Introducción al lenguaje C

1. El entorno de desarrollo lcc

1.1. Instalación

El compilador lcc es un programa de código abierto que se puede descargar de www.cs.virginia.edu/~lcc-win32/ o desde www.q-software-solutions.de/products/lcc-win32/index.shtml. En cualquiera de los dos casos es interesante descargar también el manual de usuario para obtener la ayuda en línea del entorno de desarrollo (archivo manual.exe). También puede ser interesante descargar el tutorial de C en pdf (archivo tutorial.pdf).

A la hora de instalar el compilador, en lugar de hacerlo en el típico "Program files" o "Archivos de programa", conviene hacerlo en la carpeta que sugiere (`c:\lcc`), ya que puede dar problemas con los espacios en blanco en el nombre de los directorios. Además crea una carpeta en "Mis Documentos" llamada `lcc` con información sobre los proyectos que se vayan generando.

Si no se dice lo contrario, los programas fuente los creará en la carpeta `c:\lcc\projects`, y por cada proyecto (o ejecutable) que se cree, utilizará una carpeta `c:\lcc\projects\lcc`, `c:\lcc\projects\lcc1`, `c:\lcc\projects\lcc2`, etc. (la ubicación de los fuentes y de los archivos de salida se pueden cambiar con las propiedades del proyecto).

1.1.1. Componentes

En la instalación se incluyen, entre otros, los siguientes componentes:

- Compilador de línea de órdenes (`lcc.exe`), compila y genera los programas objeto.
- Linker (`lcclnk.exe`), genera los archivos ejecutables a partir de uno o varios programas objeto.
- El entorno de desarrollo integrado (IDE) llamado `wedit.exe`. Desde el entorno de desarrollo se editan los programas, se compilan, montan y ejecutan. También incluye el depurador, editores de recursos para aplicaciones basadas en entornos gráficos Windows, etc.

1.1.2. Compilador de línea de órdenes

A partir de un programa fuente, se puede compilar un programa directamente desde la línea de órdenes. Hay que tener en cuenta que los binarios (`lcc.exe` y `lcclnk.exe`) están en la carpeta `c:\lcc\bin`, por lo que es interesante agregar esa carpeta a la variable de entorno `path`.

1. Abrir una ventana de consola. INICIO/EJECUTAR/CMD en Windows XP.
2. Establecer la variable de entorno `PATH`:

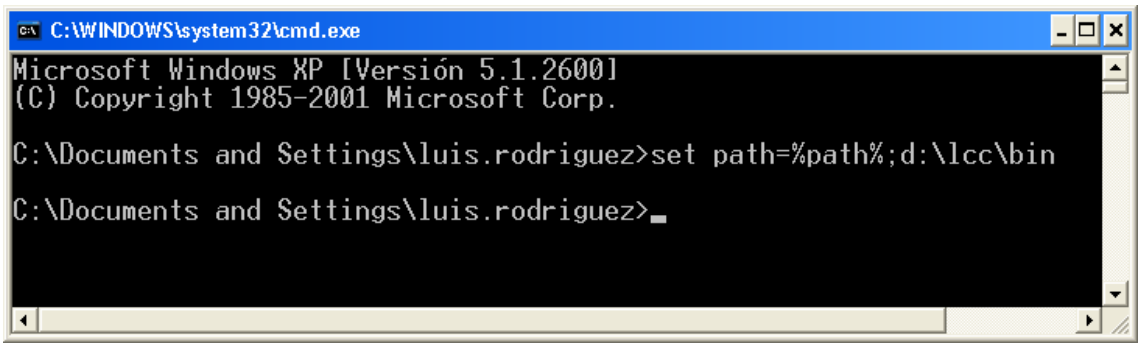

Figura 1.1. Establecer la variable path desde la línea de órdenes

También se puede establecer esa variable de entorno como permanente con el panel de control de Windows. En el PANEL DEL CONTROL de Windows XP, elegir la opción SISTEMA. En la pestaña de OPCIONES AVANZADAS, seleccionar el botón VARIABLES DE ENTORNO.

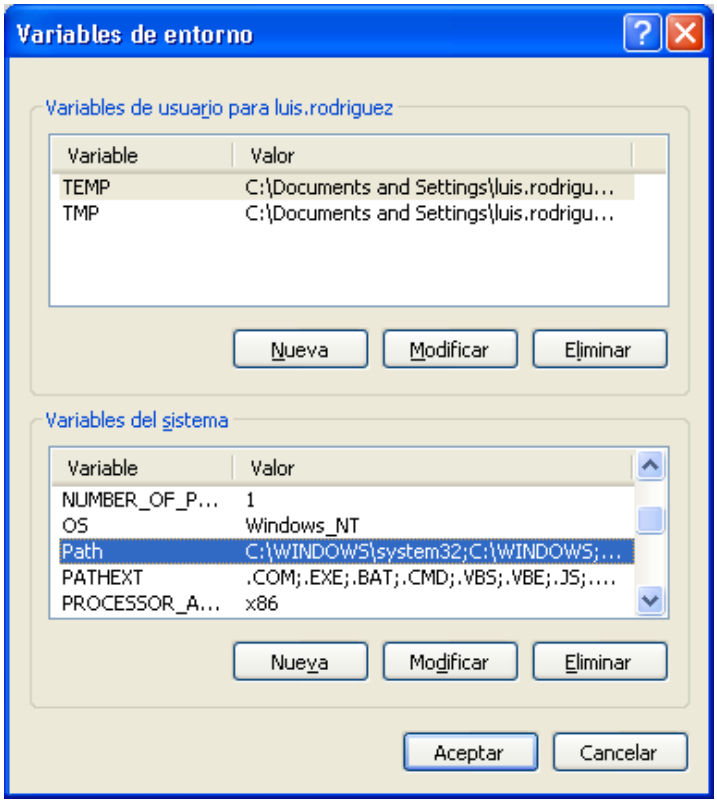

Figura 1.2. Establecer la variable path desde la interfaz Windows

A partir de esa ventana habría que marcar la variable `path`, pulsar el botón MODIFICAR e incluir en ella la carpeta `bin` del directorio dónde está instalado el programa (figura 1.3).

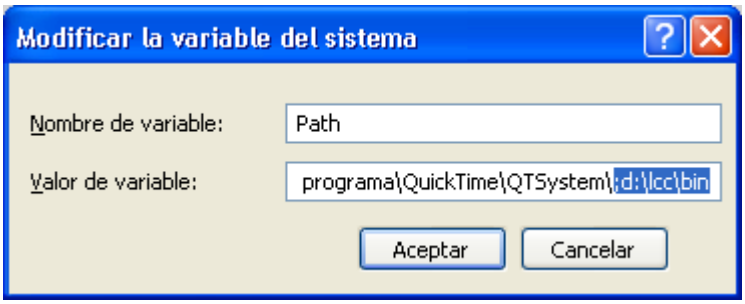

Figura 1.3. Modifica la variable Path

3. Con un editor de texto plano (por ejemplo el Bloc de notas), crear el programa fuente en C.

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

Guardar el archivo con extensión .c (por ejemplo, "`holamundo.c`").

4. En la interfaz de línea de órdenes, llamar al compilador de C, `lcc`.

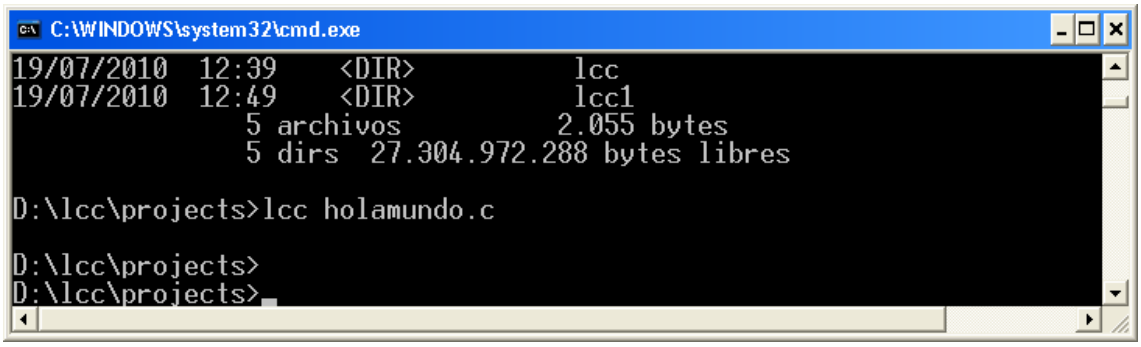

Figura 1.4. Llamada al compilador desde la línea de órdenes

Esto creará, entre otras cosas, el programa objeto, `holamundo.obj`.

