Primeros pasos en C++

Antonio Garrido / Javier Martínez Baena

Dpto. Ciencias de la Computación e Inteligencia Artificial
E.T.S. de Ingenierías Informática y de Telecomunicación
Universidad de Granada

Fundamentos de Programación

Grado en Ingeniería Informática

Índice de contenido

1.Objetivos....3
2.Primeros pasos en C++....4
2.1.Los laboratorios de la ETSIIT de la Universidad de Granada....4
2.2.El programa “Hola Mundo”....4
2.2.1.Archivos en disco....7
2.3.Compilación y ejecución del programa....8
2.4.Consola y ficheros del proyecto....9
3.Errores de compilación....10
3.1.Estructura de un programa....10
3.2.Compilación....10
3.2.1.Introducción a la E/S....11
3.3.Ejemplo de error de compilación....12
3.3.1.Lista de múltiples errores....13
3.3.2.Avisos....14
4.Referencias....14

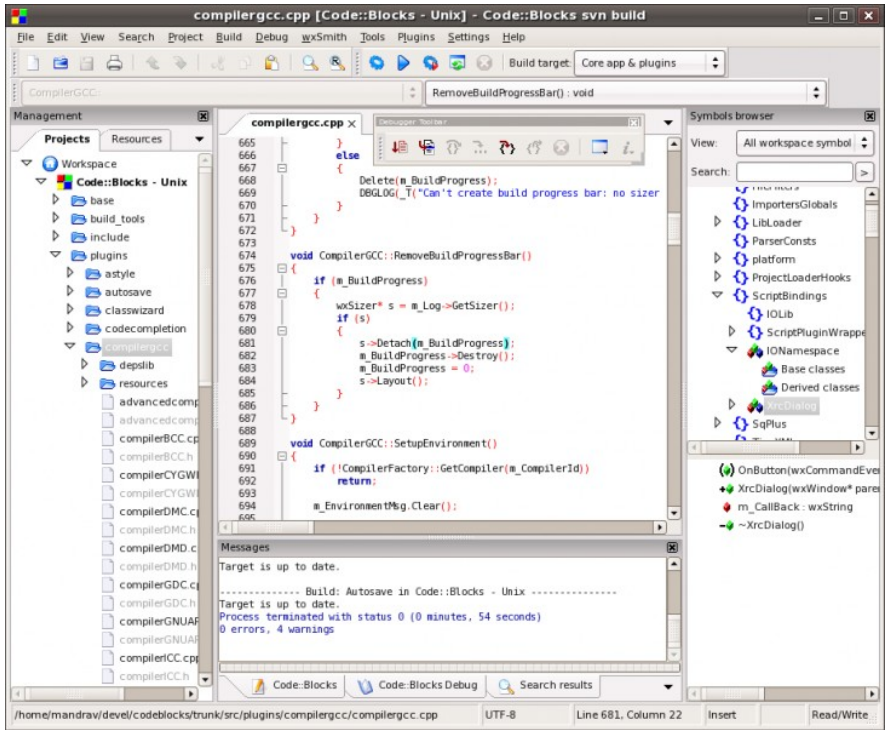
1. Objetivos

En esta práctica se introduce el uso de un "Entorno de desarrollo integrado" (en inglés, "*Integrated Development Environment*" o "*IDE*") para el desarrollo de programas en C++, junto con los primeros ejemplos de programas simples en este lenguaje.

Recordemos que un *IDE* es un programa especialmente diseñado para facilitar el desarrollo de software a los programadores. Algunos de los módulos que forman parte de él son:

- Un editor de código fuente. Es un editor de texto que nos permite escribir uno o varios ficheros con el programa que se está desarrollando.
- Un compilador. El conjunto de programas necesarios para transformar el programa fuente en un programa ejecutable.
- Herramientas para la generación automática de los ejecutables. Programas para facilitar el manejo de múltiples fuentes y recursos para formar el programa final.
- Depurador. Permite trazar (ejecutar paso a paso) el programa fuente y estudiar su comportamiento para detectar y corregir errores.

Concretamente, en este documento se realiza una introducción a *Code::Blocks,* que será el que usemos durante el curso para desarrollar programas en C++. En la siguiente figura podemos ver una captura de pantalla de este *IDE,* donde podemos observar que está compuesta de múltiples subventanas, correspondientes a distintos aspectos del proyecto que se está desarrollando.



Finalmente, cabe indicar que, aunque éste es un entorno especialmente adecuado para esta asignatura, el objetivo de la misma es aprender programación haciendo uso del lenguaje C++.

Por tanto, si se desea usar otro entorno, como "Microsoft Visual Studio" o el obsoleto "Dev-C++", la asignatura puede desarrollarse sin ningún problema. Tenga en cuenta que la mayor parte del tiempo y del trabajo se centrará en aprender a programar en C++.

2. Primeros pasos en C++

En esta sección se va a introducir al alumno en los primeros programas en C++. Para ello, se indicará la forma en que se pueden usar los laboratorios de la ETSIIT, y se presentarán algunos ejemplos y ejercicios que el alumno puede desarrollar.

2.1. Los laboratorios de la ETSIIT de la Universidad de Granada

Lógicamente, si el alumno está usando este guión en su ordenador personal, no será necesario que tenga en cuenta esta sección y puede pasar directamente a la siguiente.

Para poder usar los ordenadores de los laboratorios deberá tener una cuenta de usuario en el sistema. El acceso a un ordenador se hace indicando tres datos:

1. Nombre de usuario.
2. Password.
3. Código.

Si el usuario tiene cuenta en el sistema, deberá conocer los dos primeros. Para el tercer campo deberá escribir las cuatro letras *fpcb* (las iniciales de "*Fundamentos-Programación-Code-Blocks"*). Este código se introduce al sistema para que el ordenador arranque con una instalación de *Microsoft Windows XP* que incluye el entorno de programación "*Code::Blocks*".

Cada vez que se arranca de esta forma, se realiza una instalación completa y limpia del sistema en el disco duro. Por ello, el contenido previo de la unidad C: quedará eliminado. En otras palabras, todo lo que almacenamos en la unidad C: se perderá al reiniciar el ordenador.

