CAPÍTULO 1

LENGUAJE C.

Presentamos en este capítulo una primera vista de la programación en lenguaje C. El objetivo ahora es mostrar los conceptos básicos de un entorno de programación, y redactar, con el entorno que cada uno quiera (a lo largo del curso emplearemos fundamentalmente el Turbo C++, de la casa Borland), un primer programa en C, que nos servirá para conocer las partes principales de un programa.

Introducción.

Los lenguajes de programación están especialmente diseñados para programar computadoras. Sus características fundamentales son:

1. Son **independientes de la arquitectura física del ordenador**. Los lenguajes están, además, normalizados, de forma que queda garantizada la portabilidad de los programas escritos en esos lenguajes.

2. Normalmente **un mandato** en un lenguaje de alto nivel da lugar, al ser introducido, a **varias instrucciones** en lenguaje máquina.
3. Utilizan **notaciones cercanas a las habituales**, con sentencias y frases semejantes al lenguaje matemático o al lenguaje natural.

El lenguaje C se diseñó en 1969. El lenguaje, su sintaxis y su semántica, así como el primer compilador de C fueron diseñados y creados por Dennis M. Ritchie y Ken Thompson, en los laboratorios Bell. Más tarde, en 1983, se definió el estándar **ANSI C** (que es el que aquí presentaremos).

El lenguaje C tiene muy pocas reglas sintácticas, sencillas de aprender. Su léxico es muy reducido: tan solo **32 palabras**.

A menudo se le llama **lenguaje de medio nivel**, más próximo al código máquina que muchos lenguajes de más alto nivel. Es un lenguaje apreciado en la comunidad científica por su probada eficiencia. Es el lenguaje de programación más popular para crear software de sistemas, aunque también se utiliza para implementar aplicaciones. Permite el uso del lenguaje ensamblador en partes del código, trabaja a nivel de bit, y permite modificar los datos con operadores que manipulan bit a bit la información. También se puede acceder a las diferentes posiciones de memoria conociendo su dirección.

El lenguaje C es un lenguaje del paradigma imperativo, estructurado. Permite con facilidad la programación **modular**, creando unidades que pueden compilarse de forma independiente, que pueden posteriormente enlazarse. Así, se crean funciones o procedimientos, que se pueden compilar y almacenar, creando bibliotecas de código ya editado y compilado que resuelve distintas operaciones. Cada programador puede diseñar sus propias bibliotecas, que simplifican luego considerablemente el trabajo futuro. El ANSI C posee una amplia colección de bibliotecas de funciones estándar y normalizadas.

Entorno de programación.

Para realizar la tarea de escribir el código de una aplicación en un determinado lenguaje, y poder luego compilar y obtener un programa que realiza la tarea planteada, se dispone de lo que se denomina un entorno de programación.

Un **entorno de programación** es un conjunto de programas necesarios para construir, a su vez, otros programas. Un entorno de programación incluye **editores**, **compiladores**, **archivos para incluir**, **archivos de biblioteca**, **enlazadores** y **depuradores** (ya veremos todos estos conceptos en el primer Capítulo de este manual). Gracias a Dios existen entornos de programación integrados, de forma que en una sola aplicación quedan reunidos todos estos programas. Ejemplos de entornos integrados de programación en C son el programa Microsoft Visual C++, o el Turbo C++ de Borland.

Un **editor** es un programa que permite construir ficheros de caracteres, que el programador introduce a través del teclado. Un programa no es más que archivo de texto. El programa editado en el lenguaje de programación se llama **fichero fuente**. Algunos de los editores facilitan el correcto empleo de un determinado lenguaje de programación, y advierten de inmediato la inserción de una palabra clave, o de la presencia de un error sintáctico, marcando el texto de distintas formas.

Un **compilador** es un programa que compila, es decir, genera **ficheros objeto** que “entiende” el ordenador. Un archivo objeto todavía no es una archivo ejecutable.

El entorno ofrece también al programador un conjunto de archivos para incluir o **archivos de cabecera**. Esos archivos suelen incluir abundantes parámetros que hacen referencia a diferentes características de la máquina sobre la que se está trabajando. Así, el mismo programa en lenguaje de alto nivel, compilado en máquinas diferentes, logra archivos ejecutables distintos. Es decir, el mismo código fuente es así

portable y válido para máquinas diferentes.

