Prácticas de Programación 1

Grado en Ingeniería Informática

Escuela de
Ingeniería y Arquitectura
Universidad Zaragoza

Miguel Ángel Latre y Javier Martínez
Área de Lenguajes y Sistemas Informáticos
Departamento de Informática e Ingeniería de Sistemas

Universidad
Zaragoza

Curso 2018-19

Presentación

Las prácticas de la asignatura **Programación 1** no deben imaginarse como una sucesión de sesiones en laboratorio a las que el alumno debe acudir *a que le cuenten cosas*. Nada más lejos de la realidad; el alumno debe acudir a cada 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 alumno 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[1].

La información de este manual está organizada en seis capítulos. Cada capítulo está asociado a una práctica.

- Cada capítulo comienza con una breve descripción de los objetivos que se persiguen realizando la práctica.
- 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 alumno 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 de laboratorio asociadas a ella. El trabajo debe iniciarse una o dos semanas antes, para poder concluirlo durante la sesión en laboratorio asociada a la práctica y que pueda ser supervisado por un profesor, recibiendo las indicaciones y sugerencia que permitan, en su caso, mejorarlo.
- Cada alumno debe tener claro que el trabajo de prácticas es un **trabajo que le corresponde realizar a él mismo**, como una parte fundamental del estudio y aprendizaje de la asignatura. El profesor le puede ayudar, pero esta ayuda es inútil si él no ha trabajo previamente lo suficiente.

Este manual es una guía inicial a las prácticas de la asignatura. Cada alumno deberá acostumbrase a acudir frecuentemente y consultar las siguientes fuentes de información, a las que se puede acceder a través de **página web de la asignatura**: `http://webdiis.unizar.es/asignaturas/PROG1/`

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

[1] `https://estudios.unizar.es/estudio/asignatura?anyo_academico=2018&asignatura_id=30204&estudio_id=20180148&centro_id=110&plan_id_nk=439`

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

La siguiente tabla muestra el calendario previsto de sesiones prácticas de **Programación 1** correspondiente al curso 2018-19. Estas sesiones tienen lugar en el **`laboratorio L0.04`** situado en la planta baja del **`edificio Ada Byron`** del **`Campus Río Ebro`** de la **`Universidad de Zaragoza`**.

Calendario de sesiones prácticas de PROGRAMACIÓN 1. Curso 2017-18				
Sesiones	**Lunes A**	**Martes A**	**Lunes B**	**Martes B**
1ª	L-24-SEP	M-25-SEP	L-01-OCT	M-02-OCT
2ª	L-15-OCT	M-16-OCT	L-22-OCT	M-23-OCT
3ª	L-05-NOV	M-30-OCT	L-12-NOV	M-06-NOV
4ª	L-19-NOV	M-13-NOV	L-26-NOV	M-20-NOV
5ª	L-3-DIC	M-27-NOV	L-10-DIC	M-20-DIC
6ª	L-17-DIC	M-18-DIC	L-14-ENE	M-08-ENE

En las prácticas de **Programación 1** se desea evitar que el alumno se dedique a explorar el extenso territorio que ofrece un lenguaje como C++ y acabe perdido en él. Hemos optado porque el alumno solo utilice una parte reducida de lo que el lenguaje C++ ofrece y, eso sí, lo haga bien, dominando y comprendiendo en todo momento la tecnología que utiliza y el significado de lo que programa. La parte del lenguaje C++ que está permitido utilizar coincide esencialmente con los elementos presentados y utilizados en el curso de **Programación 1**.

A programar se aprende programando. De ahí la importancia de que cada alumno 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 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 alumno.

Zaragoza, septiembre de 2018

Miguel Ángel Latre y Javier Martínez
Departamento de Informática e Ingeniería de Sistemas
de la Universidad de Zaragoza

Práctica 1: Cómo desarrollar programas escritos 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 alumnos 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.

Los alumnos que no hayan formalizado aún su matrícula en la asignatura también deben realizar la práctica. Si a un alumno le resultara imposible realizar la práctica con su grupo de prácticas, deberá realizarla con cualquiera de los restantes grupos, sin importar si se trata de un grupo de mañana o de tarde.

El desarrollo de la sesión correspondiente a esta práctica tiene dos partes diferenciadas.

1. En la primera parte cada alumno deberá atender las explicaciones del profesor sobre los primeros apartados de este guion y seguir sus instrucciones.
2. En la segunda parte cada alumno deberá desarrollar individualmente el trabajo propuesto en el apartado *1.3 Trabajo a desarrollar en esta práctica*. Para ello conviene que, antes de acudir al laboratorio a 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 *1.2.4 Manipuladores para dar formato a los datos de salida*.

El trabajo asociado a esta práctica concluye nada más acabar la primera sesión de prácticas en laboratorio ya que al día siguiente hay que empezar a trabajar la práctica siguiente.

