Tutorial breve de DevCpp para las prácticas de la asignatura 'Fundamentos de Informática', EPSIG

José Otero Rodríguez

Índice general

Capítulo 1

Introducción

Devcpp es un IDE (Integrated Development Environment) para los lenguajes de programación C/C++. El propósito de un entorno de este tipo es facilitar la escritura de programas (en lenguaje C++ en el caso de esta asignatura), su compilación y depuración. La compilación es el proceso mediante el cual una aplicación informática, denominada compilador, convierte el fichero de texto conteniendo las órdenes escritas en un lenguaje de programación en un programa ejecutable en un determinado tipo de ordenador. El fichero que contiene esas órdenes y que ha sido tecleado por un programador/a se denomina fichero fuente. El programa ejecutable puede llevarse a otro ordenador del mismo tipo y ejecutarse en él, sin necesidad de que en ese ordenador exista compilador alguno y sin necesidad de compilarlo de nuevo. Los lenguajes de programación modernos también permiten compilar el mismo fichero fuente en distintos tipos de ordenadores, de modo que se producen programas ejecutables para cada uno de esos tipos de ordenadores. La depuración consiste en la prueba del programa, incluso paso a paso (parándose en cada instrucción).

El entorno permite la escritura de programas en dichos lenguajes mediante el editor de texto incorporado. Dado lo específico de este editor, incorpora ciertas características útiles cuando se escribe un programa, como la numeración de líneas, el resaltado de la sintaxis, destaca los paréntesis y llaves que están emparejados, etc. Además mantiene muchas de las características de cualquier editor de textos, se puede copiar/cortar y pegar texto, sobre escribir o insertar de la misma forma que en cualquier programa de ese tipo. Las teclas de Inicio, Fin, Avance de Página, Retroceso de Página, etc. mantienen la misma funcionalidad. Lo mismo sucede con las teclas de desplazamiento del cursor. Las teclas de Suprimir y Retroceso sirven para borrar carácter a carácter, como es usual. El tipo de letra utilizado en el editor no se guarda con en el fichero en el que se guarda el código fuente, tiene una importancia secundaria, no afecta al resultado de la compilación. La elección del tipo, tamaño y color se puede realizar libremente, pero es útil que tenga un ancho constante y que su color se destaque sobre el fondo. El hecho de poseer un ancho constante facilita el examen de un programa por columnas, lo cual es deseable en ocasiones. Cuando el programa alcanza cierta envergadura e incluya funciones y clases, existe la posibilidad de navegar por el código fuente utilizando el browser de clases.

Desde el mismo entorno se puede compilar el programa, y ejecutarlo, de modo que la prueba del mismo resulta muy cómoda. También es posible depurar el programa, ejecutarlo paso a paso, consultar los valores de las variables que intervienen en los cálculos aunque no se muestren por la pantalla, etc.

Este entorno incorpora otra serie de características que lo hacen apto para el desarrollo de programas en equipo, aunque su uso cae fuera del ámbito de este curso.

1.1. Notación

El entorno Devcpp posee diversos menús y botones que son altamente configurables, de modo que se puede cambiar completamente el aspecto del mismo. Si bien esto permite adaptarlo a los gustos y necesidades del usuario/a, dificulta la redacción de un manual válido para cualquier configuración posible. Por ese motivo en esa guía se explica como manejar el programa utilizando menús y no botones. Sin embargo, el lector/a puede fácilmente aprender por si mismo que botones debe pulsar para realizar las distintas tareas necesarias para completar las prácticas de la asignatura, sin más que identificar los 'atajos' (notas amarillas que aparecen al desplazar el cursor por encima de los botones) con las tareas y correspondientes menús que se explican aquí. El programa posee ficheros de internacionalización en distintos idiomas, lo que permite que los distintos menús aparezcan en el idioma que se desee. Los idiomas oficiales en las distintas comunidades autónomas

españolas están también incluidos.