Otros archivos son los **archivos de biblioteca**. Son programas previamente compilados que realizan funciones específicas. Suele suceder que muy diversos programas tienen idénticas o muy parecidas muchas partes del código. Ciertas partes que son ya conocidas porque son comunes a la mayor parte de los programas están ya escritas y vienen recogidas y agrupadas en archivos que llamamos **bibliotecas**. Ejemplos de estas funciones son muchas matemáticas (trigonométricas, o numéricas,…) o funciones de entrada de datos desde teclado o de salida de la información del programa por pantalla. Desde luego, para hacer uso de una función predefinida, es necesario conocer su existencia y tener localizada la biblioteca donde está precompilada; eso es parte del aprendizaje de un lenguaje de programación, aunque también se disponen de grandes índices de funciones, de fácil acceso para su consulta.

Al compilar un programa generamos un archivo objeto. Habitualmente los programas que compilemos harán uso de algunas funciones de biblioteca; en ese caso, el archivo objeto no es aún un fichero ejecutable, puesto que le falta añadir el código de esas funciones. Un entorno de programación que tenga definidas bibliotecas necesitará también un **enlazador** que realice la tarea de "juntar" el archivo objeto con las bibliotecas empleadas y llegar, así, al **código ejecutable**.

La creación e implementación de un programa no suele terminar con este último paso descrito. Con frecuencia se encontrarán **errores**, bien **de compilación** porque haya algún error sintáctico o de expresión y manejo del lenguaje; bien **de ejecución**, porque el programa no haga exactamente lo que se deseaba. No siempre es sencillo encontrar los errores de nuestros programas; un buen entorno de programación ofrece al programador algunas herramientas llamadas **depuradores**, que facilitan esta tarea.

Podríamos escribir el algoritmo que define la tarea de crear un

programa. Ese algoritmo podría tener el aspecto del recogido en el flujograma de la Figura 1.1.