5. Enlazar el archivo llamando al linker `lcclnk`.

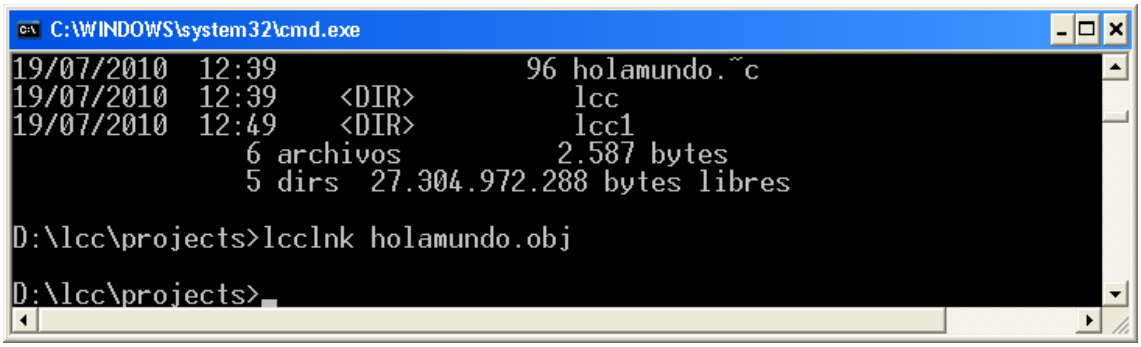

Figura 1.5. Llamada al *linker* desde la línea de órdenes

Con lo que se crea el ejecutable `holamundo.exe`.

6. Ejecutar el programa. Simplemente hay que llamar al ejecutable `holamundo.exe`.

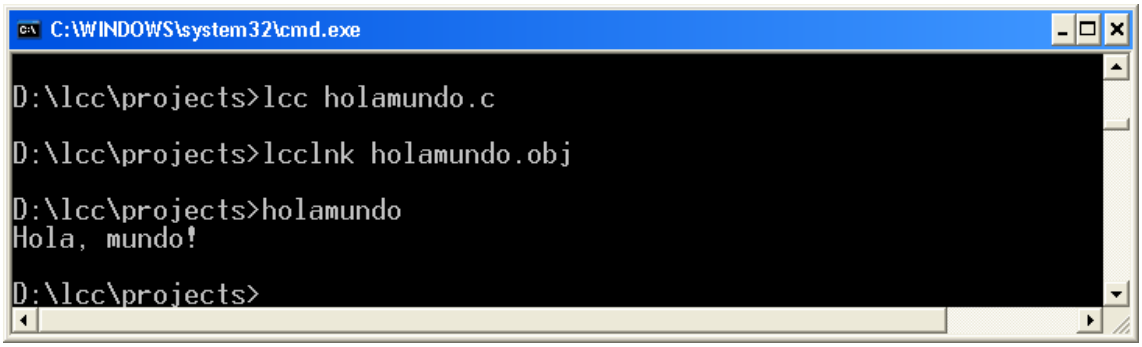

Figura 1.6. Ejecución y salida de resultados desde la línea de órdenes

1.2. El entorno de desarrollo

1.2.1. La interfaz de usuario

Aunque en ocasiones es más flexible utilizar el compilador de línea de órdenes, lo normal es utilizar un entorno de programación integrado (IDE, *Integrated Development Environment*). Un IDE (figura 1.7) normalmente incluye un editor de textos (casi siempre preparado de alguna forma para introducir el programa fuente en un lenguaje determinado) y herramientas para compilar y montar el programa sin salir del propio entorno mediante un sistema de menús. Además suelen incluir otro tipo de herramientas para gestionar los errores sintácticos que se produzcan, depuradores de código[1], gestores de archivos make (*make files*[2]), etc.

[1] Herramientas que permiten hacer un seguimiento del programa paso a paso para ver qué valores van tomando las variables en cada una de las instrucciones, comprobar si el flujo de control pasa por algún punto determinado, detener la ejecución si se da alguna condición o si se pasa por algún punto, etc.
[2] Archivos de texto que automatizan la compilación de programa complejos.

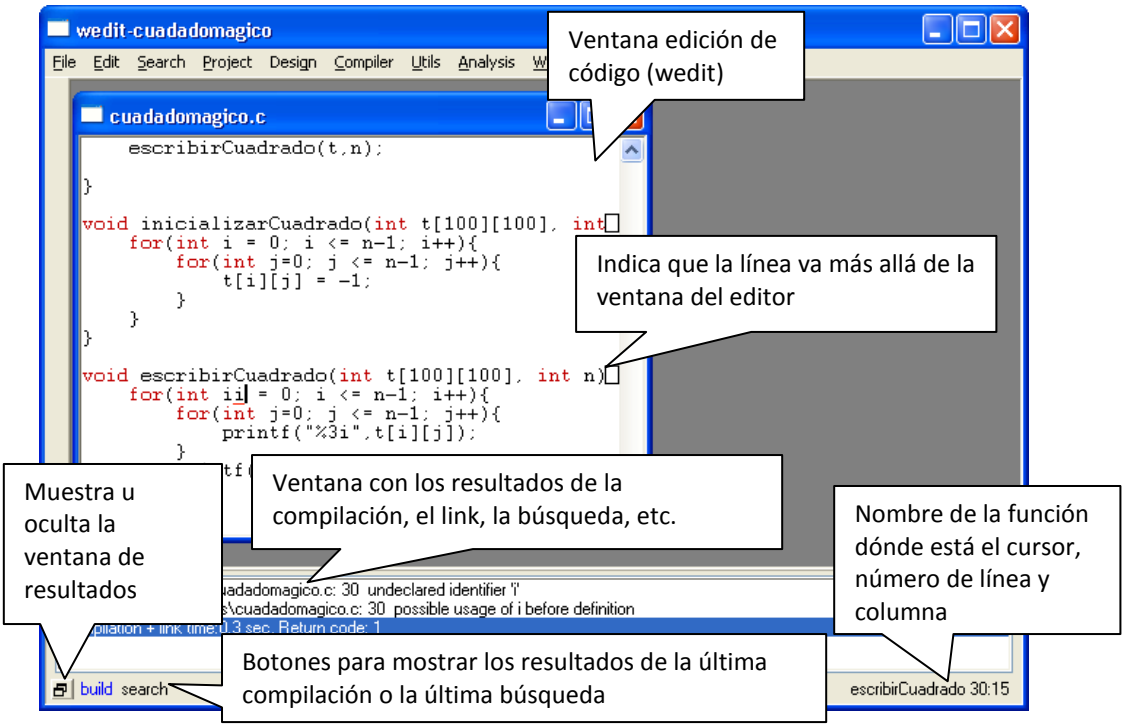

Figura 1.7. Pantalla principal de wedit

La ventana del editor de código de wedit no tiene barra de desplazamiento horizontal. Para ver la parte oculta de la línea de código, hay que pulsar con el ratón en la parte derecha, cerca de la barra de desplazamiento vertical, o en la parte izquierda, cerca del borde de la ventana.

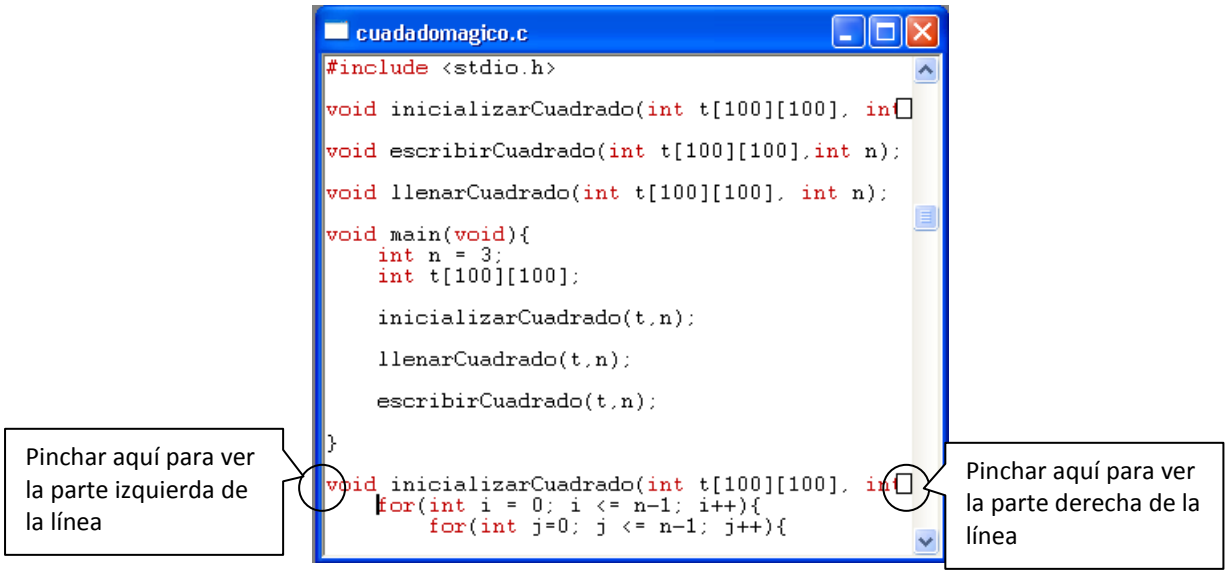

Figura 1.8. Ventana del editor de código

Otra utilidad del editor de código es la posibilidad de ver los distintos niveles de las llaves en el código fuente. La opción LEVEL del menú EDIT, permite visualizar una columna con los niveles de las llaves.