Para que el alumno pueda mantener sus datos y haya un control sobre el espacio que ocupa, los servicios de informática ofrecen un espacio de disco que está disponible cuando se arranca este u otro sistema en las aulas. Este espacio de disco no está físicamente en el aula donde trabajamos, sino que se incorpora a través de la red desde los servidores de la escuela.

En el sistema que acabamos de arrancar, se ha incorporado la unidad "U:" para poder acceder a este espacio personal. Todo lo que se almacene en esa unidad no se perderá, y estará disponible en cada arranque, incluso con otros sistemas operativos.

Es recomendable, por tanto, crear una carpeta (por ejemplo, *u:\fp*) que se use para almacenar toda la información relacionada con la asignatura. Tenga en cuenta que en esta carpeta se almacenarán los proyectos de programación que se desarrollen en la asignatura. Por ello, se recomienda que los nombres de esta carpeta, así como otras subcarpetas no tengan caracteres especiales, es decir, caracteres con acentos, la letra 'ñ', espacios en blanco, etc.

Esta restricción sobre los nombres de carpetas y archivos se debe especialmente a la naturaleza multiplataforma del entorno de programación, donde no se han incluido algunas particularidades de codificación del sistema *Microsoft Windows.*

2.2. El programa "Hola Mundo"

Para comenzar, vamos a crear nuestro primer programa: simplemente escribirá el mensaje "*Hola Mundo*". Para ello, lanzaremos el entorno de trabajo y seleccionaremos la opción para crear un nuevo proyecto ("*Create a new project*").

En la ilustración 1 podemos ver el aspecto del entorno, donde hemos indicado el enlace donde seleccionar la creación de un nuevo proyecto. Es interesante observar que se han reorganizado las distintas barras de herramientas. Como puede ver, el usuario puede organizar el espacio de trabajo a su gusto.

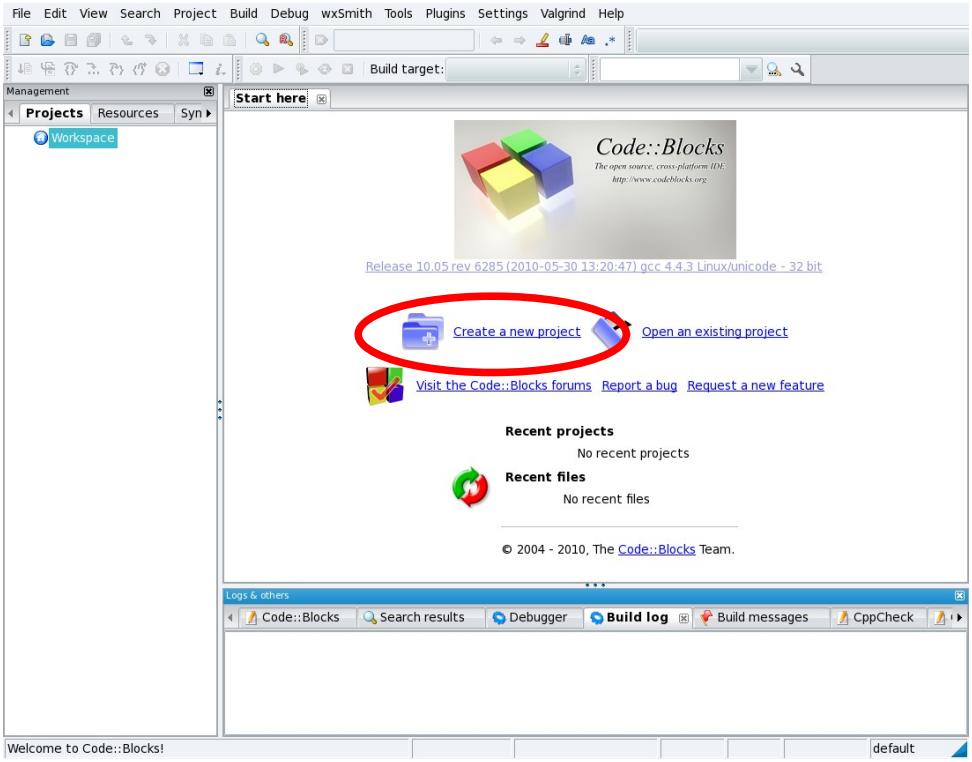
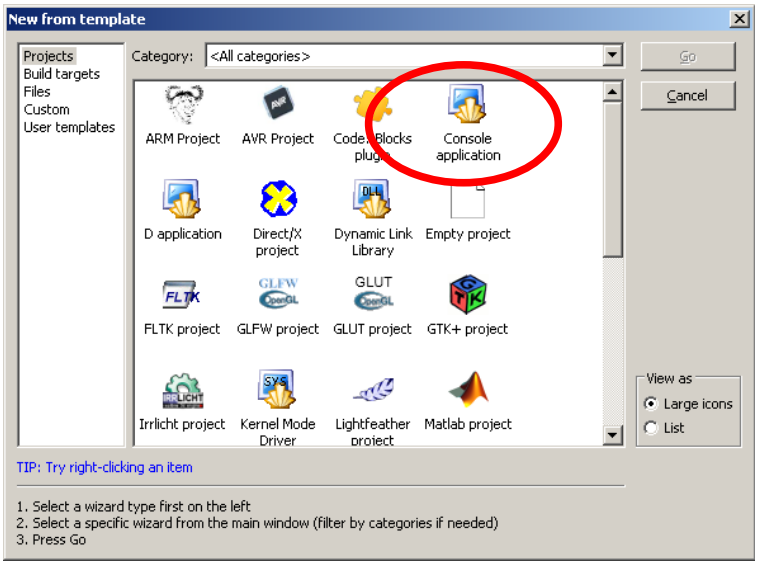