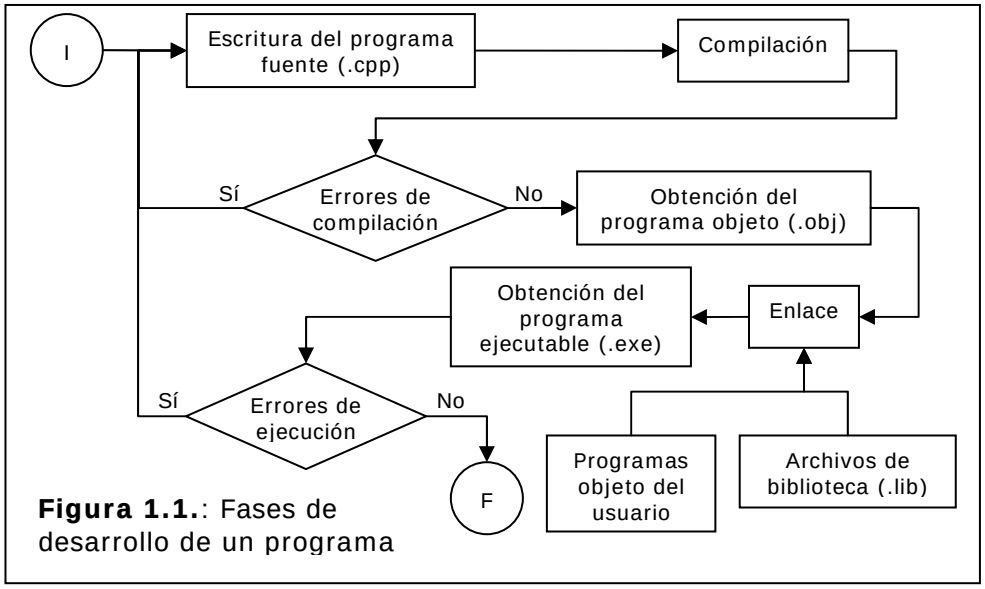

Figura 1.1.: Fases de desarrollo de un programa

En el caso del lenguaje C, el archivo de texto donde se almacena el código tendrá un *nombre* (el que se quiera) y la extensión **.cpp** (si trabajamos con un entorno de programación de C++), o **.c**. Al compilar el fichero fuente (*nombre.cpp*) se llega al código máquina, con el mismo nombre que el archivo donde está el código fuente, y con la extensión **.obj**. Casi con toda probabilidad en código fuente hará uso de funciones que están ya definidas y precompiladas en las bibliotecas. Ese código precompilado está en archivos con la extensión **.lib**. Con el archivo .obj y los necesarios .lib que se deseen emplear, se procede al linkado o enlazado que genera un fichero ejecutable con la extensión **.exe**.

Estructura básica de un programa en C.

Aquí viene escrito un sencillo programa en C. Quizá convenga ponerse ahora delante del ordenador y, con el editor de C en la pantalla, escribir estar líneas y ejecutarlas. Más adelante se muestra cómo trabajar en un entorno de programación (se toma el Borland C++).

```
#include <stdio.h>
/* Este es un programa en C. */
// Imprime un mensaje en la pantalla del ordenador
void main(void)
{
   printf("mi primer programa en C");
}
```

Todos los programas en C deben tener ciertos componentes fijos. Vamos a ver los que se han empleado en este primer programa:

1. ***#include <stdio.h>***: Los archivos **.h** son los archivos de cabecera en C. Con esta línea de código se indica al compilador que se desea emplear, en el programa redactado, alguna función que está declarada en el archivo de biblioteca **stdio.h**. Esta archivo contiene las declaraciones de una colección de programas de entrada y salida por consola (pantalla y teclado).

   Esta instrucción nos permite utilizar cualquiera de las funciones declaradas en el archivo. Esta línea de código recoge el nombre del archivo **stdio.h**, donde están recogidos todos los prototipos de las funciones de entrada y salida estándar. Todo archivo de cabecera contiene identificadores, constantes, variables globales, macros, prototipos de funciones, etc.

   Toda línea que comience por **#** se llama **directiva de preprocesador**. A lo largo del libro se irán viendo diferentes directivas.

2. ***main***: Es el nombre de una función. Es la **función principal** y establece el punto donde comienza la ejecución del programa. La función *main* es necesaria en cualquier programa de C que desee ejecutar instrucciones. **Un código será ejecutable si y sólo si dispone de la función *main***.

3. ***void main(void)***: Los paréntesis se encuentran siempre después de un identificador de función. Entre ellos se recogen los parámetros

que se pasan a la función al ser llamada. En este caso, no se recoge ningún parámetro, y entre paréntesis se indica el tipo ***void***. Ya se verá más adelante qué significa esta palabra. Delante del nombre de la función principal (*main*) también viene la palabra ***void***, porque la función principal que hemos implementado no devuelve ningún valor.

4. ***/* comentarios */***: Símbolos opcionales. Todo lo que se encuentre entre estos dos símbolos son comentarios al programa fuente y no serán leídos por el compilador.

   Los comentarios no se compilan, y por tanto no son parte del programa; pero son muy necesarios para lograr unos códigos inteligibles, fácilmente interpretables tiempo después de que hayan sido redactados y compilados. Es muy conveniente, cuando se realizan tareas de programación, insertar comentarios con frecuencia que vayan explicando el proceso que se está llevando en cada momento. Un programa bien documentado es un programa que luego se podrá entender con facilidad y será, por tanto, más fácilmente modificado y mejorado.

   También se pueden incluir comentarios precediéndolos de la doble barra *//*. En ese caso, el compilador no toma en consideración lo que esté escrito desde la doble barra hasta el final de la presente línea.

5. ***;***: Toda sentencia en C termina con el **punto y coma**. En C, se entiende por sentencia todo lo que tenga, al final, un punto y coma. La línea antes comentada (***#include <stdio.h>***) no termina con un punto y coma porque no es una sentencia: es (ya lo hemos dicho) una directiva de preprocesador.