Los sucesivos menús que se deben desplegar para realizar las distintas tareas se representan en este tutorial mediante el nombre de los menús y opciones separados por `->` , por ejemplo `Menu->opcion1->otra_opcion`. Se utilizarán los nombres en castellano para los distintos menús y opciones.

Las combinaciones de teclas se representan mediante el nombre de la tecla o el carácter que representan separados por '-', por ejemplo 'CTRL-N' significa pulsar la tecla control y sin soltarla pulsar también la tecla N.

Muchas de las tareas tienen asociadas tanto opciones de menús como combinaciones de teclas.

El entorno muestra la combinación de teclas correspondiente a los menús a continuación de cada opción y como un pequeño letrero amarillo que aparece si pasamos el cursor por encima de los botones del entorno, sin hacer clic.

1.2. Signos especiales

Existen varios signos que no son letras, dígitos o signos de puntuación de los utilizados habitualmente. Para teclear alguno de ellos es necesaria la pulsación de varias teclas.

- El signo # (almohadilla) se teclea presionando la tecla 'ALT-GR' (a la derecha de la barra espaciadora) y sin soltarla se presiona la tecla en donde figura el dígito 3.

- El signo '|' (barra vertical, pipe en inglés) se obtiene de la misma forma pero presionando la tecla del dígito 1 en lugar del 3.

- La vírgula '~' se obtiene con la combinación de teclas 'ALT-GR' y 4.

- Las llaves abierta '{' y cerrada '}' están entre la 'ñ' y la tecla de return, para que aparezcan en el editor es necesario pulsar también la tecla 'ALT-GR' al igual que en el caso anterior.

- Lo mismo sucede con los corchetes, los caracteres '[' y ']', que están entre la 'p' y la tecla return.

- Los signos de puntuación también son utilizados en los programas. Por ejemplo la exclamación y la interrogación cerradas, los caracteres '!' y '?' respectivamente, que se obtienen de la forma usual, se pulsa la tecla de mayúsculas y sin soltarla la tecla en donde aparezcan. Los mismos caracteres abiertos no se usan en C++ como parte de las instrucciones.

- El resto de los signos de puntuación si se usan en C++ como parte de las instrucciones: comillas dobles, simples, el punto, los dos puntos, el punto y coma así como los paréntesis.

- Adicionalmente, los caracteres '&' (y inglesa, abreviatura de and) y '_' (guión bajo o subrayado) también son empleados. El primero está en la tecla del dígito 6 y el segundo a la derecha del punto.

- Los símbolos de las operaciones aritméticas se usan en C++ para denotar las mismas operaciones (y alguna otra en algún caso), junto con el carácter del tanto por ciento '%' que representa una operación aritmética poco habitual fuera de la informática.

Capítulo 2

Primeros pasos con Devcpp

El entorno se ejecuta haciendo doble clic sobre el icono correspondiente en el escritorio, un clic si está en el menú de inicio o bien se despliega el menú 'Programas' de windows hasta que aparezca el icono del programa y entonces se hace clic en el. Si es la primera vez que se ejecuta, se realizan una serie de preguntas. La más relevante es el idioma, si va a seguir este tutorial, seleccione 'Español(Castellano)'.

Al ejecutar el programa, se muestra una imagen de 'splash' a modo de presentación, mientras se inicializa el programa y a continuación una ventana con un recordatorio sobre un aspecto particular de su uso, es una forma de aprender su funcionamiento día a día, puede desactivarse. Al cerrar esa ventana aparece el entorno vacío, tal y como se ve en la figura 2.1.

Desde este punto comenzaremos a explicar el funcionamiento del programa.