Ilustración 1: Comienzo de un nuevo proyecto

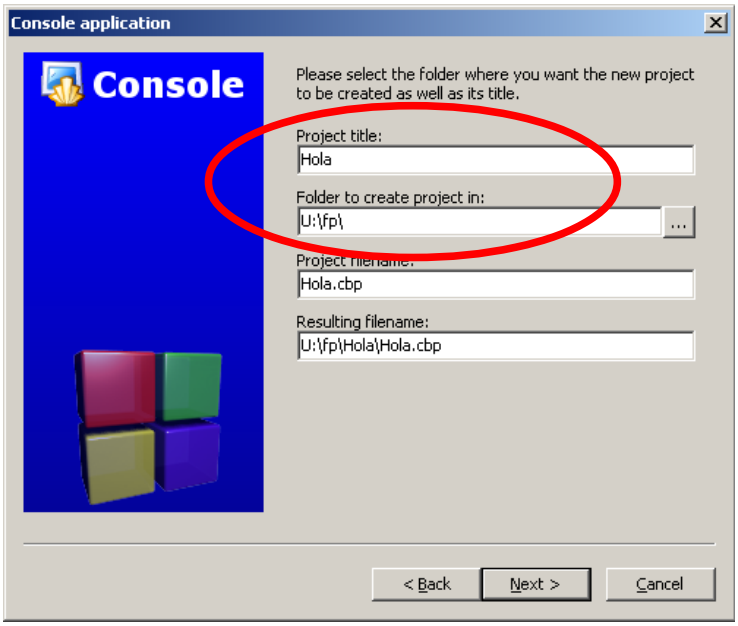
A continuación, seleccionamos el tipo de proyecto que vamos a crear de forma que Code::Blocks genera todos los recursos necesarios dependiendo de lo que vamos a necesitar. En nuestro caso, que vamos a crear aplicaciones muy simples con un nivel de entrada/salida (E/S) muy básico, seleccionaremos siempre el perfil "aplicación para consola" ("*Console application*").



Una vez seleccionado este tipo de aplicación, tendremos que indicar que el proyecto será en C++, y algunos parámetros:

1. *Project title*: El nombre del proyecto. Debería ser un nombre suficientemente descriptivo para reconocer lo que hace el programa.
2. *Folder to create project in*. Directorio donde se guardará la carpeta con todos los archivos del proyecto. Recuerde que habíamos recomendado crear una carpeta especialmente para la asignatura.

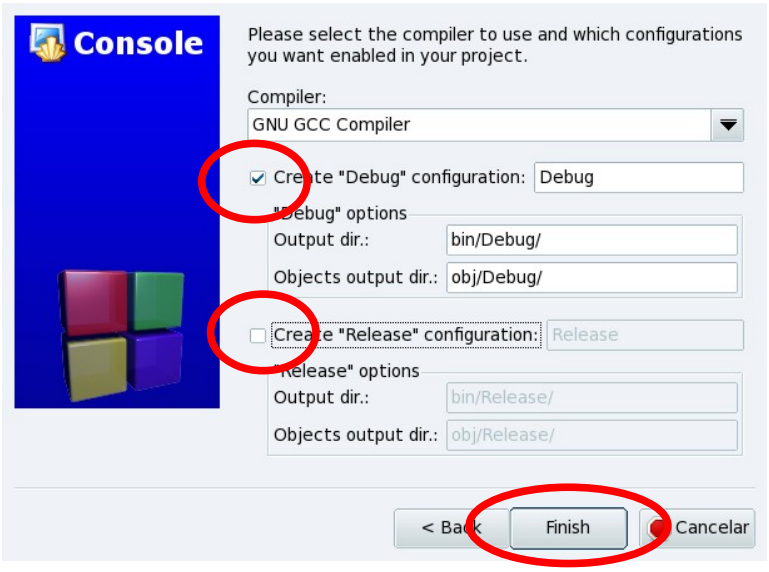
Recuerde que para estos dos parámetros, recomendamos que los nombres no tengan caracteres especiales (acentos, 'ñ',) o espacio. El resto de campos se crearán automáticamente desde estos dos valores, por lo que podemos pulsar el botón “Next>” para ir al siguiente paso.



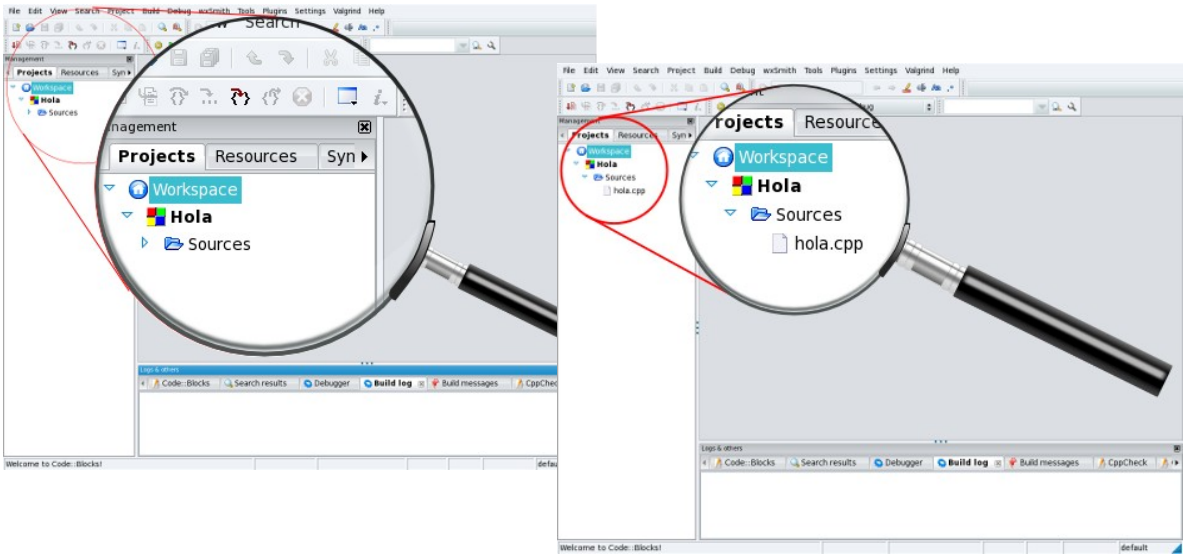
Finalmente, obtendremos una última ventana para seleccionar las configuraciones asociadas a este proyecto. Podemos seleccionar dos configuraciones:

1. *Debug*. Indica una configuración cuyo objetivo es generar los resultados de forma que se pueda depurar. Cuando se está desarrollando una aplicación, es muy probable que sea necesario localizar y reparar errores estudiando el comportamiento del programa paso a paso. Para ello, es necesario que los programas generados contengan información acerca de la relación del código ejecutable con el código fuente.
2. *Release*. Una vez que el programa se da por terminado, y los errores conocidos quedan resueltos, podemos generar los ejecutables de forma que no contengan la información de depurado e, incluso, se optimicen para un comportamiento más eficiente tanto en tiempo como en espacio.