1.2. Tecnología y herramientas

1.2.1. Los puestos de trabajo

Cada uno de los puestos de trabajo de los laboratorios de programación 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 del Linux RHEL (*Red Hat Enterprise Linux*). Para trabajar con él hay que disponer de una cuenta de usuario. Cada alumno dispone de una cuenta de usuario abierta por el administrador del sistema, una vez ha formalizado su matrícula en los estudios. Una vez cargado CentOS, el usuario debe identificarse mediante su nombre de usuario (*username*) y su contraseña (*password*). Por ejemplo:

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

- **Windows 8.1**. Para trabajar con él se recomienda hacerlo a través de su propia cuenta de usuario. Una vez cargado el sistema, el usuario debe cerrar la sesión de invitado que se abre automáticamente,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 los dos sistemas operativos anteriores son comunicados a cada alumno de nuevo ingreso en los estudios mediante un mensaje electrónico a su cuenta de correo en la Universidad. 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 CentOS

En las prácticas de Programación 1 trabajaremos con el sistema operativo CentOS.

Tras acceder a nuestra cuenta y, tras un tiempo de espera prudencial, aparece una ventana con un asistente para realizar una configuración inicial de CentOS, que ofrece la posibilidad de elegir el idioma de la interfaz gráfica (podemos seleccionar la opción «Español»), la disposición del teclado (elegiremos también «Español»), la privacidad (autorización del acceso de algunas aplicaciones a nuestra ubicación; podemos desactivar dicha opción) y la configuración del acceso a cuentas en línea (que no nos interesa hacer en este momento).

Tras ello, aparecen en la pantalla dos iconos («*Home*» y «*Trash*») y dos menús desplegables en su parte superior («Aplicaciones» y «Lugares»[2]).

Nos interesa especialmente la carpeta «*Home*», ya que en ella podremos crear nuevas carpetas, borrarlas cuando queramos y almacenar en ellas ficheros.

[2]Hasta que no volvamos a iniciar sesión, aparecerán en inglés

Conviene saber que estas carpetas no se encuentran almacenadas en un disco local del computador personal de nuestro terminal sino 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, nuestras carpetas y ficheros podrán ser accedidos desde cualquier otro puesto de trabajo de los que dispone el Departamento de Informática e Ingeniería de Sistemas (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.

Desde el menú desplegable «Aplicaciones» 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:
  - La web `http://webdiis.unizar.es/asignaturas/PROG1/` de la asignatura Programación 1.
  - La web `http://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 `http://www.cplusplus.com/reference/`.
- El entorno de desarrollo integrado Code::Blocks (más información de este entorno en `http://www.codeblocks.org/`).

Cuando hayamos concluido nuestra sesión de trabajo deberemos cerrar las ventanas abiertas y salir de nuestra cuenta de trabajo CentOS. Hay que evitar dejarla abierta para impedir que alguien pueda hacer un uso indebido de ella. Para ello debemos seleccionar la orden apagado situada en el menú desplegable de la parte superior derecha de la pantalla.

1.2.3. Code::Blocks. Un entorno de desarrollo integrado

Un entorno de desarrollo integrado (en inglés *integrated development environment* o *IDE*) es un programa informático que integra un conjunto de herramientas que facilitan el diseño, la puesta a punto, la ejecución y el 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 Code::Blocks para desarrollar programas escritos en C++.

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 seis prácticas. La organización que tendrá la carpeta **`programacion1`** se muestra a continuación.

```
Programacion1                       // carpeta asociada a la asignatura
    Practica1                       // carpeta asociada a la 1ª práctica
        Bienvenida                  // carpeta del proyecto Bienvenida
        Circunferencias             // carpeta del proyecto Circunferencias
        Circulo                     // carpeta del proyecto Circulo
        Formatear                   // carpeta del proyecto Formatear
        practica1.workspace         // contenido del área de trabajo practica_01
        practica1.workspace.layout  // proyecto activo en el área practica_01
    Practica2                       // carpeta asociada a la 2ª práctica
        ...                         //       . . .
        practica2.workspace         // contenido del área de trabajo practica_02
        practica2.workspace.layout  // proyecto activo en el área practica_02
    ...
    Practica6                       // carpeta asociada a la 6ª práctica
        ...                         //       . . .
        practica6.workspace         // contenido del área de trabajo practica_06
        practica6.workspace.layout  // proyecto activo en el área practica_06
    datos                           // carpeta para almacenar ficheros de datos
```

Por el momento nos vamos a limitar a crear la carpeta **`programacion1`** y, dentro de ella, la carpeta **`practica1`**, por el momento vacía de contenido.