En este curso los programas muy probablemente constarán de un solo fichero fuente. Para comenzar a escribir un programa existen dos alternativas:

- `Archivo->Nuevo->Código Fuente` o bien CTRL-N
- `Archivo->Nuevo->Proyecto` y seleccionar 'Aplicación de Consola' y 'En C++'

Si se utiliza la primera de las alternativas, aparecerá una ventana con una pestaña en la que aparece el nombre 'SinNombre' posiblemente seguido de algún número si no es el primer programa que escribimos en la misma sesión de trabajo. El entorno toma ahora el aspecto de la figura 2.2

En la parte de la derecha, en donde aparece el cursor, es en donde se escribe el programa que vamos a compilar. La primera vez que guardamos un fichero fuente, por defecto, podemos cambiar su nombre. Si deseamos guardarlo con otro nombre en otra ocasión (después de haberlo guardado más veces) seleccionaremos `Archivo->Guardar Como...`. La extensión del archivo puede de ser '.cpp', de entre otras que representan programas en c++, también válidas. Suponga que hemos llamado al fichero 'hola.cpp' y que se ha escrito lo siguiente (para iniciar una nueva línea en la ventana del editor, pulse return, la tecla que muestra un ángulo recto terminado en una flecha):

```
#include<iostream>
using namespace std;
main()
{
   cout<<"Hola"<<endl;
}
```

En la figura 2.3 se puede apreciar el aspecto del programa en la ventana del editor.

Si deseamos compilar el programa pulsaremos el icono compilar, la combinación de teclas CTRL-F9 o seleccionaremos `Ejecutar->Compilar`. Durante el proceso aparece una ventana que informa del progreso de compilación (figura 2.4). Si no hay errores aparecerá una ventana con el mensaje 'Done' y un único botón con la leyenda 'Cerrar' (figura 2.5).

Ahora se puede ejecutar el programa pulsando el icono ejecutar, la combinación de teclas CTRL-F10 o `Ejecutar->Ejecutar`. Aparecerá una ventana con el fondo negro (una consola de MSDOS) y se cerrará rápidamente, de modo que lo más probable es que no veamos nada. Para corregir esto añadimos la línea `system("PAUSE")` al programa, de modo que quedaría como se ve a continuación (para añadir una línea entre dos existentes, sitúese al principio de la línea siguiente o al final de la anterior y pulse return):

```
#include<iostream>
using namespace std;
```