En nuestro caso, podemos dejar seleccionada la opción “*Debug*” y pulsar el botón “*Finish*” para terminar de crear el nuevo proyecto. Más adelante volveremos a hablar de estas configuraciones.



Una vez que hemos aceptado pulsando “Finish”, volvemos a la ventana principal del entorno, donde podemos ver que hemos generado un nuevo proyecto, con nombre “*Hola*”. Observe que de ese proyecto “cuelga” un directorio “Sources” (fuentes), donde se encontrarán los ficheros en C++ que iremos editando y/o insertando posteriormente. Se ha generado un fichero automáticamente (“*main.cpp*”) que podemos renombrar pulsando con el botón derecho y seleccionando “*rename*”, para que tenga un nombre más ilustrativo (“*hola.cpp*”).



Si pinchamos dos veces sobre ese fichero, el entorno abre un editor para que podamos modificarlo. Podemos observar que ya se ha generado automáticamente un programa simple que, en nuestro caso, coincide con la aplicación que queremos hacer. Concretamente, el programa es el siguiente:

```
#include <iostream>

using namespace std;

int main()
{
  cout << "Hello world!" << endl;
  return 0;
}
```

2.2.1. Archivos en disco

Una vez generado el proyecto y cambiado el nombre del archivo C++ a “*hola.cpp*”, podemos consultar los efectos que ha tenido en el disco.

Supongamos que el directorio donde estamos guardando los proyectos es “*proyectos*”. En la figura de la derecha se presenta el esquema de archivos y directorios que se han generado automáticamente.

Como vemos, se ha creado un nuevo directorio “*Hola*” para gestionar el proyecto actual, y dentro de este directorio los archivos asociados. Observe que se incluye el archivo fuente que estamos editando, así como un archivo adicional que el programa *Code::Blocks* usa para gestionar el proyecto.

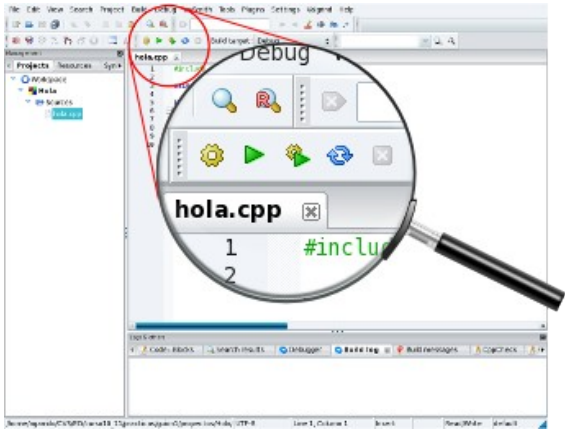
2.3. Compilación y ejecución del programa

Este programa generado está terminado y listo para traducir y ejecutar. Corresponde al programa "Hola mundo", típico como primer ejercicio en el estudio de un lenguaje de programación, y que además nos puede servir para confirmar el correcto funcionamiento de nuestro entorno de programación.

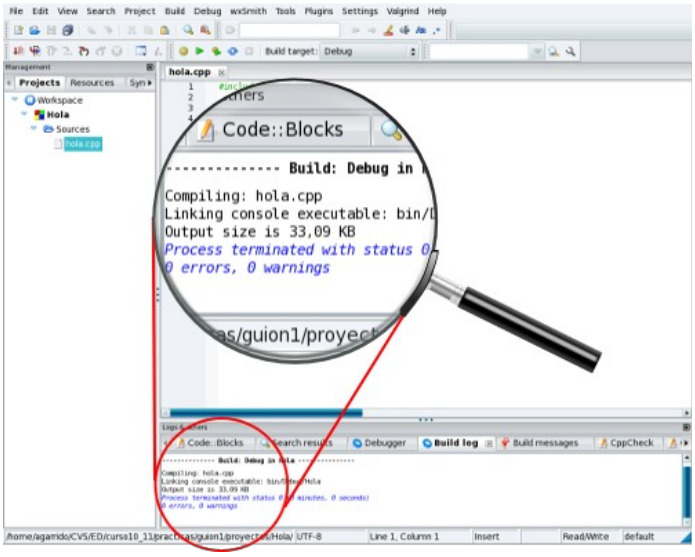
El efecto de ejecutar este programa es, simplemente, el de imprimir el mensaje "Hola mundo". Dado que el programa está terminado, es el momento de generar el ejecutable a partir de los fuentes.

Para ello, podemos usar la barra de herramientas para construcción (*build*) y ejecución (*run*) de programas. Tenemos distintas posibilidades:

1. *Build*. Permite generar el ejecutable. Se comprueban los cambios realizados y, si es necesario, crea de nuevo el ejecutable.
2. *Run*. Permite lanzar el ejecutable.
3. *Build and run.* Las dos anteriores en un solo paso.
4. *Rebuild*. Generar de nuevo todo el proyecto. Independientemente de si se han realizado cambios o no, vuelve a obtener el ejecutable desde los fuentes.
5. *Abort*. Detiene la ejecución de un programa.



Para probarlo, vamos a generar el ejecutable pulsando el botón "*Build*" (la rueda dentada). El resultado es que el entorno lanza el compilador con el archivo "*hola.cpp*", obteniendo un resultado como el que se muestra en la siguiente figura.