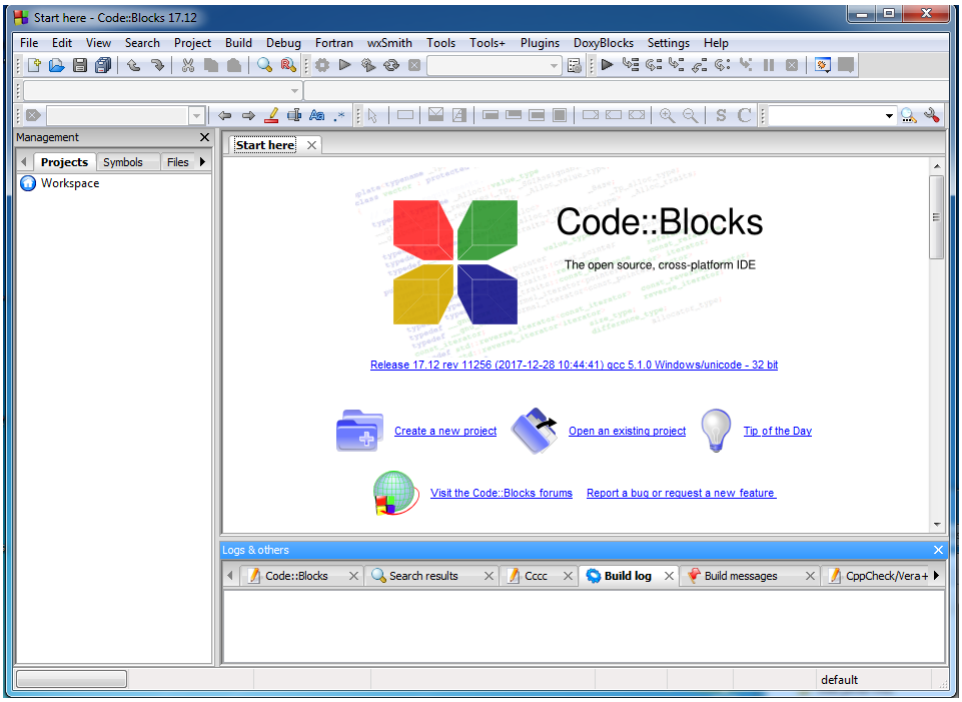
```
Programacion1
    Practica1
```

Área de trabajo o *workspace*

La primera vez que ejecutemos Code::Blocks, el entorno de programación presenta un cuadro diálogo en el que muestra el conjunto de todos los compiladores de C++ localizados por Code::Blocks en `hendrix`. Entre ellos, aparecen «GNU GCC Compiler» y «LLVM Clang Compiler». El compilador establecido por defecto es «GNU GCC Compiler», pero vamos a seleccionar «LLVM Clang Compiler» y marcarlo como compilador por defecto haciendo clic en el botón «Set as default».

También aparece una ventana con la «sugerencia del día». La primera sugerencia explica como desactivar dicha ventana, aunque puede ser útil mantenerla activa para aprender cosas nuevas del entorno Code::Blocks conforme lo vayamos utilizando.

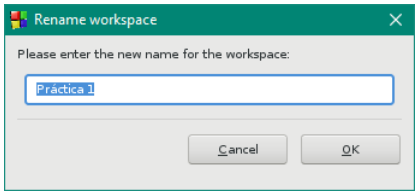
Finalmente, aparece la ventana principal de Code::Blocks, compuesta por una serie de paneles con los que iremos trabajando. Al iniciarse, el panel principal es uno de bienvenida, denominado *«Start here»*, que posibilita comenzar el trabajo en un **área de trabajo** predeterminada definida con el nombre *«Workspace»* tal como se muestra en el panel *«Management»* que aparece en la parte izquierda de la ventana.



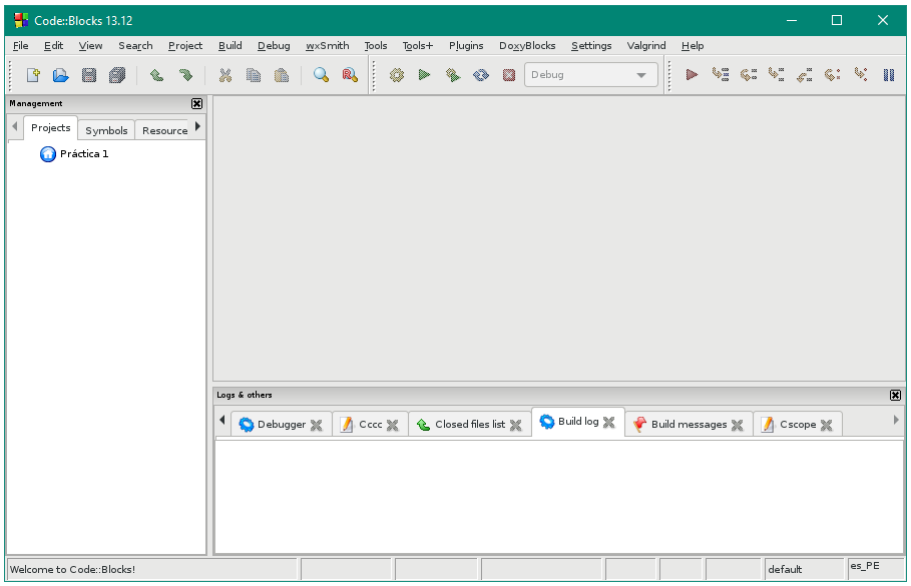
- **Creación, denominación y guardado de un área de trabajo o *workspace***

  Lo primero que vamos a hacer es definir una nueva área de trabajo, darle un nombre y guardarla en la carpeta que seleccionemos.

  Para ello vamos modificar el nombre del área de trabajo actual, cuyo nombre es *«Workspace»*, para pasar a denominarla «Práctica 1». Para lograrlo posicionamos el ratón sobre el nombre del área de trabajo, pulsamos a continuación su botón derecho, seleccionamos la opción *«Rename workspace...»*, escribimos el nuevo nombre y ejecutamos la orden de cambio de nombre.



  El resultado de la orden anterior se aprecia inmediatamente en el panel *«Management»*.