6. ***{}***: Indican el principio y el final de todo bloque de programa. Cualquier conjunto de sentencias que se deseen agrupar, para formar entre ellas una sentencia compuesta o bloque, irán marcadas

por un par de llaves: una antes de la primera sentencia a agrupar; la otra, de cierre, después de la última sentencia. Una función es un bloque de programa y debe, por tanto, llevarlas a su inicio y a su fin.

Elementos léxicos.

Entendemos por **elemento léxico** cualquier palabra válida en el lenguaje C. Serán elementos léxicos, o palabras válidas, todas aquellas palabras que formen parte de las palabras reservadas del lenguaje, y todas aquellas palabras que necesitemos generar para la redacción del programa, de acuerdo con una normativa sencilla.

Para crear un **identificador** (un identificador es un símbolo empleado para representar un objeto dentro de un programa) en el lenguaje C se usa cualquier secuencia de una o más **letras** (de la 'A' a la 'Z', y de la 'a' a la 'z', excluida las letras 'Ñ' y 'ñ'), **dígitos** (del '0' al '9') o **símbolo subrayado** ('_'). Un identificador es cualquier palabra válida en C. Con ellos podemos dar nombre a variables, constantes, tipos de dato, nombres de funciones o procedimientos, etc. También las palabras propias del lenguaje C son identificadores; estas palabras se llaman **palabras clave** o **palabras reservadas**.

Además de la restricción en el uso de caracteres válidos para crear identificadores, existen otras reglas básicas para su creación en el lenguaje C:

1. Debe comenzar por una letra del alfabeto o por el carácter subrayado. Un identificador no puede comenzar por un dígito.
2. El compilador sólo reconoce los primeros 32 caracteres de un identificador, pero éste puede tener cualquier otro tamaño mayor. Aunque no es nada habitual generar identificadores tan largos, si alguna vez así se hace hay que evitar que dos de ellos tengan iguales los 32 primeros caracteres, porque entonces para el compilador ambos identificadores serán el mismo.

3. Las letras de los identificadores pueden ser mayúsculas y minúsculas. El compilador distingue entre unas y otras, y dos identificadores que se lean igual y que se diferencien únicamente en que una de sus letras es mayúscula en uno y minúscula en otro, son distintos.
4. Un identificador no puede deletrearse igual y tener el mismo tipo de letra (mayúscula o minúscula) que una palabra reservada o que una función definida en una librería que se haya incluido en el programa mediante una sentencia *include*.

Las palabras reservadas, o palabras clave, son identificadores predefinidos que tienen un significado especial para el compilador de C. Sólo se pueden usar en la forma en que han sido definidos. El conjunto de palabras clave o reservadas (que siempre van en minúscula) en ANSI C es muy reducido (un total de 32) y son las siguientes:

auto	***double***	***int***	***struct***
break	***else***	***long***	***switch***
case	***enum***	***register***	***typedef***
char	***extern***	***return***	***union***
const	***float***	***short***	***unsigned***
continue	***for***	***signed***	***void***
default	***(goto)***	***sizeof***	***volatile***
do	***if***	***static***	***while***

A lo largo del manual se verá el significado de cada una de ellas. La palabra goto viene recogida entre paréntesis porque, aunque es una palabra reservada en C y su uso es sintácticamente correcto, de hecho no es una palabra permitida en un paradigma de programación estructurado como es el paradigma del lenguaje C.

Sentencias simples y sentencias compuestas.

Una **sentencia simple** es cualquier expresión válida en la sintaxis de C que termine con el carácter de punto y coma. **Sentencia compuesta** es

una sentencia formada por una o varias sentencias simples.

La sentencia simple queda definida cuando el programador termina una expresión válida en C, con un punto y coma. La sentencia compuesta se inicia con una llave de apertura (*{*) y se termina con una llave de clausura (*}*).

Errores y depuración.