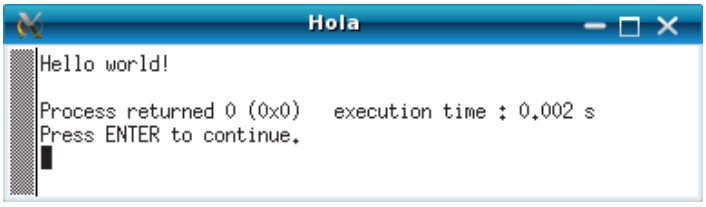


En esta figura hemos resaltado cómo en la pestaña "*Build log*" aparecen los mensajes que informan sobre el resultado de la construcción del ejecutable.

Podemos ver que indica que se lleva a cabo la compilación (*Compiling*), y el enlazado (*linking*).

Finalmente nos informa sobre el tamaño del resultado así como el resumen. Cuando un proceso devuelve un cero como resultado ("*Process terminated with status 0*") indica que no ha habido errores. Además, podemos observar que ha habido 0 errores (*errors*) y 0 avisos (*warnings*)

Si desea probar el programa, puede pulsar el triángulo verde, lo que hace que el sistema lance el ejecutable en una consola. En ella podrá observar que se imprime el mensaje "*Hello world!*".

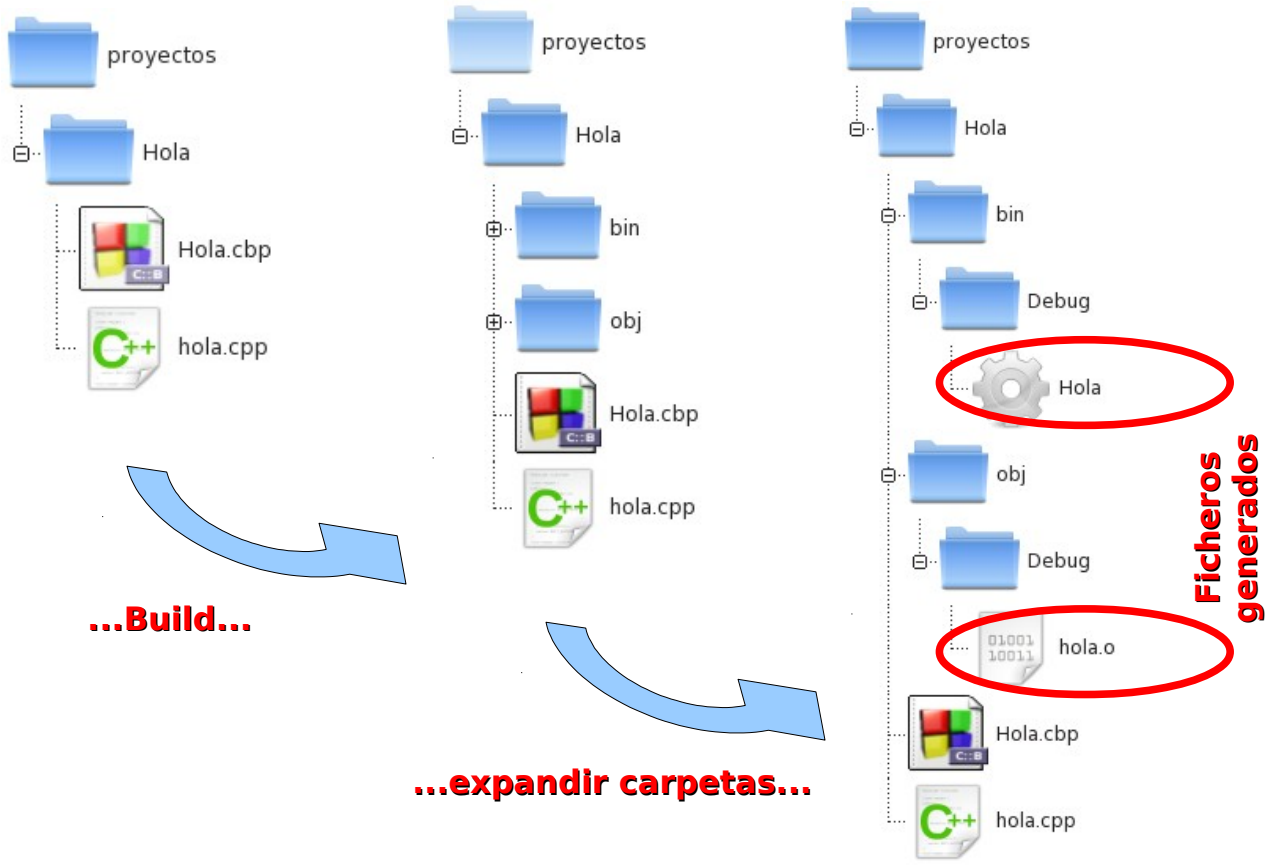


Al finalizar el programa, el entorno añade un mensaje indicando el resultado de su ejecución ("*Process returned 0*") y el tiempo que ha tardado. El resultado es un valor entero, que si vale cero, indica que la ejecución ha terminado con éxito. Cualquier otro valor indica un error de ejecución.

Recuerde la instrucción "*return*" que incorpora el entorno automáticamente. Esta instrucción es la que indica el valor a devolver. En caso de no ponerla, el compilador la añade al final de "*main*" automáticamente. Más adelante volveremos sobre ella.

2.4. Consola y ficheros del proyecto

Si volvemos a navegar sobre el sistema de archivos, y comprobamos los directorios y ficheros que se han generado, observamos una situación similar a la que muestra la siguiente figura:



Como puede comprobar se han generado dos nuevas carpetas:

1. Carpeta "*bin*". En este directorio se encontrará el ejecutable correspondiente a nuestro programa. En *Microsoft Windows*, este fichero tendrá una extensión ".exe". Por tanto, este fichero es el resultado final de la traducción.
2. Carpeta "*obj*". En este directorio se encuentran los archivos "*objeto*". Cuando se traduce un programa fuente C++, primero se traducen los archivos "*cpp*" a objeto (compilación) y finalmente se obtienen los ejecutables uniendo estos archivos objeto con los recursos necesarios para obtener un programa final (enlazado).

Por otro lado, se han generado los archivos en una carpeta "*Debug*", ya que al crear el proyecto, hemos indicado que vamos a trabajar sobre esta configuración. Más adelante estudiaremos más detenidamente la creación de diferentes configuraciones y sus efectos.