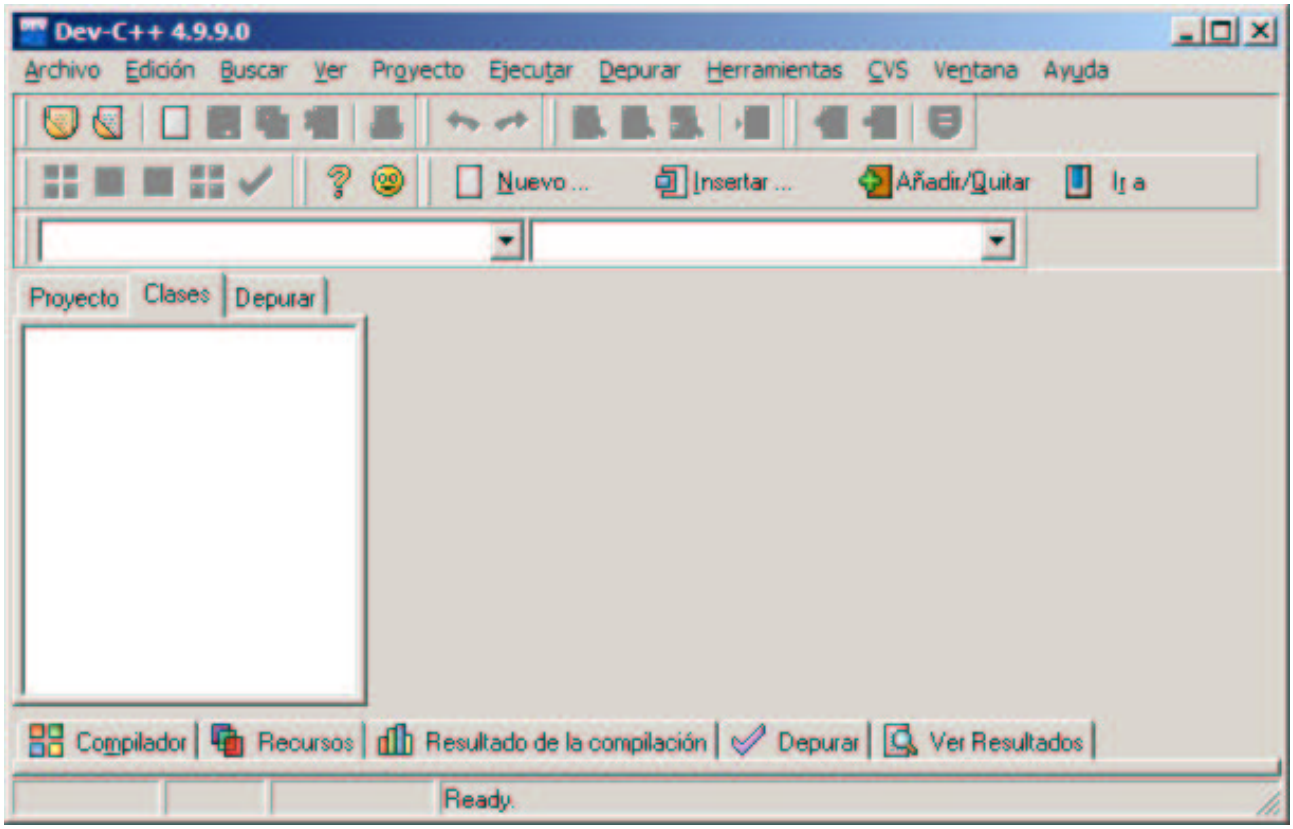

Figura 2.1: IDE tras arrancar, sin ningún programa en edición

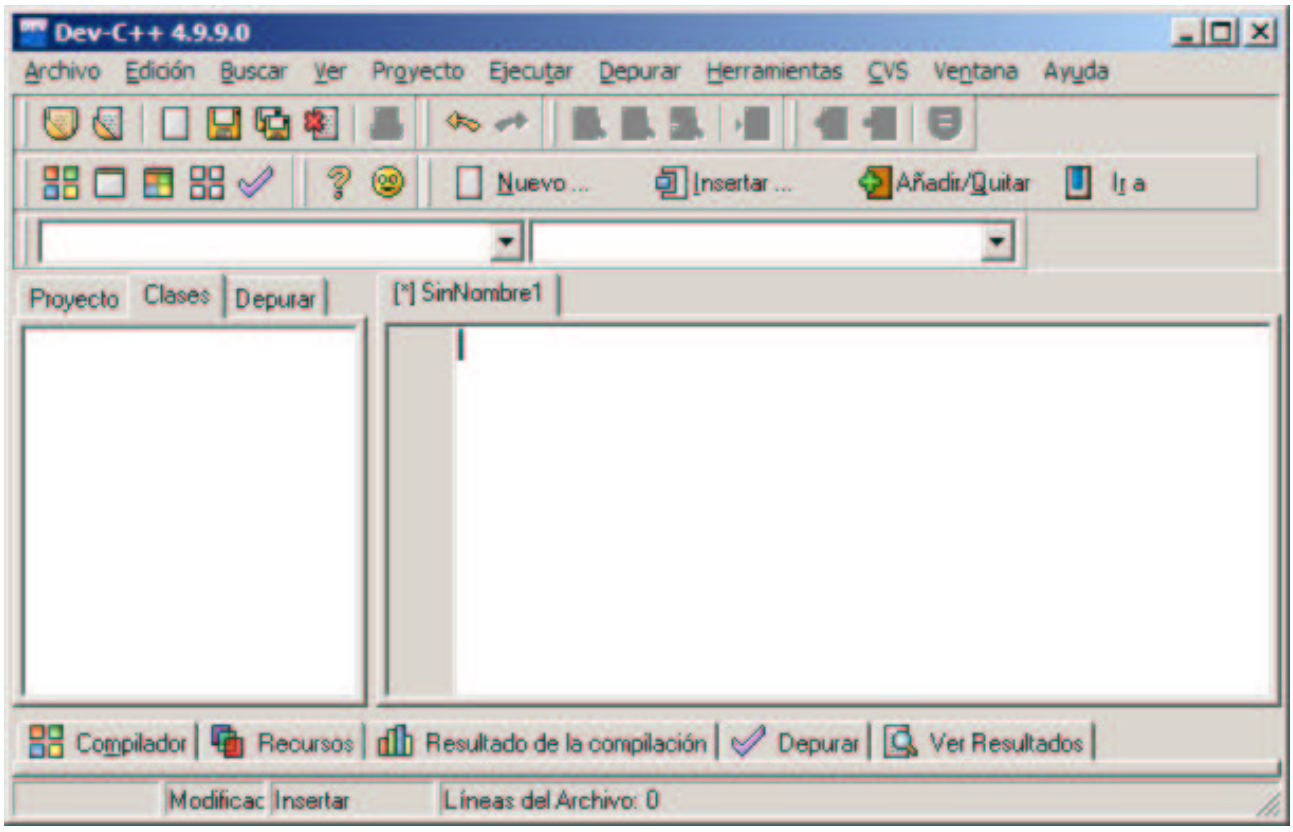

Figura 2.2: IDE tras la creación de un nuevo fichero fuente

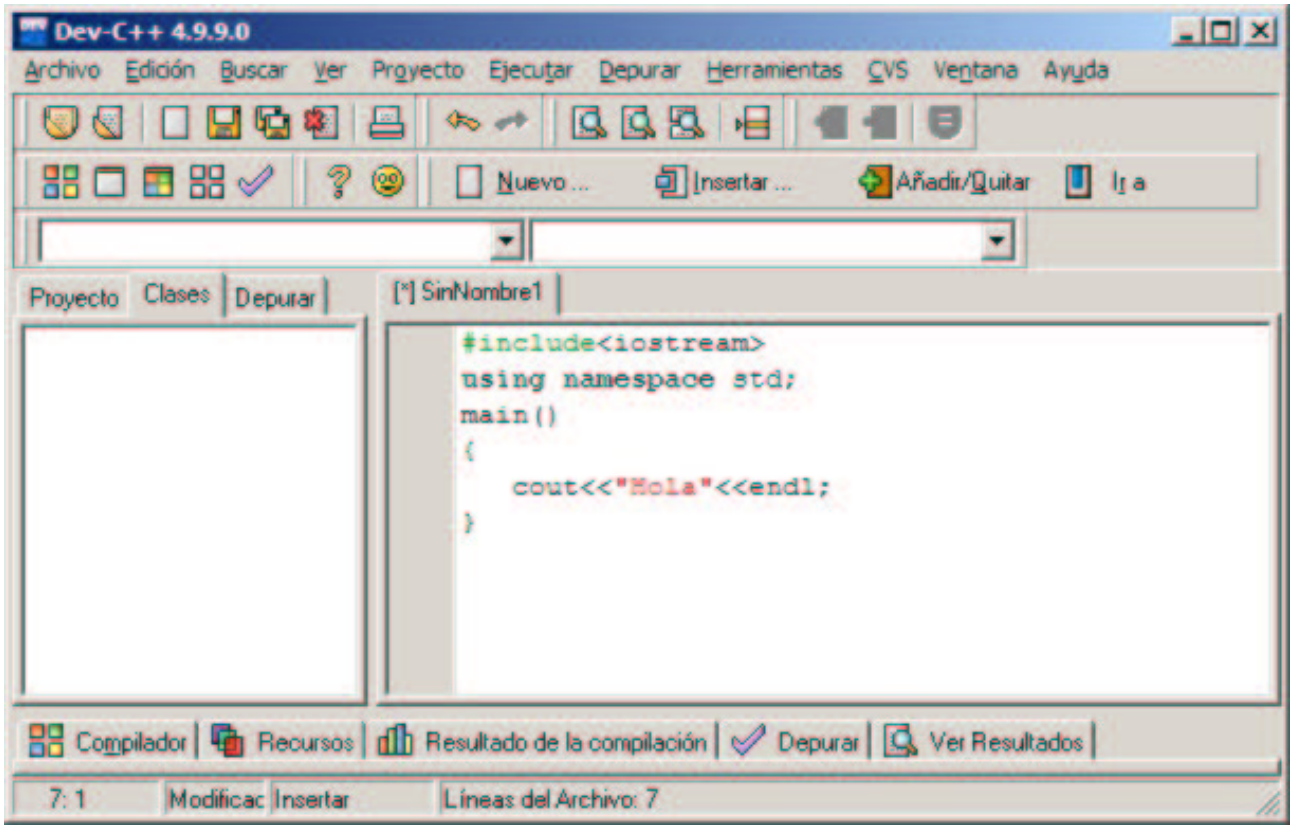

Figura 2.3: IDE tras la escritura de un programa

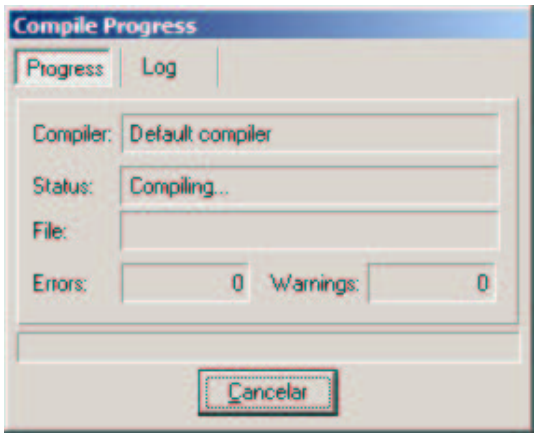

Figura 2.4: Información durante la compilación

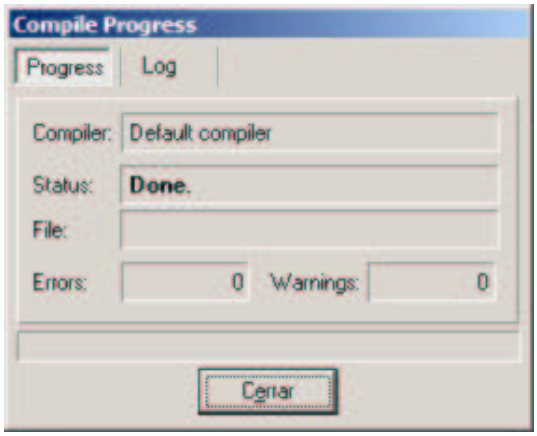

Figura 2.5: Información tras la compilación

```
main()
{
   cout<<"Hola"<<endl;
   system("PAUSE");
}
```

Como hemos modificado el programa, hemos de compilarlo de nuevo, para que en el programa ejecutable que genera el compilador se reflejen los cambios realizados. Es conveniente salvar el programa antes de compilar. Una vez compilado el programa, podemos ejecutarlo y en este caso aparecerá una ventana (por defecto con el fondo negro y los caracteres en blanco) en la que se leerá:

```
Hola
Presione una tecla para continuar . . .
```

Efectivamente si pulsamos una tecla, la ventana desaparece. Es importante notar que mientras la ventana con la salida del programa (la consola de MSDOS) no se cierra, no se puede compilar el programa ni ejecutarlo de nuevo, los botones correspondientes aparecerán en gris.

Capítulo 3

Corrección de errores

Es habitual que al escribir un programa en C++ o en cualquier otro lenguaje se cometan distintos errores. Estos pueden ser de distintos tipos, los más habituales en este curso serán:

- Se ha escrito de forma incorrecta una instrucción.
- Se ha omitido algún signo o instrucción.
- Se ha escrito una instrucción en un lugar incorrecto.
- Las instrucciones que se han escrito no producen el resultado correcto por no describir el cálculo correspondiente.