En el área de trabajo «Práctica 1» no hemos desarrollado aún ningún proyecto de programación y, por lo tanto, está vacía de proyectos. No obstante lo anterior, vamos a guardar la información que describe el área de trabajo dentro de la carpeta «practica1». Para ello ejecutamos la orden *«File»* → *«Save workspace as...»* y damos un nombre al fichero en el que se describe el contenido del área de trabajo, por ejemplo, le damos el nombre «practica1.workspace». Obsérvese que el nombre de este fichero no tiene por qué coincidir con el nombre que hemos dado al área de trabajo «Práctica 1». **En particular, al dar nombre a los ficheros que almacenan la configuración de un área de trabajo, no utilizaremos ni espacios en blanco ni caracteres acentuados.**

- **Abandono de un área de trabajo o *workspace***

  Si deseamos abandonar este área de trabajo para, por ejemplo, poder trabajar con otra área de trabajo basta con ejecutar la orden *«File»* → *«Close workspace»*

  Si hubiéramos realizado algún cambio en la definición del área de trabajo (en su nombre o en los proyectos que contiene) nos pedirá confirmación de estos cambios.

- **Recuperación de un área de trabajo o *workspace***

  Si deseamos volver trabajar con un área de trabajo ya creada, por ejemplo con el área «Práctica 1», basta con ejecutar la orden *«File»* → *«Open»* y, navegando por el sistema de carpetas y ficheros, seleccionar el fichero que describe la información de esta área. En el supuesto planteado, deberíamos seleccionar el fichero «practica1.workspace», ubicado en la carpeta «practica1».

Proyectos

Desde el IDE Code::Blocks se pueden desarrollar proyectos de programación escritos en C++ y asociarlos a una o mas áreas de trabajo.

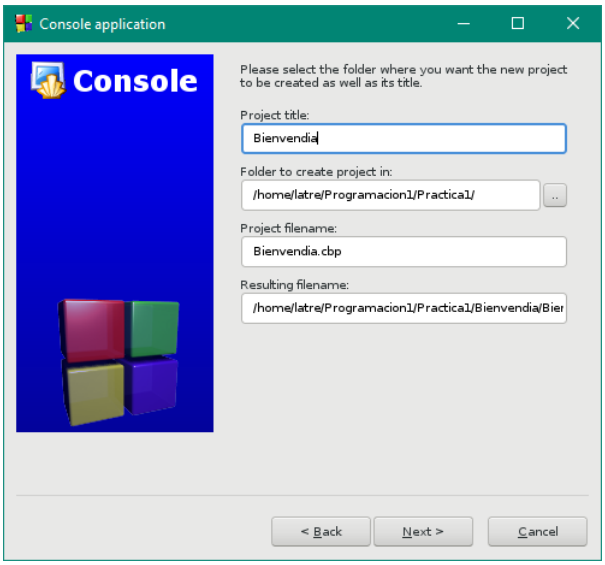
En los apartados que siguen se va a explicar cómo crear un proyecto, cómo trabajar en su desarrollo y puesta a punto y cómo gestionar los proyectos asociados a un área de trabajo.

- **Creación de un proyecto**

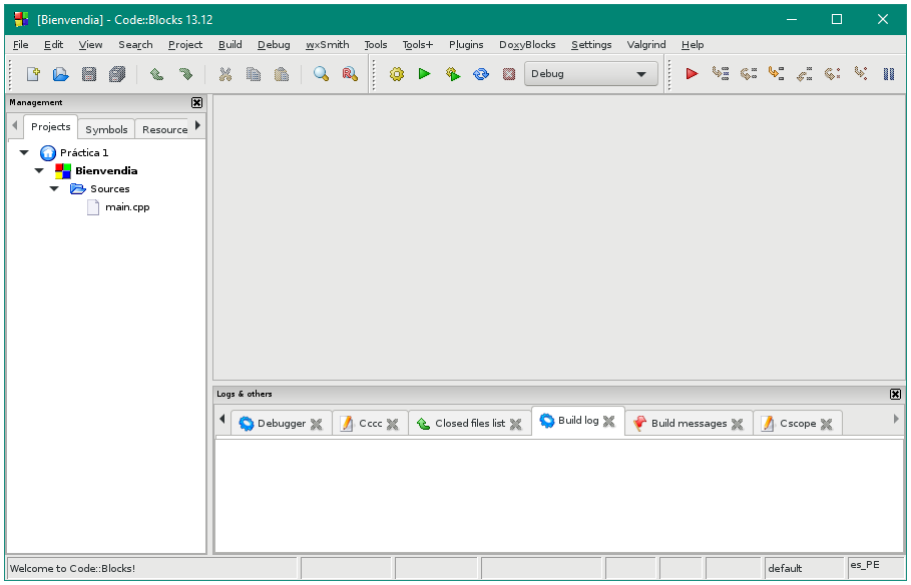
  Vamos a crear un proyecto por cada uno de los programas a desarrollar en esta primera práctica. Todos ellos los vamos a asociar al área de trabajo «Práctica 1».

  Vamos a proceder a crear el primer proyecto ejecutando la orden *«File»* → *«New»* → *«Project...»*. Seleccionaremos seguidamente *«Console application»* como tipo de proyecto a desarrollar. En la

sucesión de ventanas que se nos presenten a continuación deberemos seleccionar C++ como lenguaje de programación, deberemos seleccionar la carpeta «Practica1» como ubicación de los ficheros descriptivos del proyecto y deberemos dar un nombre al proyecto, por ejemplo, el nombre «Bienvenida».