> ***Ejercicio** 1: Ejecución independiente del entorno de desarrollo*
>
> Una vez generado el ejecutable, ejecútelo tanto desde el explorador de archivos (doble click) como desde la consola (símbolo del sistema). ¿Qué diferencia observa?

3. Errores de compilación

En esta sección vamos a estudiar la estructura de un programa simple, algunos detalles sobre su contenido, y cómo el compilador los traduce detectando los posibles errores.

3.1. Estructura de un programa

En el programa de ejemplo que acabamos de presentar, con el que mostramos un mensaje, podemos observar tres partes diferenciadas:

1. Directiva "*#include*", que indica al sistema que se incluyen los recursos que el lenguaje C++ ofrece en el fichero "*iostream*". En este caso, este fichero es necesario para poder usar "*cout*" y "*endl*". A estos archivos ("*iostream*" y otros) los denominaremos "*de cabecera*".
2. Instrucción "using", indicando al sistema que usaremos "*std*". Así podemos usar directamente los identificadores que ofrece el estándar de C++. La incluiremos en todos nuestros programas para facilitar la escritura.
3. Módulo "*main*", donde aparecen las distintas instrucciones, separadas con ";", que componen el programa que desarrollamos.

En los primeros temas del curso, pondremos especial atención a este bloque "*main*", ya que podemos decir que la ejecución de nuestro programa consiste en la ejecución secuencial de cada una de las líneas que componen este módulo.

3.2. Compilación

Realizamos algunas modificaciones sobre el programa "*hola.cpp*" que hemos mostrado anteriormente para motivar algunos detalles más sobre el funcionamiento del compilador. El siguiente programa es equivalente al anterior, ya que el resultado de su ejecución es idéntico:

```
// Programa hola.cpp
// Escribe un saludo
// ------------ Comienzo sección includes
#include <iostream>    // Permite usar cout y endl

// Instrucción using para usar el estándar
using namespace std;

int main()  {
    // <-- Aquí comienzan las instrucciones del programa

    cout << "Hello world!" // El mensaje
         << endl;          // Salto de línea

    /* La última instrucción, que es opcional */  return 0;
}
```

Observe que en este listado:

- Hemos introducido líneas que empiezan con "//", lo que indica que son comentarios que el compilador debe ignorar hasta el final de la línea. Se introducen para documentar el código fuente, aclarando detalles a los posibles interesados en entender el código (ya sea el propio autor u otros programadores).
- La línea que comienza por *cout* la hemos dividido en dos partes. El programa se compone de una secuencia de "*tokens*" que pueden estar separados por espacios, tabuladores y saltos de línea. Para el compilador, la separación de esos "*tokens*" es igualmente válida independientemente del número de separadores y su disposición.
- La última instrucción "*return*" es opcional. Hemos incluido otro comentario, que como no queremos que alcance el final de línea, lo hemos realizado con el par "/*" y "*/".

El proceso de traducción que realiza el compilador se puede esquematizar como:

1. Se eliminan los comentarios. Realmente no se traducen, ya que el proceso de traducción propiamente dicho recibe un programa sin comentarios.
2. Se recorre el programa, desde la primera línea a la última, extrayendo todos y cada uno de los *tokens* que lo componen[1]. Podríamos decir que el compilador divide todo el código en una secuencia de palabras, cada una de ellas independiente. Las palabras se pueden separar con espacios, tabuladores, saltos de línea, etc. Al compilador le es indiferente el que dos "palabras" consecutivas estén diferenciadas por un único separador, por varios, o incluso el tipo de separador. Por ello, el programa anterior es equivalente al original.
3. Se analiza la sintaxis de la secuencia de *tokens*. Al igual que en castellano no sería sintácticamente correcto que un artículo apareciera dos veces seguidas en una frase, el compilador analiza[2] si la secuencia de *tokens* responde a la sintaxis del lenguaje. Por ejemplo, después de "*main*" el compilador espera que siga el *token* "(".
4. Finalmente, es necesario convertir esa secuencia en el código objeto, es decir, código traducido. Para ello, deberá determinar el significado de las sentencias que estamos analizando y sintetizar el resultado de la traducción[3].

Los detalles de esta traducción son bastante más complicados para el nivel en el que estamos, sin embargo, es importante que el alumno entienda que nuestro programa no es más que una secuencia de *tokens* que se deben analizar para comprobar que son correctos, y así poder obtener el resultado de la traducción.

3.2.1. Introducción a la E/S

Un programa recibe un conjunto de datos de entrada y obtiene unos datos de salida. Para ilustrar esta idea, así como para mostrar cómo se realizan estas operaciones mediante la consola, vamos a estudiar un ejemplo en el que se incluyen entradas y salidas. Por ejemplo, podemos escribir un programa para calcular la suma de dos valores introducidos por teclado:

```
#include <iostream>  // cout, cin, endl
using namespace std;

int main()
{
   double dato1;
   double dato2;

   cout << "Introduzca el dato 1: ";
   cin >> dato1;
   cout << "Introduzca el dato 2: ";
   cin >> dato2;

   double suma;
   suma= dato1+dato2;

   cout << "Resultado de la suma de "
        << dato1 << " con " << dato2 << ": " << suma << endl;
}
```

Si ejecuta este programa en su entorno, comprobará que el entorno lanza una consola de ejecución, solicita que se introduzcan dos datos numéricos, y escribe el resultado de la suma.

---

1 *Técnicamente, la extracción de la secuencia de tokens se denomina análisis léxico.*

2 *Técnicamente hablamos de análisis sintáctico.*

3 *Técnicamente hablamos de análisis semántico y síntesis.*

***Ejercicio** 2: Prueba de E/S de datos en consola*

Cree un nuevo proyecto (Suma) y compruebe el correcto funcionamiento de este programa.