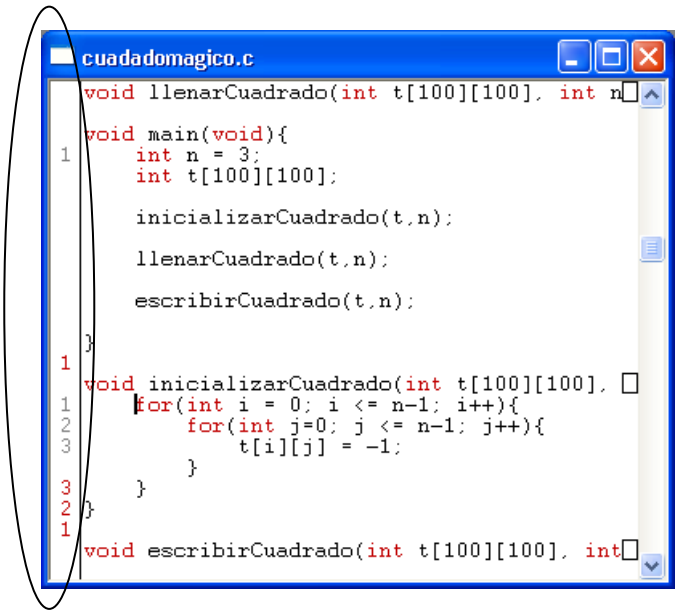

Figura 1.9. Niveles de llaves en el editor de código

Otra posibilidad del editor es ver los cambios que se han realizado desde la última carga del programa. En el menú EDIT, la opción SHOW CHANGES muestra una ventana con los cambios que se han realizado.

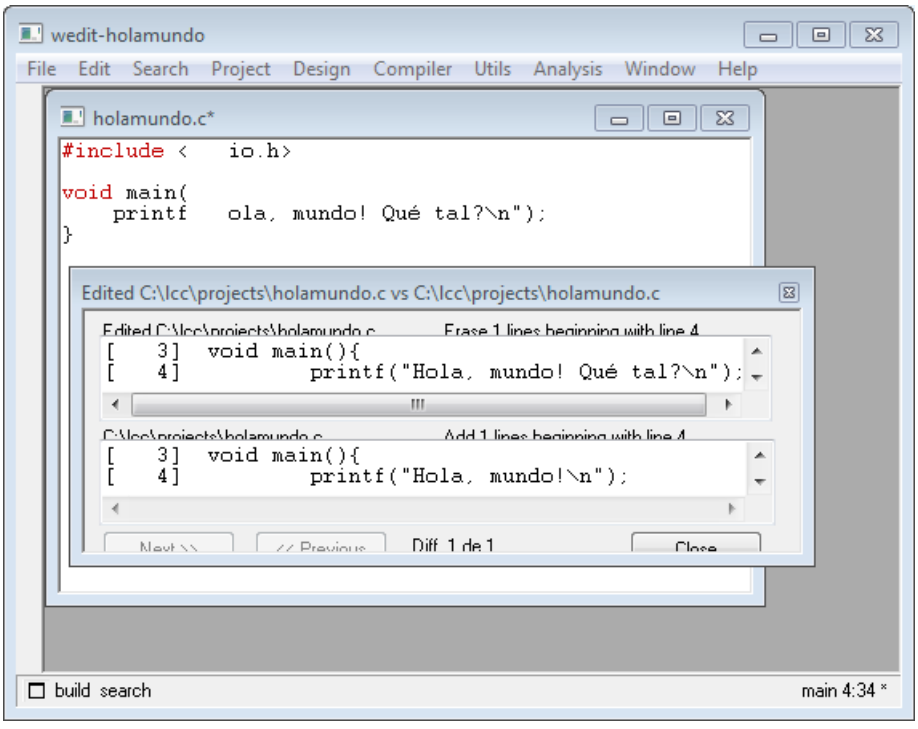

Figura 1.10. Vista de las modificaciones con la opción SHOW CHANGES

También es posible descartar los últimos cambios, recargando el archivo original con la opción RELOAD del menú FILE.

La parte inferior del IDE puede mostrar la pantalla de salida. Para mostrarla y ocultarla se utiliza el botón de la esquina inferior izquierda (figura 1.7).

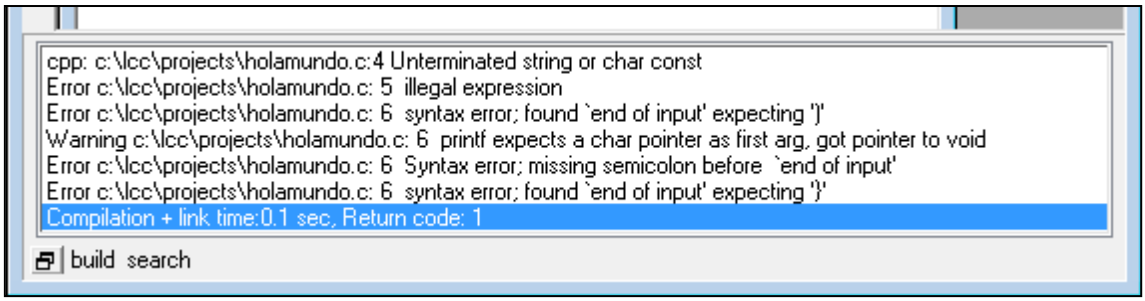

Figura 1.11. Vista de los errores sintácticos

La ventana de salida muestra los resultados de las distintas opciones que se pueden hacer en el entorno, como los resultados de la compilación, con los errores en tiempo de compilación, los resultados del *linker*, etc.

1.2.2. Crear y editar un programa con wedit

Aunque wedit trabaja con proyectos (se verá un poco después), la forma más fácil de crear un programa simple es con la opción NEW del menú FILE y seleccionar la opción FILE. Preguntará entonces el nombre del programa…

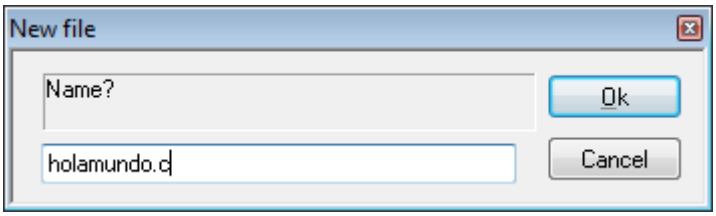

Figura 1.12. Cuadro de diálogo para crear un nuevo programa

Al pulsar el botón OK, se abrirá el editor de código y podremos escribir el programa fuente.

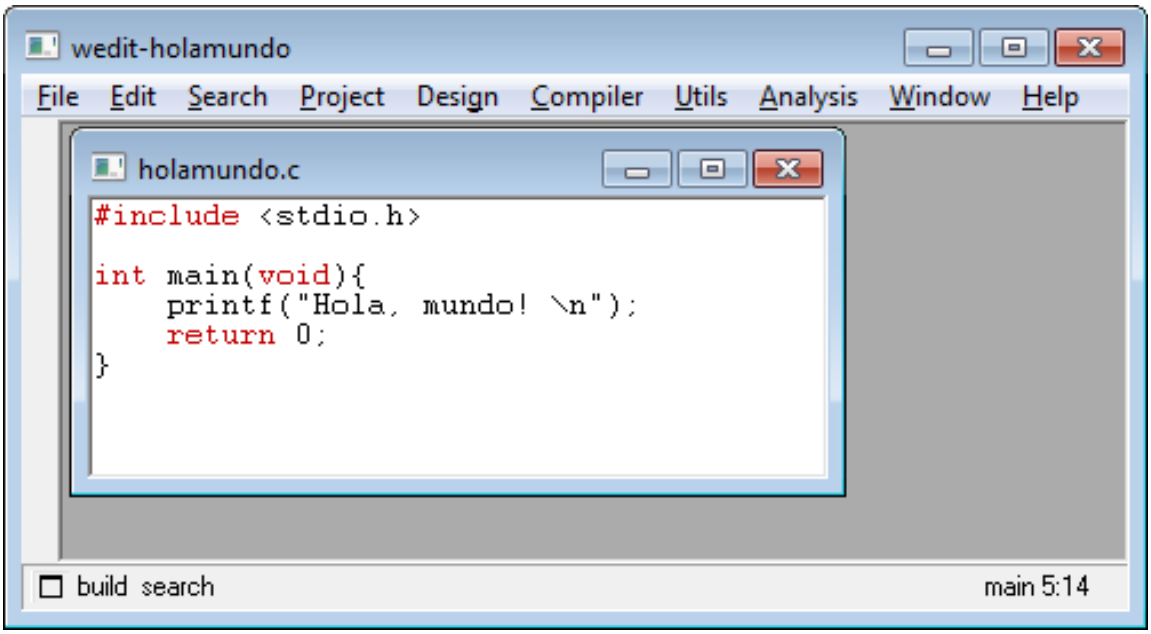

Figura 1.13. El programa holamundo.c en el editor de código

Explicación del programa "holamundo.c"