En el caso de los tres primeros tipos, el proceso de compilación no se completa, el compilador genera al menos un error y nos informa de cual es la línea en la que se produce. Es destacable el hecho de que puede dar la impresión de que la línea en donde se produce el error (según el programador/a) y aquella indicada por el compilador difieren. Este hecho se produce por la lógica estricta con la que funciona el compilador, que difiere de la que utiliza un programador/a en la fase de aprendizaje. También es importante que desde que se produce el primer error, el compilador sigue analizando el programa, sin embargo el significado de las instrucciones se ve alterado por ese error inicial, de modo que muy probablemente aparezcan errores que son desencadenados por el primero. Por esta razón, salvo en los casos más evidentes, es recomendable corregir los errores comenzando por los que se producen en primer lugar. También es posible que un error enmascare otros. Evidentemente, tras corregir los errores (en el texto del programa, en el código fuente) es necesario compilar de nuevo el programa, para que el compilador pueda progresar más allá del punto en donde se produjeron los primeros errores y examinar el significado de las instrucciones sin la alteración producida por aquellos.

En el caso del último de los tipos de error que figuran más arriba, el compilador no informa de ningún error, sin embargo a la hora de probar el programa, observaremos que el resultado no es correcto. Esto se debe a que la secuencia de órdenes que hemos escrito no describe con exactitud el cálculo que debíamos realizar. En este caso debemos examinar el código que hemos escrito y modificarlo de modo que la secuencia de instrucciones se corresponda con los cálculos que deseamos realizar. De nuevo una vez realizadas las modificaciones es necesario compilar el programa para que estas modificaciones se reflejen en el ejecutable que se produzca.

El compilador informa de los errores encontrados en la parte inferior del entorno. Cuando es posible, el cursor del editor se desplaza automáticamente a la línea en donde se produce el error, sin más que hacer doble clic encima del informe del error. Es importante tener en cuenta que la descripción de los errores que da el compilador puede ser difícil de entender, la realización de varios ejercicios permite aprender a identificar los distintos errores a partir de la descripción.

Finalmente, es posible que el programador/a escriba una instrucción que en sí no es incorrecta, pero que resulte 'sospechosa'. La mayor parte de estas situaciones de este estilo son examinadas por el compilador y en este caso no se informa de un error sino de un 'warning', un aviso, reclamando la atención del programador/a sobre una línea concreta, al objeto de que 'repase' esa parte del código.