Los detalles sobre este programa se estudiarán en teoría y en los siguientes guiones de prácticas. Aún así, es interesante indicar:

- Hemos puesto un comentario a la primera línea. Realmente, hemos indicado el motivo por el que se incluye la línea "*include*", ya que es necesaria para usar esos tres identificadores en nuestro programa.
- Se usan tres datos. Le hemos dado un nombre a cada uno (*dato1, dato2, y suma*). Cada uno de ellos aparece en una línea precedido de "*double*", que indica al programa que serán tres datos numéricos.
- Las líneas con "*cin*" se usan para indicar las entradas. Observe que simplemente le tenemos que decir el nombre del dato que estamos leyendo.
- Las líneas con "*cout*" se usan para indicar las salidas. Observe que el resultado que se obtiene en la salida corresponde al encadenamiento de cada uno de los datos que se le van indicando con "<<".

***Ejercicio** 3: Media aritmética de 3 números*

Intente modificar el anterior programa para leer 3 números y escribir su media aritmética.

Finalmente, es importante indicar que a este nivel del curso supondremos que los datos que le introducimos son correctos. Es decir, que si espera que le demos dos números, se van a introducir dos números, evitando errores al encontrarse con datos inesperados. Más adelante se estudiará con detalle por qué se generan y cómo podemos resolverlo.

***Ejercicio** 4: Prueba de entrada de datos inesperados*

Compruebe el comportamiento del programa de suma cuando se introduce como primer dato una letra.

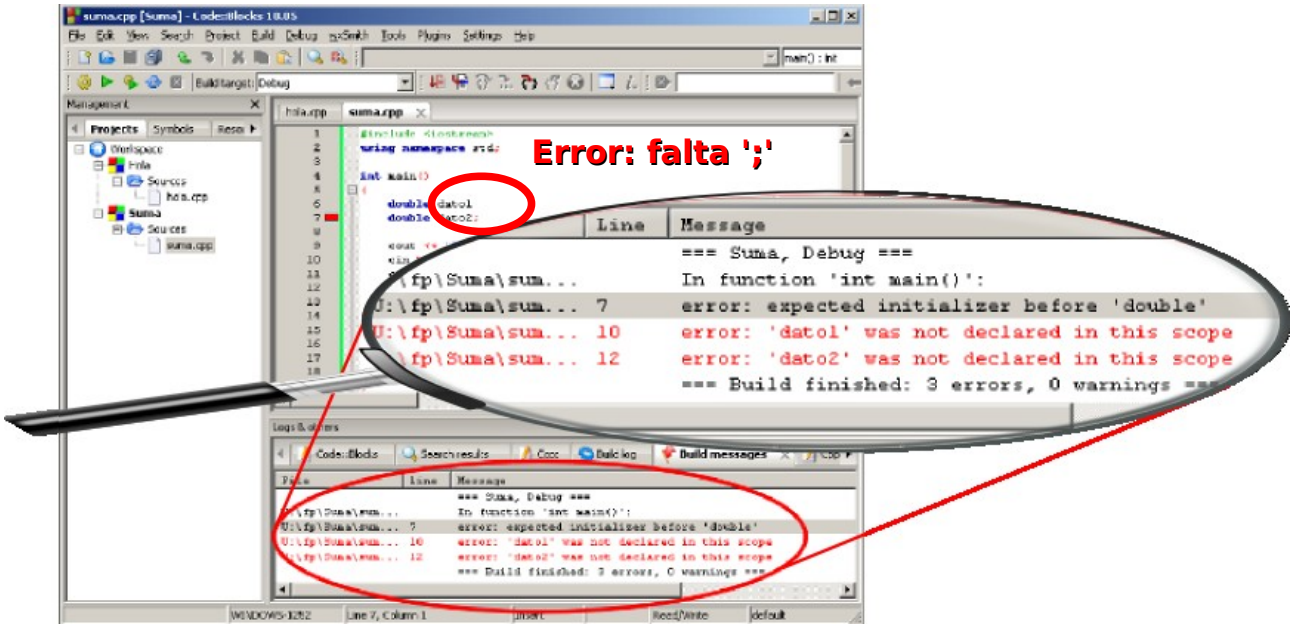
3.3. Ejemplo de error de compilación

Los ejemplos que hemos usado hasta ahora han sido programas sin errores. Para mostrar el comportamiento de Code::Blocks cuando el compilador encuentra errores, vamos a modificar el ejemplo anterior -de suma de dos números- introduciendo un error de compilación. Concretamente, vamos a eliminar el carácter ";" de la primera línea del programa.

***Ejercicio** 5: Error de compilación en el programa de suma*

Elimine el carácter punto y coma de la primera línea del programa de suma (donde aparece por primera vez *dato1*) y compruebe el resultado de compilar (pulsando "*Build*").

El efecto de la compilación, después de pulsar el botón de la rueda dentada, se muestra en la siguiente figura. Es interesante que observe que el entorno que le presentamos en esta figura contiene dos proyectos abiertos. El primero el proyecto "*Hola*", y el actual, el proyecto "*Suma*". Como puede ver, el proyecto activo se presenta en negrita, que es el que se compila cuando pulsamos el botón "*Build*".



Observe que el error se obtiene en la línea 7, aunque podemos decir que está en la línea 6. Recuerde que el compilador separa el programa en *tokens*, y por tanto, el carácter ";" podría estar en la línea 7. El error, realmente, se ha detectado en la palabra *double* ya que ahí es donde se ha encontrado con un *token* inesperado.

Por otro lado, si lee el mensaje de error, notará que hace referencia a un inicializador; más adelante estudiaremos en detalle en qué consiste esta inicialización. Como puede ver, si añadimos un simple punto y coma al final de la línea 6, todo funciona perfectamente.

3.3.1. Lista de múltiples errores

En el ejemplo anterior el compilador ha mostrado tres errores en este programa, a pesar de haber cometido sólo uno. ¿Qué ha ocurrido?

Un aspecto importante que debemos tener en cuenta cuando compilemos es que, normalmente, cuando éste intenta realizar una traducción, no se detiene en el primer error que localiza, sino que intenta saltárselo y avanzar en el resto del programa fuente con la intención de generar un informe lo más completo posible.

El objetivo final de este comportamiento se debe a que, si tenemos que realizar una compilación por cada uno de los errores, es posible que necesitáramos un número muy alto de compilaciones y, por consiguiente, una pérdida de tiempo importante. Por tanto, el compilador intenta revisar todo el código con el fin de obtener una lista con todos los errores. Una vez que tenemos todos los errores, los podemos revisar y corregir sin necesidad de volver a compilar.