1. El lenguaje C tiene muy pocas palabras clave. Casi todas sus capacidades las obtiene a partir de la funciones de bibliotecas (un conjunto de funciones escritas normalmente en C) que se incluyen en el programa fuente y que están incluidas en el estándar de C, creadas por terceros o creadas por el propio programador. El "preprocesador" de los compiladores C permiten incluir ese código fuente mediante la directiva `#include` que permite añadir los llamados "archivos de cabecera" (archivos con extensión .h). En este programa se incluye la biblioteca `stdio.h`, que contiene las funciones estándar de entrada y salida. El nombre del archivo de cabecera está encerrado entre los símbolos < y >, lo que indica que se trata de un archivo de cabecera estándar que se encuentra en la ubicación estándar de los archivos de cabecera (la carpeta `c:\lcc\include`). Si se deseara incluir un archivo de cabecera ubicado en otra carpeta, se encerraría entre comillas ("); por ejemplo `#include "micabecera.h"`.
2. La función `main` es la función de entrada de un programa en C. Al compilar el programa, en el código objeto se indica que el programa debe empezar por la primera instrucción o declaración que aparezca en esa función. En su formato más simple, la función devolverá un valor entero (por eso comienza por la palabra reservada `int`) y no recibe ningún argumento (entre paréntesis, la palabra reservada `void` indica que no recibe ningún dato).
3. El cuerpo del programa va encerrado entre llaves. Cada bloque de programa se encierra entre llaves y puede haber varios niveles de anidamiento de llaves (figura 1.7).
4. Para sacar por pantalla la cadena `Hola, mundo!`, se utiliza una llamada a la función estándar `printf` que está contenida en el archivo de cabecera `stdio.h`. Una llamada a una función tiene el nombre de la función y, encerrados entre paréntesis la lista de argumentos, cuyo número, orden y tipo cambiará de una función a otra.
5. En este caso, el argumento de la función `printf` es la constante de cadena `Hola, mundo!`. En C, las constantes de cadena van encerradas entre comillas dobles. Al final de la cadena hay una secuencia de escape (`\n`) que indica que después de escribir la cadena por la salida estándar se hará un retorno de carro (se salta a la siguiente línea).
6. Al final de la instrucción aparece un punto y coma. Las instrucciones en C terminan con el carácter ;.
7. La palabra reservada `return` indica el valor de retorno que tendrá la función, es decir el valor que devolverá la función al ser llamada. Se trata de un valor entero (hay que recordar que la función `main` devuelve un entero) que en este caso será 0. Aunque lo correcto es poner el valor de retornola función, se podría evitar incluir esta instrucción ya que no se hará uso de dicho valor.
8. La llave de cierre indica el fin de la función.

El programa se guarda con la opción SAVE del menú FILE (o pulsando CTRL+S) con el nombre indicado al principio. Normalmente el programa fuente tendrá una extensión `.c`. El archivo se guardará, si no se indica lo contrario en el directorio `lcc\projects`.

1.2.3. Compilar el programa

La opción COMPILE del menú COMPILER permite compilar el programa. Si tenemos abierto un proyecto, pregunta si queremos incluir el archivo fuente en ese proyecto o crear un nuevo proyecto.

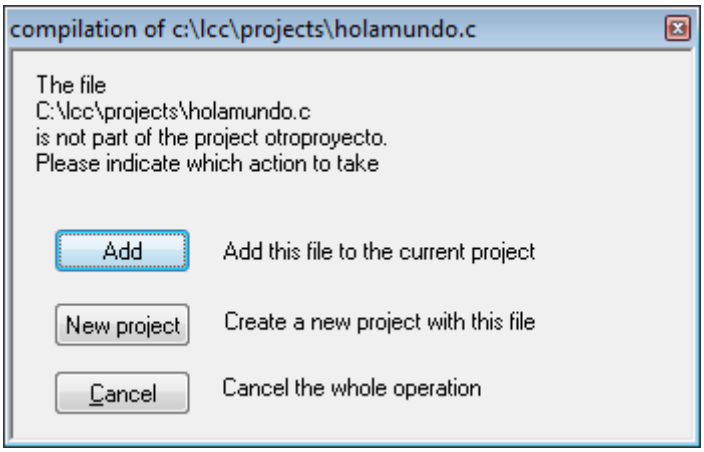

Figura 1.14. Ventana de compilación de un programa fuente no incluido en un proyecto

En nuestro caso será un proyecto nuevo, por lo que pulsaremos el botón NEW PROJECT con lo que se indicará que se ha creado el proyecto `holamundo` y la carpeta dónde se ha guardado. A no ser que se indique lo contrario, el proyecto se guardará en la carpeta `lcc\projects\lccxx`, dónde *xx* es un número correlativo. En esa carpeta se generarán los archivos necesarios para la compilación y el enlace del programa (por ejemplo el programa objeto `.obj` o el archivo ejecutable `.exe`).

El proceso de compilación

Al utilizar la opción de compilación (o la opción MAKE del menú COMPILER o F9) ocurren una serie de acciones transparentes para el usuario. En primer lugar wedit llama a propio compilador `lcc.exe`. Este programa leer el archivo fuente y genera el archivo objeto. C soporta la compilación separada y permite crear varios programas objeto que se enlazarán con el linker `lcclnk.exe`. Este programa también es necesario para producir el programa ejecutable. En el ejemplo, se generaría un archivo `holamundo.obj` y el ejecutable `holamundo.exe`.

El proceso de compilación sigue los siguientes pasos:

1. El archivo fuente primero es preprocesado. Se resuelven las directivas `#include` y el código de los archivos de cabecera se insertan en el código fuente.
2. El compilador procesa el texto resultante. En este paso genera una serie de instrucciones de código intermedio.
3. El generador de código utiliza ese código intermedio y genera instrucciones en lenguaje ensamblador.
4. El ensamblador toma esas instrucciones, las codifica de forma que la CPU las pueda entender y las empaqueta en el programa objeto.
5. Este programa pasa por el linker y crea el ejecutable.

Si durante estos pasos se detecta algún error sintáctico en el programa fuente, la compilación se detiene y wedit muestra en la ventana de salida los errores producidos en tiempo de compilación, es decir, los errores sintácticos.

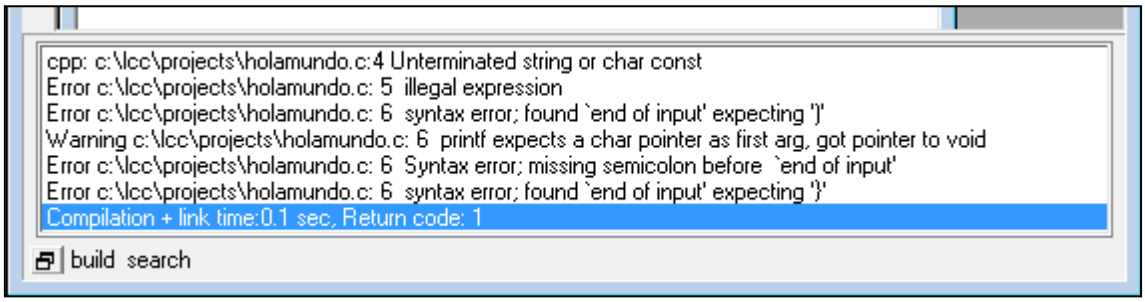

Figura 1.15. Vista de los errores sintácticos

En el caso de que todo vaya correctamente, la ventana de salida sólo mostrará el mensaje de que el proceso se ha realizado de forma satisfactoria.

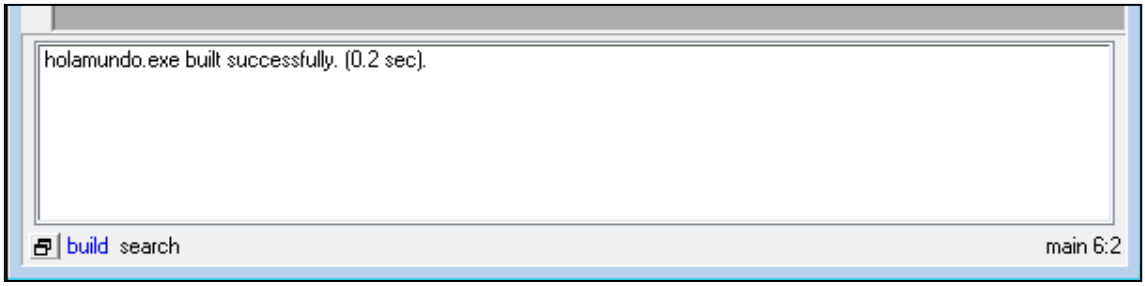

Figura 1.16. Vista de los resultados de la compilación de un programa correcto sintácticamente

1.2.4. Ejecutar el programa

Para ejecutar el programa hay que utilizar la opción EXECUTE del menú COMPILER (CTRL+F5). Si se trata de una aplicación de consola (aplicación con salida en modo texto) se abrirá una ventana del sistema con la salida del programa.