No es extraño que, al terminar de redactar el código de un programa, al iniciar la compilación, el compilador deba abortar su proceso y avisar de que existen errores. El compilador ofrece algunos mensajes que clarifican frecuentemente el motivo del error, y la corrección de esos errores no comporta habitualmente demasiada dificultad. A esos errores sintácticos los llamamos **errores de compilación**. Ejemplo de estos errores pueden ser que se haya olvidado terminar una sentencia con el punto y coma, o que falte una llave de cierre de bloque de sentencias compuestas, o sobre un paréntesis, o se emplee un identificador mal construido…

Otras veces, el compilador no haya error sintáctico alguno, y compila correctamente el programa, pero luego, en la ejecución, se producen errores que acaban por abortar el proceso. A esos errores los llamamos **errores de ejecución**. Un clásico ejemplo de este tipo de errores es forzar al ordenador a realizar una división por cero, o acceder a un espacio de memoria para el que no estamos autorizados. Esos errores también suelen ser sencillos de encontrar, aunque a veces, como no son debidos a fallos sintácticos ni de codificación del programa sino que pueden estar ocasionados por el valor que en un momento concreto adquiera una variable, no siempre son fácilmente identificables, y en esos casos puede ser necesario utilizar los depuradores que muchos entornos de programación ofrecen.

Y puede ocurrir también que el código no tenga errores sintácticos, y por

tanto el compilador termine su tarea y genere un ejecutable; que el programa se ejecute sin contratiempo alguno, porque en ningún caso se llega a un error de ejecución; pero que el resultado final no sea el esperado. Todo está sintácticamente bien escrito, sin errores de compilación ni de ejecución, pero hay errores en el algoritmo que pretende resolver el problema que nos ocupa. Esos errores pueden ser ocasionados sencillamente por una errata a la hora de escribir el código, que no genera un error sintáctico, ni aborta la ejecución: por ejemplo, teclear indebidamente el operador suma (+) cuando el que correspondía era el operador resta (-).

Todo error requiere una modificación del programa, y una nueva compilación. No todos los errores aparecen de inmediato, y no es extraño que surjan después de muchas ejecuciones.

Recapitulación.

En este capítulo hemos introducido los conceptos básicos iniciales para poder comenzar a trabajar en la programación con el lenguaje C. Hemos presentado el entorno habitual de programación y hemos visto un primer programa en C (sencillo, desde luego) que nos ha permitido mostrar las partes básicas del código de un programa: las directivas de preprocesador, los comentarios, la función principal, las sentencias (simples o compuestas) y las llaves que agrupan sentencias. Y hemos aprendido las reglas básicas de creación de identificadores.

El entorno de Borland C++.

Al ejecutar la aplicación de Borland C++, aparece una pantalla gris. Lo primero que se debe hacer es crear un nuevo documento donde podremos escribir nuestro código. Para esto basta seguir los pasos indicados en la Figura 1.2.

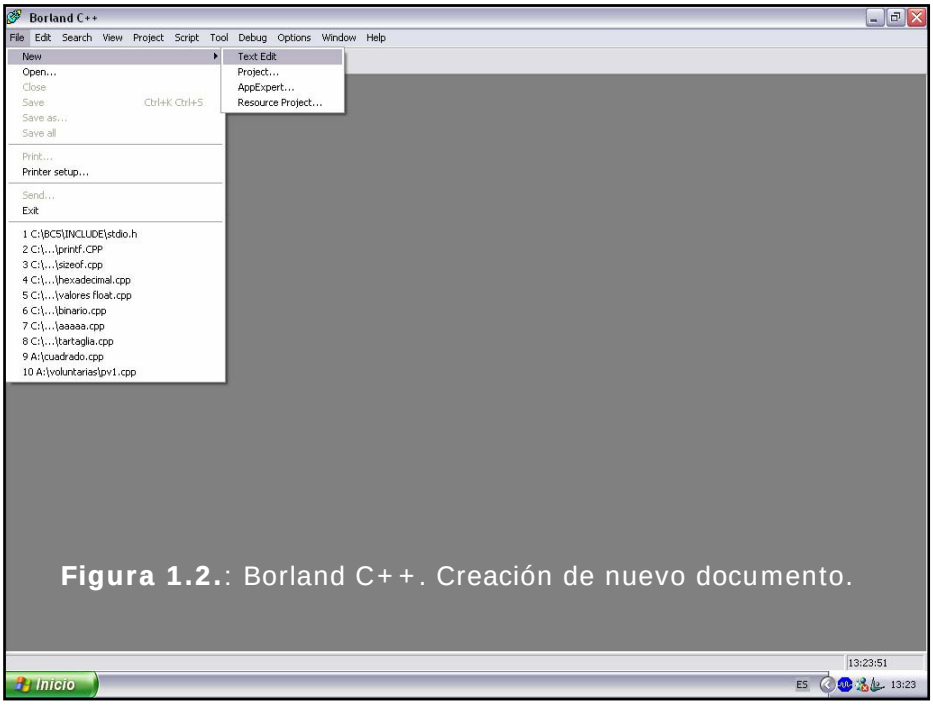

Figura 1.2.: Borland C++. Creación de nuevo documento.