Sin embargo, la situación no es tan simple, ya que los trozos de código con error se ignoran para poder continuar la compilación y, por tanto, es probable que muchos errores sean consecuencia de los anteriores. Por ejemplo, como el compilador no ha sabido entender las líneas 6 y 7, las ignora, y por tanto genera un error en todas las líneas que necesiten conocer que vamos a usar los nombres "dato1" y "dato2" como datos numéricos de nuestro programa. Por consiguiente, cuando hemos revisado y corregido los primeros, es recomendable volver a compilar para generar una nueva lista en lugar de perder tiempo revisando errores que pueden ser consecuencia de los anteriores.

En nuestros ejemplos, al ser tan simples, no cuesta demasiado volver a pulsar el botón para compilar y regenerar la lista de errores muy rápidamente. Cuando tenga proyectos más grandes, con un número de errores más alto, deberá corregir los primeros y decidir en qué momento es necesario volver a compilar.

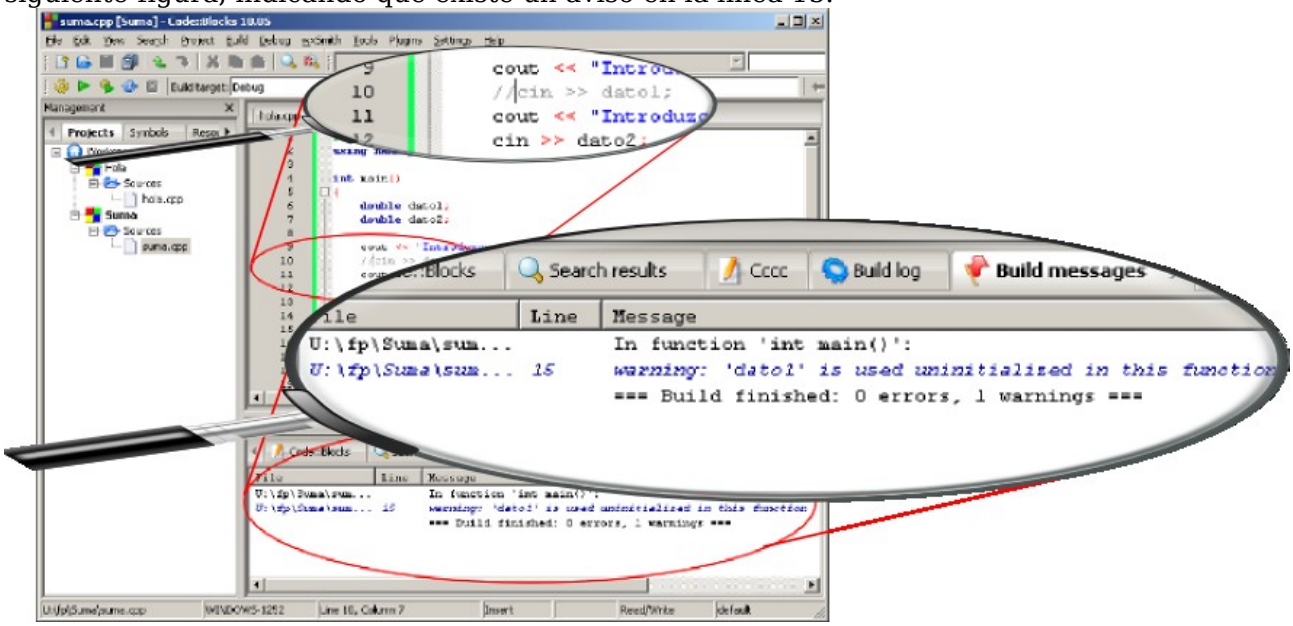
Por consiguiente, no se sorprenda si obtiene una lista muy larga de errores, corrija los primeros y vuelva a compilar.

3.3.2. Avisos

Un recurso disponible y que nos permite evitar muchos errores son los avisos ("*Warnings*"). Estos avisos no son errores en el sentido de que el código se puede traducir correctamente, aunque pueden provocar fallos en la ejecución del programa. Básicamente, consisten en avisos del compilador sobre un punto del código fuente que podría no corresponder a lo que el programador desea.

Por ejemplo, supongamos que modificamos el código fuente del programa de cálculo de la suma de dos números. Concretamente eliminamos la instrucción de lectura del valor de "dato*1*". Una forma muy simple de hacerlo es anteponer una doble barra de comentario antes de la palabra "*cin*" (véase la lupa pequeña en la figura). El programa resultante es equivalente a no haber escrito esa línea, y aunque sabemos que es incorrecto, se puede traducir sin problemas para obtener un ejecutable.

Si pulsa el botón para compilar, podrá observar un resultado como el que se presenta en la siguiente figura, indicando que existe un aviso en la línea 15.



Nos está avisando de que es posible que el programa sea incorrecto, ya que aunque se puede compilar y obtener el archivo ejecutable, hay un trozo de código "sospechoso". Efectivamente, es un error ejecutar el programa, ya que realizamos una suma entre dos valores, uno leído, y el otro indeterminado.

A pesar de todo, el programa es un programa C++ correcto sintácticamente y, por tanto, se puede traducir y ejecutar. Los avisos son simplemente mensajes para que el programador confirme que, efectivamente, es lo que desea. Algunos de ellos se podrán ignorar, porque no conducen a ningún error.

Nuestro consejo es que se revisen y se eliminen **por completo** modificando el código fuente. Además, los compiladores suelen contener opciones para "activar" la generación de nuevos mensajes de aviso. Resulta recomendable activarlos para evitar algunos errores en tiempo de ejecución.

4. Problemas adicionales

Escriba programas para resolver los siguientes problemas:

1. Escriba un programa que lea el valor de un ángulo en grados y escriba en la salida estándar el número de radianes equivalente.
2. Escriba un programa que lea los dos valores a,b que determinan la ecuación ax+b=0 y escriba la solución de dicha ecuación. Pruebe la ejecución con valores que incluyan el cero, e incluso cuando los dos valores valen cero.