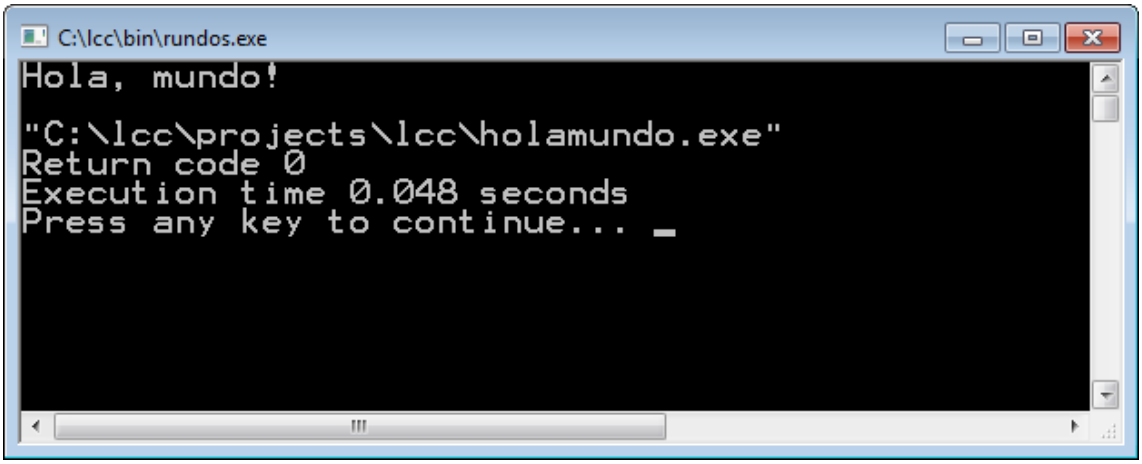

Figura 1.17. Ventana de salida de la ejecución de un programa desde el entorno de desarrollo

1.3. Trabajar con proyectos

Casi todos los entornos de programación trabajan con "proyectos"; de hecho cualquier programa fuente debe ser incluido dentro de un proyecto, de forma implícita o explícita, si se desea ejecutarlo.

Un proyecto es un conjunto de uno o más programas fuente que contienen el código de un ejecutable o una biblioteca. Puede incluir archivos de cabecera, documentación, etc. En el caso de wedit sólo incluirá archivos de código y archivos de recursos[3].

En un proyecto, cada archivo con código fuente ejecutable generará un programa objeto que se podrá compilar de forma independiente y que generarán al final un único programa ejecutable

1.3.1. Creación del proyecto

La opción CREATE del menú PROJECT se utiliza para crear un proyecto nuevo. En la creación del proyecto se debe proporcionar, por lo menos, el nombre del proyecto y la carpeta dónde se ubicarán los archivos con código fuente, es decir, el directorio de trabajo.

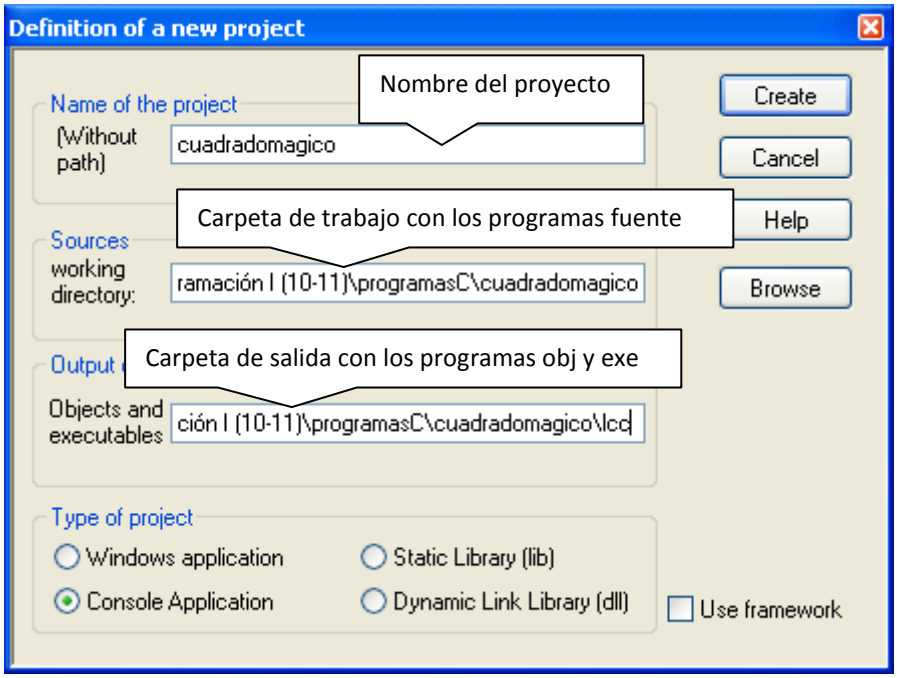

Figura 1.18. Cuadro de diálogo para la creación de un proyecto

Los programas objeto y los ejecutables se almacenarán, si no se dice lo contrario, en una carpeta dentro del directorio de trabajo que tendrá el nombre de `lcc`.

En esta ventana también se puede elegir el tipo de proyecto entre una aplicación de consola (las que se harán en este curso), una aplicación para Windows o bibliotecas estáticas o dinámicas.

[3] Los archivos de recursos se utilizan en aplicaciones complejas para almacenar mapas de bits, cursores, iconos, sonidos, elementos para aplicaciones multilingües, etc.

El botón CREATE permitirá crear el proyecto. El compilador lcc mantiene una lista de los proyectos creados dentro del archivo `projects.ini` que se almacena dentro de la carpeta `Mis documentos\lcc`.

El siguiente cuadro de diálogo permite indicar si queremos generar un esqueleto de la aplicación. La creación de un esqueleto puede ahorrar tiempo en aplicaciones complejas, pero en el caso del presente curso es preferible indicar que NO.

A continuación se pueden incluir distintos archivos fuente dentro del proyecto. Si partimos de un proyecto nuevo se daría al botón CANCELAR, si por el contrario se desea aprovechar archivos fuente ya existentes se marcarán y se pulsará el botón ACEPTAR.

La siguiente pantalla servirá para gestionar los archivos fuente incluidos en el proyecto. Si no hay más archivos que añadir se volverá a pulsar el botón CANCELAR. Si, de momento, no existe ningún archivo fuente, se advertirá de esa circunstancia, preguntando si se desea continuar con la creación de un proyecto vacío, como sería el presente caso.

Por último, se permitirá modificar algunas de las opciones del compilador, el *linker* y el depurador para el proyecto actual.

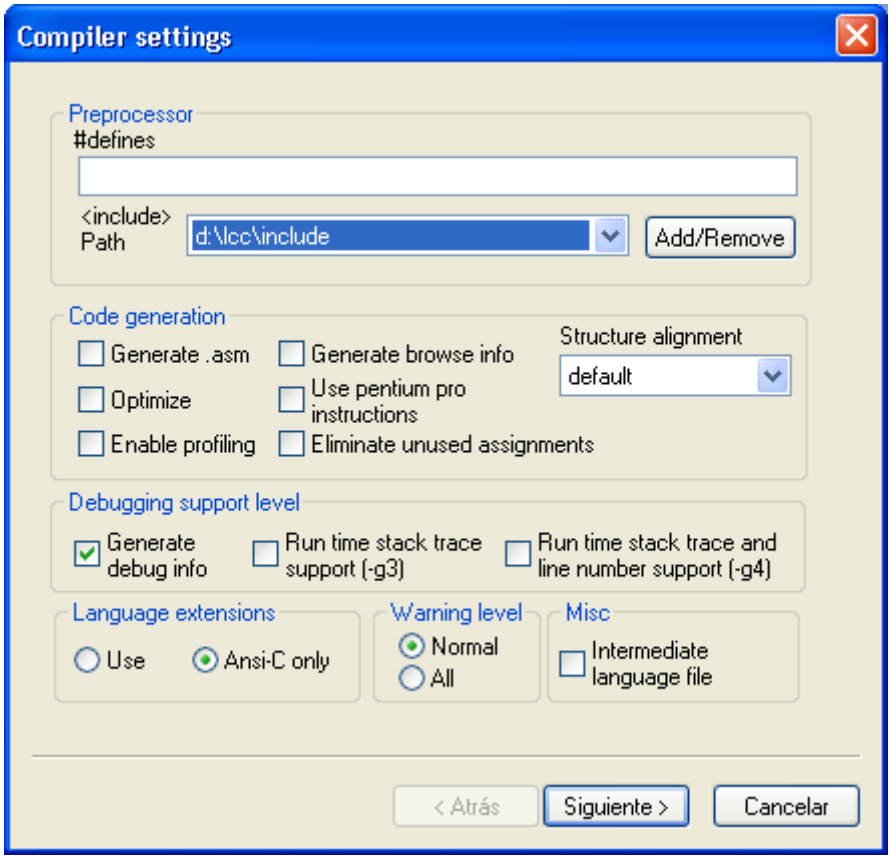

Figura 1.19. Cuadro de diálogo con las opciones del compilador

De momento no se tocarán las opciones por omisión. Sin embargo, puesto que el curso se va a centrar en el lenguaje C estándar (ANSI-C), puede ser interesante marcar el botón de radio ANSI-C ONLY para que el compilador no admita ninguna instrucción no estándar en el programa.

