cout << hex << 43;
```

escribe en la pantalla `2b`, que es como se escribe el número decimal 43 en base 16.

El modo por defecto es el decimal. La utilización de uno de estos manipuladores cambia el estado del flujo de lectura o escritura de forma permanente hasta que se utilice otro de ellos.

- Otros manipuladores relativos a la presentación de datos numéricos:

**showpos:** se muestra el signo `+` ante valores no negativos (incluido el cero).

Por ejemplo,

```
cout << showpos << 43;
```

escribe en la pantalla `+43`.

El manipulador que revierte esta situación es `noshowpos`.

**scientific:** establece el modo de notación científica como modo de presentación de datos numéricos reales.

Por ejemplo

```
cout << scientific << 17.234;
```

escribe en la pantalla `1.723400e+001`, es decir,  $1,7234 \times 10^1$ .

**fixed:** establece el modo de notación fija como modo de presentación de datos numéricos reales.

Por ejemplo,

```
cout << fixed << 17.234;
```

escribe en la pantalla 17.234

Los manipuladores `scientific` y `fixed` son opuestos. El modo por defecto es `scientific`, aunque en este curso, en general, nos va a interesar más escribir los datos numéricos reales con notación fija. Por ello, en la gran mayoría de los programas en los que queramos presentar números reales en la pantalla, tendremos que ejecutar la instrucción

```
cout << fixed;
```

**setprecision(n):** establece la precisión con la que se mostrarán los datos numéricos reales, es decir, el número de cifras que aparecen tras el separador decimal. Su valor por defecto es 6.

Por ejemplo,

```
cout << fixed << setprecision(1) << 17.287;
```

escribe en la pantalla 17.3 mientras que la instrucción

```
cout << scientific << setprecision(2) << 17.287;
```

escribe en la pantalla 1.73e+001

`setprecision` es el tercero de los tres manipuladores descritos en esta sección cuya **definición se encuentra en la biblioteca** `<iomanip>`.

- Manipuladores relativos a la presentación de datos booleanos:

**boolalpha:** establece un modo de trabajo en que los datos lógicos o booleanos son extraídos o añadidos a un flujo como una secuencia de caracteres (**true** o **false**).

Por ejemplo,

```
cout << boolalpha << true;
```

escribe en la pantalla **true** y la instrucción,

```
cout << boolalpha << (1 == 2);
```

escribe en la pantalla **false**

**noboolalpha:** establece un modo de trabajo en que los datos lógicos o booleanos son extraídos o añadidos a un flujo como valores enteros (0 o 1).

Por ejemplo

```
cout << noboolalpha << true;
```

escribe en la pantalla 1 y la instrucción,

```
cout << noboolalpha << (1 == 2);
```

escribe en la pantalla 0.

El modo por defecto para cualquier flujo es `noboolalpha`.

Un manipulador solo afecta al flujo de entrada o de salida (`cin`, `cout`, etc.) al que se aplica. El efecto de los manipuladores permanece en el flujo de entrada o de salida correspondiente hasta que se aplica otro manipulador que lo modifica, a excepción del manipulador `setw(n)`, al que hay que invocar antes de cada dato al que se le quiere definir un ancho de campo.

Más información y algunos ejemplos ilustrativos sobre los manipuladores disponibles en las bibliotecas predefinidas en C++ se puede consultar en <https://www.cplusplus.com/reference>.