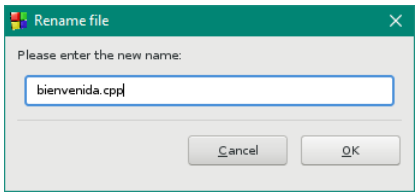
El proyecto «Bienvenida» ha sido creado y su código se limita a un único fichero que ha recibido de forma automática el nombre «main.cpp».



Si deseamos cambiar el nombre de este fichero, procedemos a cerrar la ventana que muestra su código, en el caso de que esté abierta, seleccionamos el nombre del fichero en el panel *«Management»* con el botón derecho del ratón y ejecutamos la orden *«Rename file...»* que nos permite modificar su nombre. Aunque el nombre «main.cpp» no tiene nada de malo (incluso es una buena práctica que el nombre del fichero que contiene la función `main()` sea «main.cpp»), en las prácticas que nos ocupan vamos a tener varios proyectos independientes en cada área de trabajo, por lo que es recomendable distinguir los distintos ficheros a través de su nombre.

Vamos a asignar el nombre «bienvenida.cpp» al fichero con el código de este primer proyecto[3].

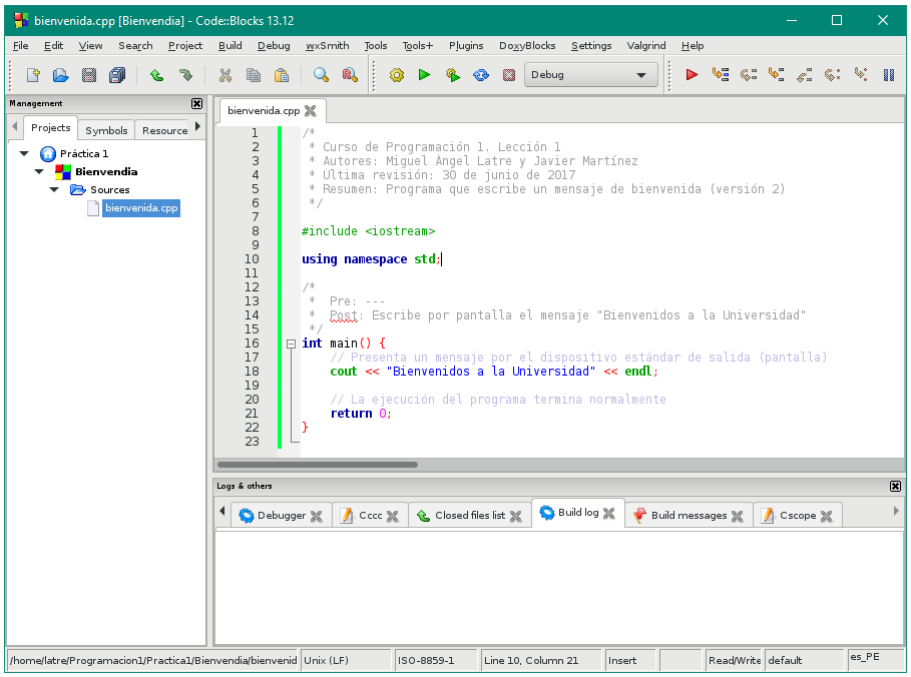
[3]La extensión «.cc» es también perfectamente válida en los ficheros que contienen código fuente en C++. El nombre «bienvenida.cc» sería también correcto.



- **Desarrollo y puesta a punto**

  En la puesta a punto de un programa escrito en C++ pueden distinguirse cuatro pasos:

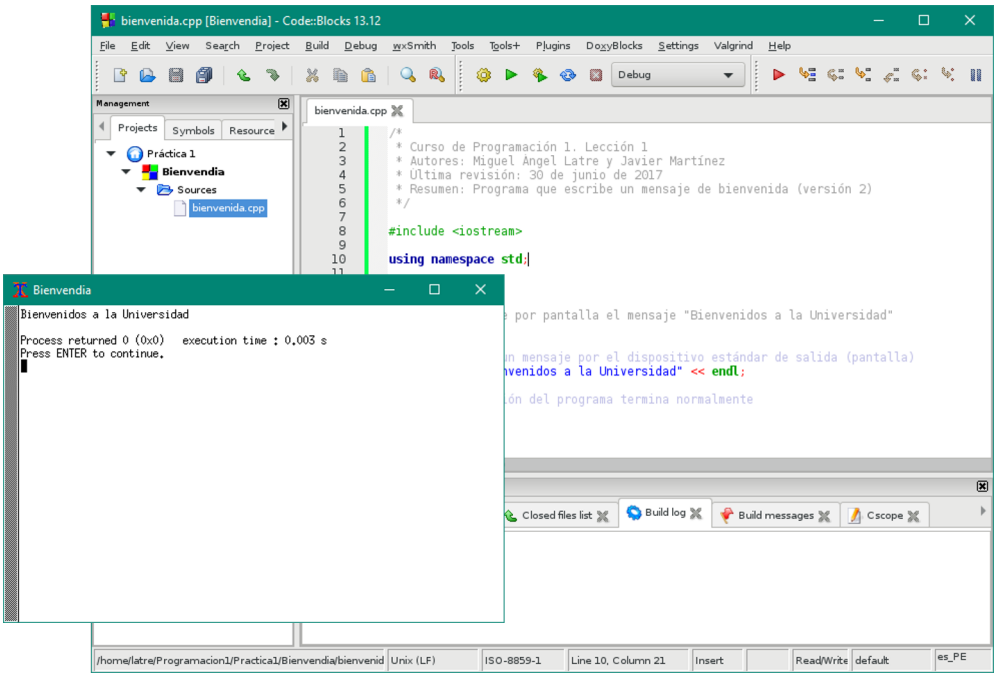
  Paso 1. Edición del código. Estamos en condiciones de editar el código de este primer proyecto, el código del fichero fuente «bienvenida.cpp». Volveremos a este paso si se detectan errores en cualquiera de los pasos que siguen.