En los siguientes cuadros de diálogo con las opciones del *linker* y del depurador se dejarán las opciones por omisión pulsando el botón SIGUIENTE y FINALIZAR respectivamente.

Una vez que el proyecto ya está creado, se puede crear el o los programas fuente y añadirlos al proyecto con la opción ADD/DELETE FILES… del menú PROJECT[4]. También, como aparece más arriba (apartado 1.2.3. Compilar el programa), si se compila un archivo nuevo que no esté todavía integrado en ningún proyecto aparece un cuadro de diálogo preguntando si se desea añadir al proyecto actual, crear un nuevo proyecto o cancelar la operación (Figura 1.14).

Con los archivos fuentes cargados podemos compilar impedientemente cada uno de los programa fuentes (opción COMPILE *NOMBREARCHIVO.C* del menú COMPILER), crear un único programa ejecutable compilando sólo los archivos fuente del proyecto que se hubieran modificaco (opción MAKE del menú COMPILER), o reconstruir todo el programa ejecutable borrando todos los antiguos programa objeto (opción REBUILD all del menú COMPILER). El resultado de estas dos últimas opciones será un único programa ejecutable (con extensión `.exe`) que se podrá ejecutar directamente desde la línea de órdenes o mediante la opción EXECUTE *NOMBREARCHIVO.EXE* del mismo menú.

1.3.2. Abrir o borrar proyectos

Las opciones OPEN y DELETE del menú PROJECT permite, respectivamente, abrir o eliminar un proyecto existente. Ambas muestran un cuadro de diálogo muy similar.

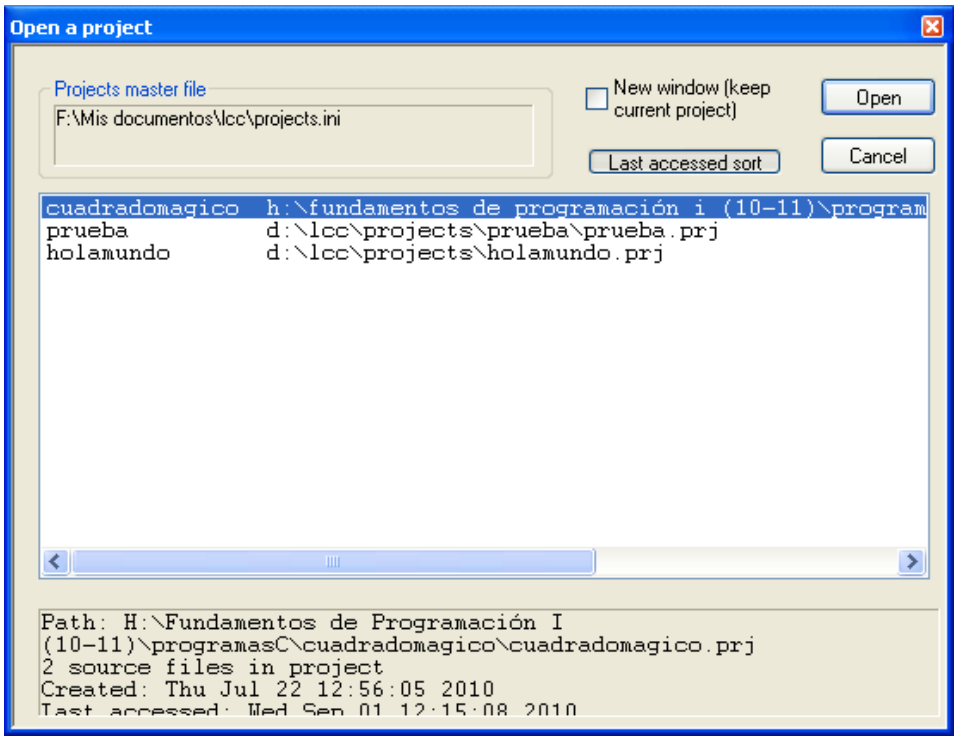

Figura 1.20. Cuadro de diálogo para abrir un proyecto

[4] Aunque el cuadro de diálogo para añadir y eliminar archivos del programa fuente incluye un botón REMOVE SELECTED FILE, la opción no funciona correctamente en la versión actual de wedit.

Hay que tener en cuenta que la eliminación de un proyecto sólo lo elimina de la lista de proyectos de lcc; tanto las carpetas como los archivos fuentes, los objeto, etc. no se eliminan. Podemos importar un proyecto completo (por ejemplo, después de haberlo eliminado) mediante la opción IMPORT del menú PROJECT, e indicando el archivo de proyecto (extensión `.prj`) del proyecto que se desea importar.

1.4. El depurador

1.4.1. Errores de programación

A la hora de realizar un programa se pueden dar dos tipos de errores:

- Errores sintácticos (errores en tiempo de compilación).
- Errores en tiempo de ejecución.

Errores en tiempo de compilación

Se producen por una escritura errónea del programa fuente, de forma que no respeta la sintaxis del lenguaje. Durante el proceso de compilación, uno de los pasos consiste en comprobar si cada elemento del programa fuente corresponde a la sintaxis del lenguaje del compilador: ¿las palabras clave están bien escritas? ¿Se han declarado todas las variables utilizadas? ¿Las distintas instrucciones se han escrito con el formato requerido? Si el compilador detecta alguno de estos errores el proceso se detiene y no se puede generar el programa objeto.

Los compiladores avisan de los errores detectados en forma de mensajes de error. Si se compila desde la línea de órdenes, aparece un listado de errores por la consola.

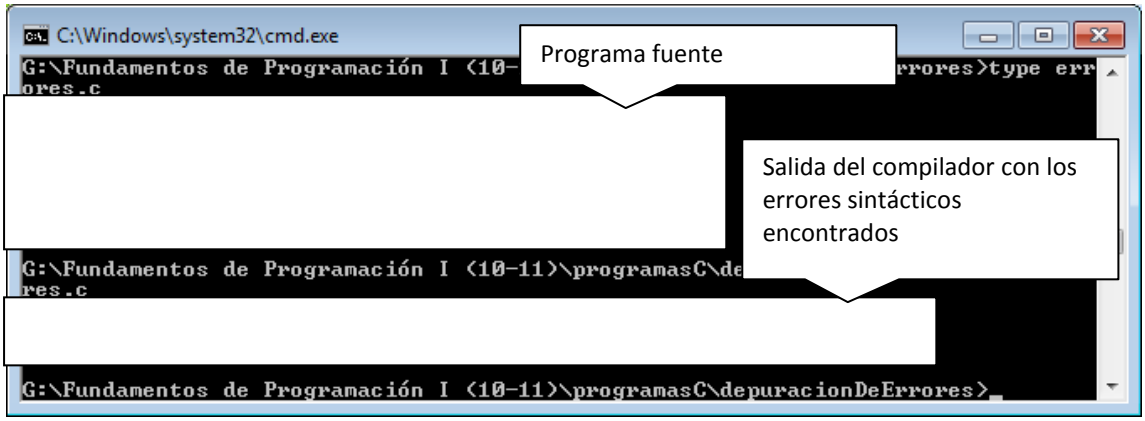

Figura 1.21. Errores sintácticos detectados mediante el compilador de línea de órdenes

El entorno de programación de lcc también muestra los errores sintácticos en la pantalla de salida de la parte inferior de la ventana.