Una vez creado el documento, ya podemos escribir en él. Vamos a copiar el texto del programa ejemplo de este capítulo. En la figura 1.3. vemos el código copiado. Podemos ver cómo ha quedado escrito en letras azules todo lo que es comentario. En letras verdes las directivas de preprocesador. Y en letras también azules los textos recogidos entre comillas. Todo este código de colores (y otros colores de letra que se irán viendo), y de tipos de letra (cursiva, negrita) ayudan en la confección del programa.

Supongamos ahora que guardamos el archivo en una carpeta nueva llamada “PrimerPrograma”. El archivo (con extensión **cpp**) ocupa 167 bytes aunque por exigencias del disco donde se almacena el archivo emplea hasta un total de 4 Kbytes.

Al compilar el programa, aparecen en la carpeta otros tantos archivos: uno con la extensión **obj**, y otro con la extensión **exe**, que es el ejecutable. Los tamaños de ambos archivos pueden verse buscando las propiedades de ambos.

Para compilar y ejecutar basta con elegir la opción Menú Debug → Run o, como indica esa misma opción, pulsar simultáneamente las teclas Control y F9. Una tercera opción es pulsar el botón en forma de rayo amarillo situado en la sexta posición de la barra de botones.

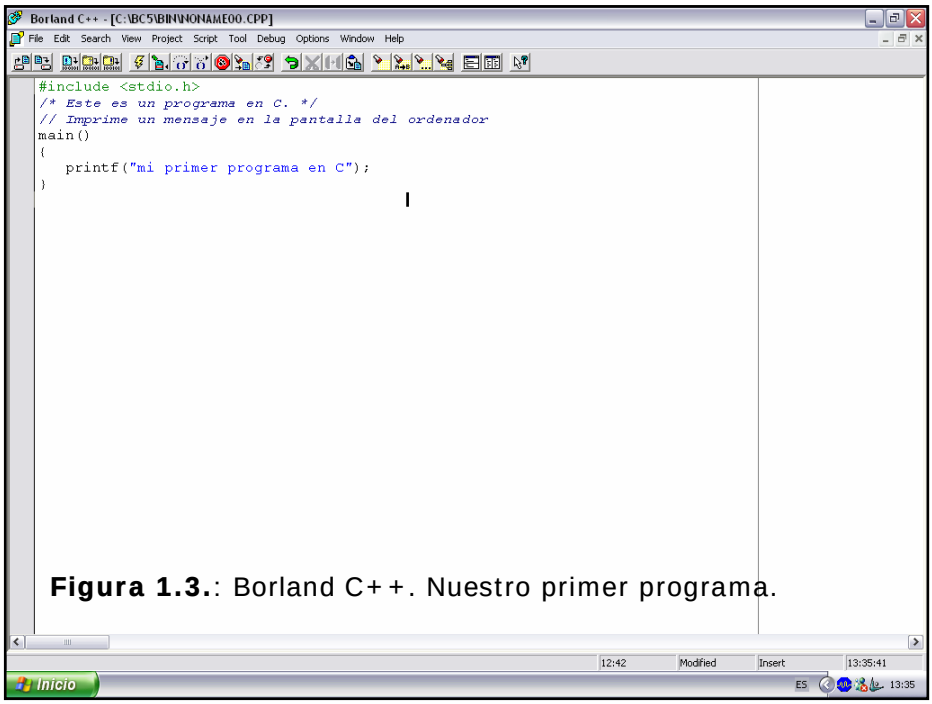

Figura 1.3.: Borland C++. Nuestro primer programa.