  Paso 2. Compilación del código. A continuación podemos compilar el fichero «bienvenida.cpp» ejecutando la orden *«Build» → «Compile curent file»*. Al hacerlo, se comprueba que el código no presenta errores de compilación y, en tal caso, se genera un fichero objeto de nombre «bienvenida.o» que es almacenado en la carpeta «Bienvenida/obj/Debug». Si el compilador detectara errores en el código, habrá que proceder a su análisis y corrección antes de avanzar al siguiente paso.

  Paso 3. Construcción de un programa ejecutable . Una vez compilado el código del programa y constatada la ausencia de errores de compilación, podemos construir un programa ejecutable invocando la orden *«Build» → «Build»*. Al hacerlo, se genera un fichero ejecutable de nombre «Bienvenida» o «Bienvenida.exe», dependiendo del sistema operativo en el que se compile), que es almacenado en la carpeta «Bienvenida/bin/Debug».

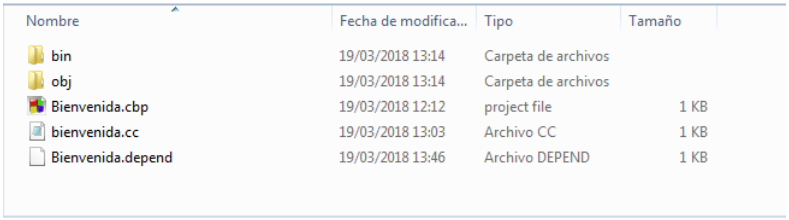
  Paso 4. Ejecución del programa . Estamos en condiciones de ejecutar el programa ejecutable «Bienvenida» directamente o desde el propio entorno Code::Blocks, ejecutando la orden *«Build» → «Run»*.



En caso de que el comportamiento del programa no responda a lo esperado habrá que revisar su diseño y repetir los pasos anteriores.

Nota adicional La secuencia integrada por los tres últimos pasos (compilación, construcción de un programa ejecutable y su ejecución final) se puede desencadenar mediante la orden *«Build» → «Build and run»*.

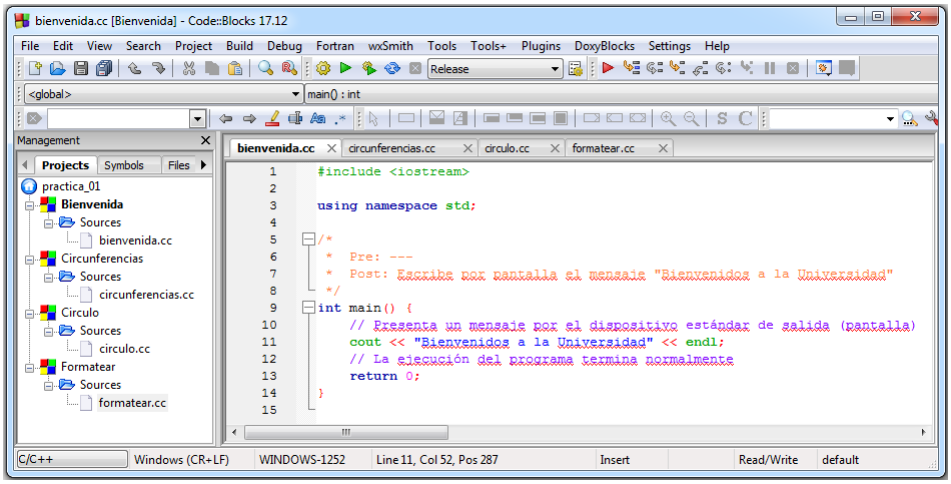
Los ficheros y carpetas asociados a un proyecto se pueden observar en la siguiente figura. Todos ellos se almacenan en la carpeta «practica1/Bienvenida». El fichero «Bienvenida.cbp» almacena la información descriptiva del proyecto y sus opciones asociadas. El fichero «Bienvenida.depend» almacena información sobre la ubicación de los ficheros de código que integran el proyecto. La carpeta «obj» almacena los ficheros objeto intermedios resultado de compilar los diferentes módulos de que consta el código del proyecto. Finalmente, la carpeta «bin» almacena el fichero ejecutable resultante del desarrollo del programa.



Nombre	Fecha de modifica...	Tipo	Tamaño
bin	19/03/2018 13:14	Carpeta de archivos	
obj	19/03/2018 13:14	Carpeta de archivos	
Bienvenida.cbp	19/03/2018 12:12	project file	1 KB
bienvenida.cc	19/03/2018 13:03	Archivo CC	1 KB
Bienvenida.depend	19/03/2018 13:46	Archivo DEPEND	1 KB

- **Gestión de los proyectos de un área de trabajo**

  Al finalizar esta práctica, el área de trabajo «Práctica 1» debe tener asociados cuatros proyectos («Bienvenida», «Circunferencias», «Circulo» y «Formatear»). Desde ella se podrá seleccionar cualquiera de dichos proyectos y trabajar en su desarrollo y puesta a punto (edición del código, compilación, construcción de un ejecutable y su ejecución).