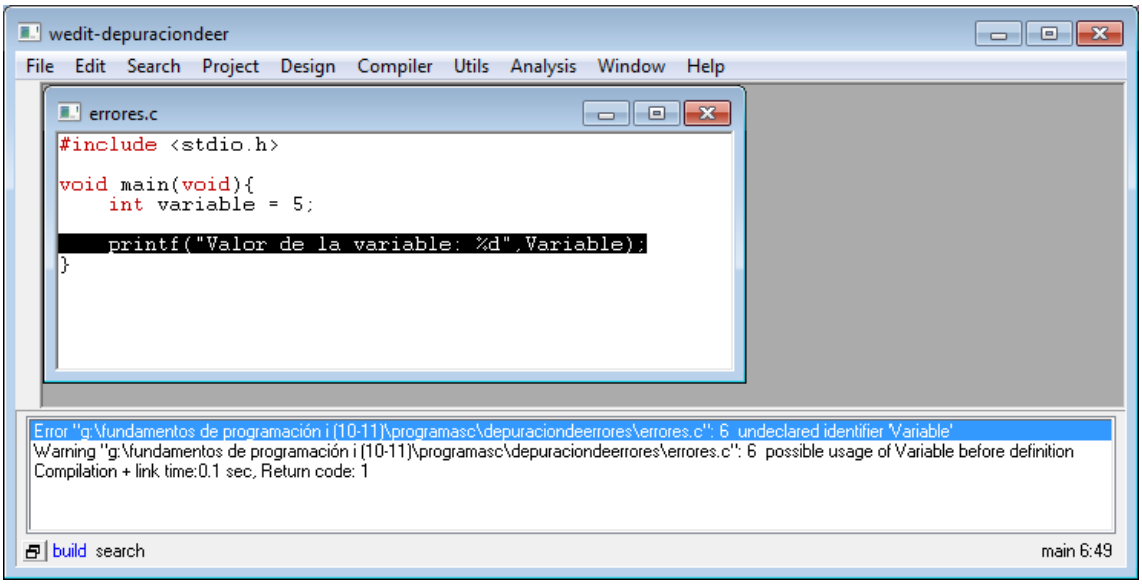

Figura 1.22. Ventana de salida de wedit con los errores sintácticos detectados

Tanto en la figura 1.21 como en la figura 1.22 aparecen los errores sintácticos del programa `errores.c`. Aparece la palabra Error, el programa fuente dónde se ha detectado el error, la línea dónde se ha detectado y una explicación del mismo.

```
Error "g:\...\errores.c": 6  undeclared identifier 'Variable'
```

En este caso, se ha producido un error en la línea 6, ya que no se ha declarado el identificador `Variable` (la declaración del identificador se ha hecho en minúsculas).

También muestra los *warnings* (advertencias) que indican posibles problemas pero que pueden no ser críticos.

Hay que tener en cuenta que los errores de compilación son, en ocasiones, más difíciles de interpretar que el ejemplo anterior. Si existen varios errores sintácticos es posible que, inicialmente, no se marquen todos los que están o que aparezcan errores que se arrastran a partir de otros y que desaparecerán al solucionar el primero de ellos.

Por ejemplo en el siguiente código fuente:

```
#include <stdid.h>
void main(void){
int variable = 5;
     printf("Valor de la variable: %d,Variable);
}
```

la salida de errores muestra el siguiente texto que omite el error anterior y añade varios más:

```
cpp: "g:\...\errores.c":6 Unterminated string or char const
Error "g:\...\errores.c": 7  illegal expression
Error "g:\...\errores.c": 8  syntax error; found `end of input'
expecting ')'
Warning "g:\...\errores.c": 8  printf expects a char pointer as
first arg, got pointer to void
```

```
Error "g:\...\errores.c": 8  Syntax error; missing semicolon
before  `end of input'
Error "g:\...\errores.c": 8  syntax error; found `end of input'
expecting '}'
```

El único error, además del nombre del identificador, es que se ha omitido la comilla que debía ir después de `%d`. El verdadero error es el primero que aparece en la línea 6: la constante de cadena no se ha finalizado porque falta la comilla. Pero esto provoca una serie de errores añadidos que arrastra ese primer error:

- La expresión es errónea, se ha llegado al final del programa y la llamada a la función `printf` es errónea (*ilegal expression*).
- Se ha llegado a la última línea del programa y se ha detectado que la llamada a la función `prinf` requiere cerrar el paréntesis.
- También se ha detectado un error en la función `printf` ya que requiere una cadena que no aparece.
- Todas las instrucciones en C requieren finalizar en un punto y coma (*semicolon*). Como no se ha cerrado la cadena, se entiende que el punto y coma que aparece pertenece todavía a la cadena.
- La función `main` require cerrar la llave (}), pero se ha llegado al final del archivo fuente y no se ha encontrado, ya que el compilador entiende que la llave existente pertenece todavía a la cadena.

Cerrando la cadena de forma correcta desaparecerían todos los errores que se han detectado.

Errores en tiempo de ejecución

A la hora de depurar un programa lo primero es solucionar todos los errores detectados en tiempo de compilación. Sin embargo, eso no quiere decir que el programa sea correcto. Puede que se haya creado sin problemas el programa ejecutable, pero que no realice la función para la que ha sido diseñado. Estos errores de detectan al ejecutar el programa, por lo que se llaman “errores en tiempo de ejecución”.

Detectar estos errores es más complicado, ya que el entorno no es capaz de averiguar la función para la que ha sido diseñado el programa y debe ser el programador el que realice un seguimiento del mismo para ver si todo funciona como debería ser. La forma más fácil de evitar estos errores es en la fase de diseño del algoritmo mediante una tabla de seguimiento que compruebe que el programa realiza los pasos necesarios, que las variables van tomando los valores previstos y que la salida es la prevista en el análisis del problema.

Si, a pesar de todo, el programa no realiza cometido de forma correcta, los entornos de programación suelen disponer de una herramienta, el depurador (*debugger*) que permite analizar el flujo de control del programa, comprobar si las distintas estructuras de control se comportan como se prevé y que las distintas variables toman los valores correctos.

1.4.2. El depurador de lcc

Un depurador es una herramienta que permite hacer una traza de un programa en tiempo de ejecución, lo que facilitará la detección de errores. El depurador permitirá ejecutar el

programa hasta un punto determinado del mismo, deteniéndose la ejecución o ejecutando las instrucciones paso a paso y permitiendo investigar el estado del mismo en ese punto.

Para iniciar el depurador hay que seleccionar la opción DEBUGGER del menú COMPILER (F5). Se abrirá entonces la ventana del depurador. En el menú principal aparece la opción DEBUG y en aparece en la pantalla la primera línea de código ejecutable (función `main`). En la parte inferior aparecerán distintas opciones para ver el estado de distintos aspectos del programa.

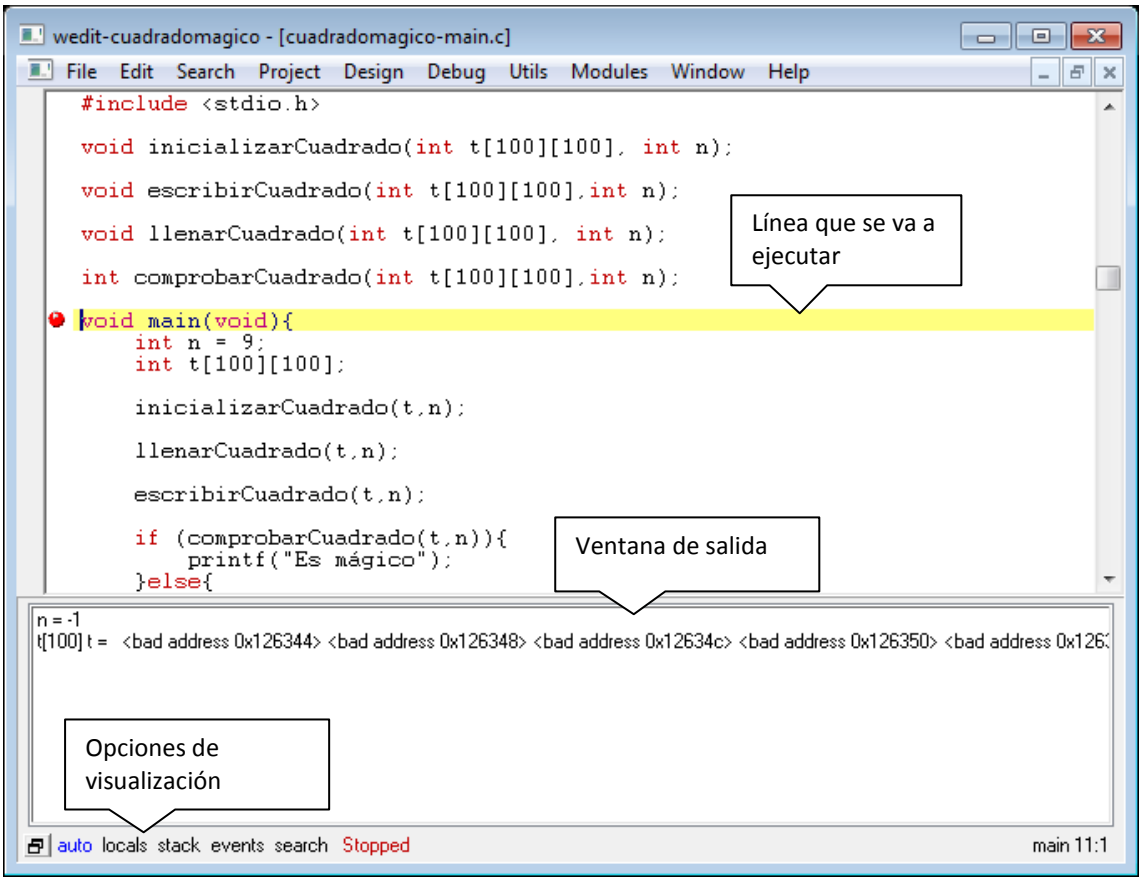

Figura 1.22. Ventana de depuración

Control de la ejecución del programa

En el menú DEBUG existen distintas opciones para controlar la ejecución del programa:

- EXECUTE (F5). Ejecuta el programa hasta el siguiente punto de interrupción (*breakpoint).*
- STEP IN (F8). Ejecuta la línea actual del programa. Si se trata de la llamada a una función, el control pasará a la primera línea de dicha función.
- SAME LEVEL (F4). Ejecuta la línea actual del programa. Si se trata de la llamada a una función, ejecuta dicha función entera y pasa a la siguiente instrucción dentro del mismo nivel.
- STEP OUT (F9). Ejecuta todas las instrucciones hasta el final de la función y retorna al punto dónde se ha llamado a dicha función.
- RUN TO CURSOR (F7). El depurador ejecuta el programa hasta la posición actual del cursor.

Pulsando con el botón derecho en una instrucción se puede seleccionar la opción SET NEXT STATEMENT. Esta opción permite que el control del programa salte a esa instrucción omitiendo todas las que se encuentran antes. Esta opción puede hacer que el programa falle por que se ha saltado alguna instrucción necesaria anterior.

También es posible salir del depurador con la opción STOP DEBUGGING, reiniciar el programa con RESTART o para la ejecución inmediatamente con BREAK (por ejemplo, si nos hemos metido en un bucle sin fin).

Puntos de interrupción (*breakpoint*)

Un punto de interrupción es un lugar del programa fuente dónde se detendrá siempre la ejecución y que se podrá utilizar para ver el estado del programa en un punto determinado del mismo. Para establecer un punto de interrupción se situa el cursor en el punto deseado y se selecciona la opción BREAKPOINT del menú DEBUG (F2).

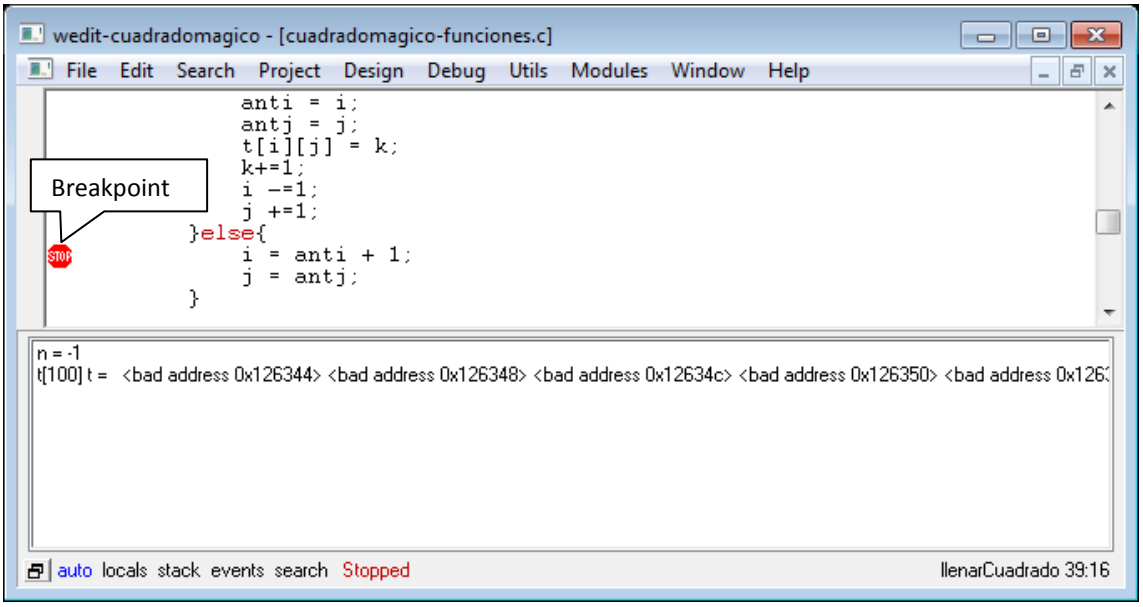

Figura 1.23. Punto de interrupción en un programa

Cuando el programa se ha detenido en dicho punto, es posible quitar el *breakpoint* pulsando de nuevo F2.

Visualizar el estado del programa

La ventana inferior muestra el estado de distintos aspectos del programa. Permite seleccionar distintos elementos. En lo que se refiere al valor de las variables:

- **auto**. El depurador intenta determinar las variables que se están utilizando en ese momento y muestra sus valores.

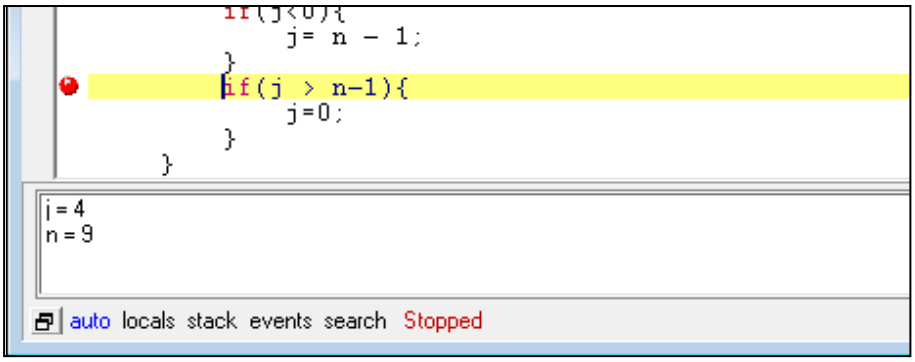

Figura 1.24. La opción auto muestra las variables j y n, ya que son las que se están utilizando en ese momento

- **locals**. Se muestran las variables locales de la función actual.

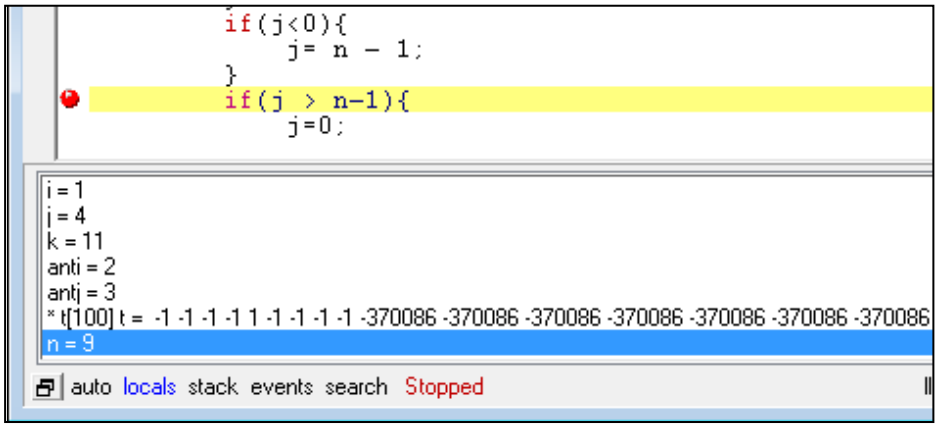

Figura 1.25. La opción locals muestra las variables locales de la función llenarCuadrado

También podemos ver el valor de las variables con la opción WATCHS. WATCHS abre una ventana secundaria donde es posible teclear el nombre de una variable y ver su valor de forma permanente. Para insertar una nueva variable en la ventana hay que pulsar la tecla INSERT. Para eliminarla se pulsa la tecla DELETE.

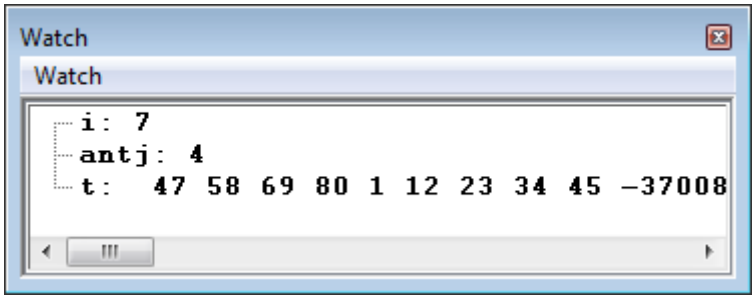

Figura 1.26. Ventana Watch

1.5. Ejercicios

1. Cree un proyecto, `cudradomagico`, que incluya los programas fuente `cuadradomagico.c` y `cuadradomagico-funciones.c`.

2. Detecte y solucione los errores del código presente en los programas fuente.
3. Ejecute y pruebe el programa.
4. Compile, realice el enlace y ejecute el programa cuadradomagico desde la línea de órdenes del sistema.
5. Modifique el programa de forma que la variable `n` tome valor 15.
6. Utilizando el depurador, obtenga el valor de las variables locales de la función `llenarCuadrado` en la línea 37 de programa cuando `k` sea igual a 8.