Una vez creada un área de trabajo puede interesarnos modificarla. Alguno de los cambios que nos puede interesar realizar en un área de trabajo se comentan a continuación.

- **Selección del proyecto activo**. Solo se puede trabajar con el provecto activo en cada momento (su nombre se destaca con letra negrita en el panel *«Management»*). Seleccionar como proyecto activo otro distinto es muy sencillo: basta situar sobre su nombre el ratón y hacer doble clic con su botón izquierdo o hacer clic con el botón derecho y seleccionar la opción *«Activate project»* del menú contextual.
- **Cambio en el orden de su lista de proyectos**. Se pueden reordenar los proyectos de un área de trabajo ejecutando la orden *«Project»* → *«Project tree»* → *«Move project up»* para permutar un proyecto con el que le precede dentro del área de trabajo o la orden *«Project»* → *«Project tree»* → *«Move project down»* para permutarlo con el que le sigue.
- **Añadir un proyecto de nueva creación al área de trabajo**. Un proyecto de nueva creación se puede añadir al área de trabajo ejecutando la orden *«File»* → *«New»* → *«Project...»*, tal como se ha hecho anteriormente con el proyecto «Bienvenida».
- **Añadir un proyecto ya existente al área de trabajo**. Un proyecto ya existente se puede añadir al área de trabajo ejecutando la orden *«File»* → *«Open»* y seleccionando en el sistemas de carpetas el fichero descriptivo del proyecto a añadir (estos ficheros tienen como sufijo «.cbp». Conviene aclarar que un proyecto puede estar asociado a una o a mas áreas de trabajo sin ningún problema.
- **Eliminar un proyecto del área de trabajo**. Un proyecto de un área de trabajo se puede eliminar de ella seleccionando el proyecto a eliminar como proyecto activo y ejecutando a continuación la orden *«File»* → *«Close project»*. La eliminación de un proyecto de un área de trabajo no significa su eliminación ni la eliminación de sus ficheros. El proyecto podrá seguir asociado a otras áreas de trabajo y sus ficheros permanecerán intactos.

1.2.4. Manipuladores para dar formato a los datos de salida

La utilización de manipuladores va a ser necesaria en los primeros programas que se diseñen en las prácticas de esta asignatura.

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.

En la primera lección de la asignatura se ha hecho uso de diversos manipuladores en los programas C++ que se han presentado. Conviene hacer una lectura atenta de ella antes de abordar el trabajo propuesto en esta práctica.

Un resumen de algunos de los manipuladores de uso más común se presenta a continuación:

- Manipuladores definidos en la biblioteca `<iostream>`, concretamente en la biblioteca `<ios>` declarada en ella:

  `endl`: vacía el búfer asociado a un dispositivo de salida presentando su contenido y finalizando la línea en curso.

  `flush`: vacía el búfer asociado a un dispositivo de salida presentando su contenido por pantalla sin finalizar la línea en curso.

  `dec`: convierte los datos numéricos a base decimal (base 10).

  `oct`: convierte los datos numéricos a base octal (base 8).

  `hex`: convierte los datos numéricos a base hexadecimal (base 16).

  `left`: presenta los datos de forma que los caracteres que no sean de relleno se alinean a la izquierda del campo.

  `right`: presenta los datos de forma que los caracteres que no sean de relleno se alinean a la derecha del campo.

  `internal`: presenta los datos numéricos de forma que el signo y los caracteres indicativos de la base están alineados a la izquierda del campo y las cifras significativas a la derecha.

  `showpos`: se muestra el signo `+` en los valores positivos.

  `scientific`: establece un modo de trabajo de forma que los datos reales son presentados en notación científica (ej. `1.7234e+01`).

  `fixed`: presenta los datos reales en notación normal de coma flotante (ej. `17.234`).

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

  `noboolalpha`: establece un modo de trabajo de forma que los datos lógicos o booleanos son extraídos de un flujo o añadidos a un flujo como valores enteros: `1` en caso de valores lógicos **`true`** o `0` en caso de valores lógicos **`false`**.

- Manipuladores definidos en la biblioteca `<iomanip>`

  `setw(`*n*`)`: establece la anchura mínima de campo de los datos a presentar por un dispositivo de salida, es decir el número mínimo de caracteres *n* a presentar completando el dato, en su caso, con el carácter de relleno definido en ese momento (ej. `setw(10)`).

  `setfill(`*ch*`)`: establece *ch* como carácter de relleno (ej. `setfill('*')`).

  `setprecision(`*n*`)`: establece el número *n* de cifras significativas de los datos numéricos o, en su caso, el número de cifras decimales; su valor por defecto es 6 (ej. `setprecision(2)`).

  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)` que hay que invocarlo 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 predefinas en C++ se puede consultar en `http://www.cplusplus.com/reference`.

1.3. Trabajo a desarrollar en esta práctica

Alguno de los fragmentos de código que se mencionan a continuación pueden ser descargados desde la sección de ***Materiales docentes*** de la web de esta asignatura: `http://webdiis.unizar.es/asignaturas/PROG1/` a través del enlace ***Código C++ y datos***.

Se propone que cada alumno realice las tareas que se describen en los apartados siguientes.

- **Tarea 1**. Se debe crear, si no se ha hecho ya, una carpeta (directorio) denominada «Programacion1» y, dentro de ella, otra carpeta (directorio) denominada «Practica1».

  A continuación se definirá un área de trabajo denominada «Práctica 1». El nombre del fichero descriptivos de dicha área llevará por nombre «practica1.workspace» y se almacenarán en la carpeta «practica1».

  Se debe crear un primer proyecto el área de trabajo «Práctica 1», el proyecto «Bienvenida», con el programa comentado en el apartado anterior. Debe compilarse su código y, si no se han producido errores, debe ser ejecutado para observar su comportamiento.

  Modificar el código del programa, provocando algún error en él. Compilarlo y observar cómo el compilador informa de los errores detectados para facilitar al programador su trabajo de puesta a punto del programa.

  Corregir los errores del programa, volver a compilarlo, a construir un programa ejecutable y a ejecutarlo hasta que el comportamiento del programa sea el deseado.

- **Tarea 2**. Deben crearse dos nuevos proyectos dentro del área de trabajo «Práctica 1», los proyectos «Circunferencias» y «Circulo», para el desarrollo de los dos programas restantes descritos en el capítulo 1 del texto de la asignatura. Deben compilarse y, cuando no haya errores de compilación, deben ejecutarse ambos programas hasta tener la convicción de que sus comportamientos son los esperados.

- **Tarea 3**. Debe crearse un nuevo proyecto cuyo nombre sea «Formatear», dentro del área de trabajo «Práctica 1». El código inicial del programa a desarrollar en él se muestra a continuación.

```
#include <iostream>
#include <iomanip>
#include <cmath>

using namespace std;

/*
 * Pre:  desde <= hasta
 * Post: Presenta por pantalla una tabla con una línea por cada uno de los valores
 *       del intervalo [desde,hasta]. En cada línea se muestra un valor, su
 *       cuadrado y su cubo
 */
void mostrarPotencias (int desde, int hasta) {
    cout  << "x" << "x^2" << "x^3" << endl;
    for (int x = desde; x <= hasta; x = x + 1) {
        cout << x << x*x << x*x*x << endl;
    }
}

...
```

```
...

/*
 * Pre:  desde <= hasta
 * Post: Presenta por pantalla una tabla con una línea por cada uno de los valores
 *       de los ángulos, expresados en grados: desde, desde + 5, desde + 10,
 *       etc., que sean inferiores o iguales a hasta. En cada línea se muestra el
 *       valor del ángulo en grados y en radianes y los valores de sus funciones
 *       seno y coseno.
 */
void mostrarAngulos (int desde, int hasta) {
    const double PI = 3.1415926;
    cout  << "grados"  << "radianes" <<  "seno" <<  "coseno" << endl;
    for (double x = desde; x <= hasta; x = x + 5.0) {
        double radianes = PI * x / 180.0;
        cout << x << radianes << sin(radianes) << cos(radianes) << endl;
    }
}


/*
 *  Pre: ---
 *  Post: Presenta por pantalla una tabla de potencias (en cada línea un valor,
 *        su cuadrado y su cubo) y una tabla de ángulos (en cada línea un ángulo
 *        en grados, en radianes,su seno y su coseno)
 */
int main() {
    // Presenta la tabla de potencias de los enteros comprendidos entre 5 y 15
    mostrarPotencias(5,15);
    // Presenta una tabla con los ángulos comprendidos entre 0 y 180 grados,
    // su valor en radianes y los valores de sus funciones seno y coseno
    mostrarAngulos(0,180);
    // Concluye normalmente
    return 0;
}
```

Este programa adolece de un grave defecto: la presentación de resultados por pantalla es penosa.

El trabajo a realizar por cada alumno consiste en modificar el diseño de las funciones `mostrarPotencias(desde,hasta)` y `mostrarAngulos(desde,hasta)` para que, al ser invocadas, presenten de forma digna y legible sus resultados. Ello se puede lograr haciendo uso de algunos de los *manipuladores* o *funciones de formato* presentados en el apartado anterior en las instrucciones de escritura de resultados (`cout << ...`).

Se recomienda intentar diferentes presentaciones jugando con varios elementos: la anchura de las columnas de datos de la tabla, la alineación a izquierda o derecha de los datos de cada línea, el número de decimales de los datos representados por números reales, etc.

El objetivo de esta tarea es practicar y aprender a presentar resultados después de conocer qué manipuladores hay disponibles en C++ y cuál es el efecto de cada uno de ellos. 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.

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.

Cada alumno debe haber completado las tareas definidas en esta primera sesión de prácticas en la propia sesión o, en caso necesario, uno o dos días después, como máximo, ya que debe comenzar cuanto antes a desarrollar el trabajo asociado a la segunda práctica.

Como resultado de esta primera práctica, cada alumno dispondrá en su cuenta de `hendrix` de una carpeta denominada «Programacion1» dentro de la cual se localizarán los siguientes proyectos de programación en C++ organizados en las áreas de trabajo y con las denominaciones que se detallan a continuación:

1. Área de trabajo «Práctica 1» con los siguientes proyectos ubicados en la carpeta «Programacion1/Practica1»:
    - Proyecto C++ «Bienvenida»
    - Proyecto C++ «Circunferencias»
    - Proyecto C++ «Circulo»
    - Proyecto C++ «Formatea»