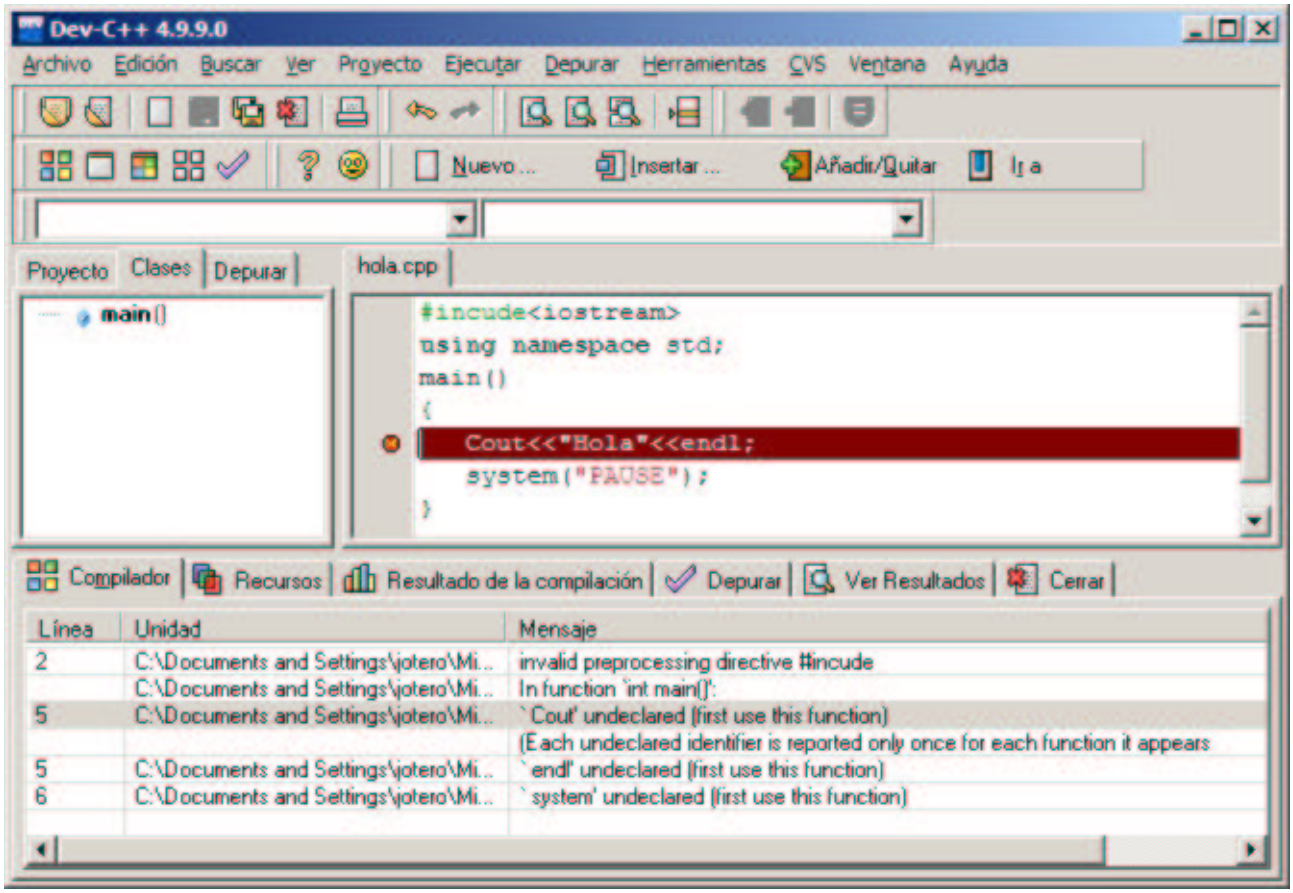

Figura 3.1: Programa incorrecto, algunas instrucciones no están bien escritas

3.1. Ejemplos

3.1.1. Escritura incorrecta de instrucciones

En el programa que se muestra en la figura 3.1 se han cometido dos errores, en lugar de escribir 'include' se ha escrito 'incude', en lugar de 'cout' se ha escrito 'Cout'. Al compilar el programa aparecen varios mensajes de error en la parte inferior del entorno. Cuando se hace doble clic encima de un mensaje de error, se resalta la línea en la que se produce este, como se puede ver en la figura 3.1.
En este caso un error en el programa desencadena otros más adelante, como la directiva del preprocesador no está escrita correctamente, el compilador entiende que 'endl' y 'system' no están declarados. Por esta razón es conveniente corregir los errores comenzando por los que ocurren antes. Por ejemplo si se corrige el primer error y se vuelve a compilar el resultado se puede ver en la figura 3.2. Como se aprecia, ahora se detecta únicamente el error consistente en escribir 'Cout' en lugar de 'cout', han desaparecido los otros dos que, realmente no eran tales.
Cuando el proceso llega a su fin sin ningún aviso de error, aparece la ventana (de la que ya hemos hablado) con un solo botón informando de que se ha completado la compilación y del número de errores (cero). El usuario/a ha de hacer clic en ese botón para que se cierre esa ventana.
Haciendo clic en la pestaña con la leyenda 'Compilador', aparece vacío el lugar en donde se informa de los errores, como se aprecia en la figura 3.3.

Se obtienen más detalles del proceso de compilación si se hace clic en la pestaña 'Resultado de la compilación', mostrando la traza (los mensajes) del propio programa compilador, como se aprecia en la figura 3.4.

3.1.2. Omisión de un signo o instrucción

En la figura 3.5 se muestra un ejemplo del segundo tipo de errores, omisión de un signo o instrucción.
En este caso se ha omitido el carácter #,un punto y coma y la llave final. Como ya hemos dicho, es

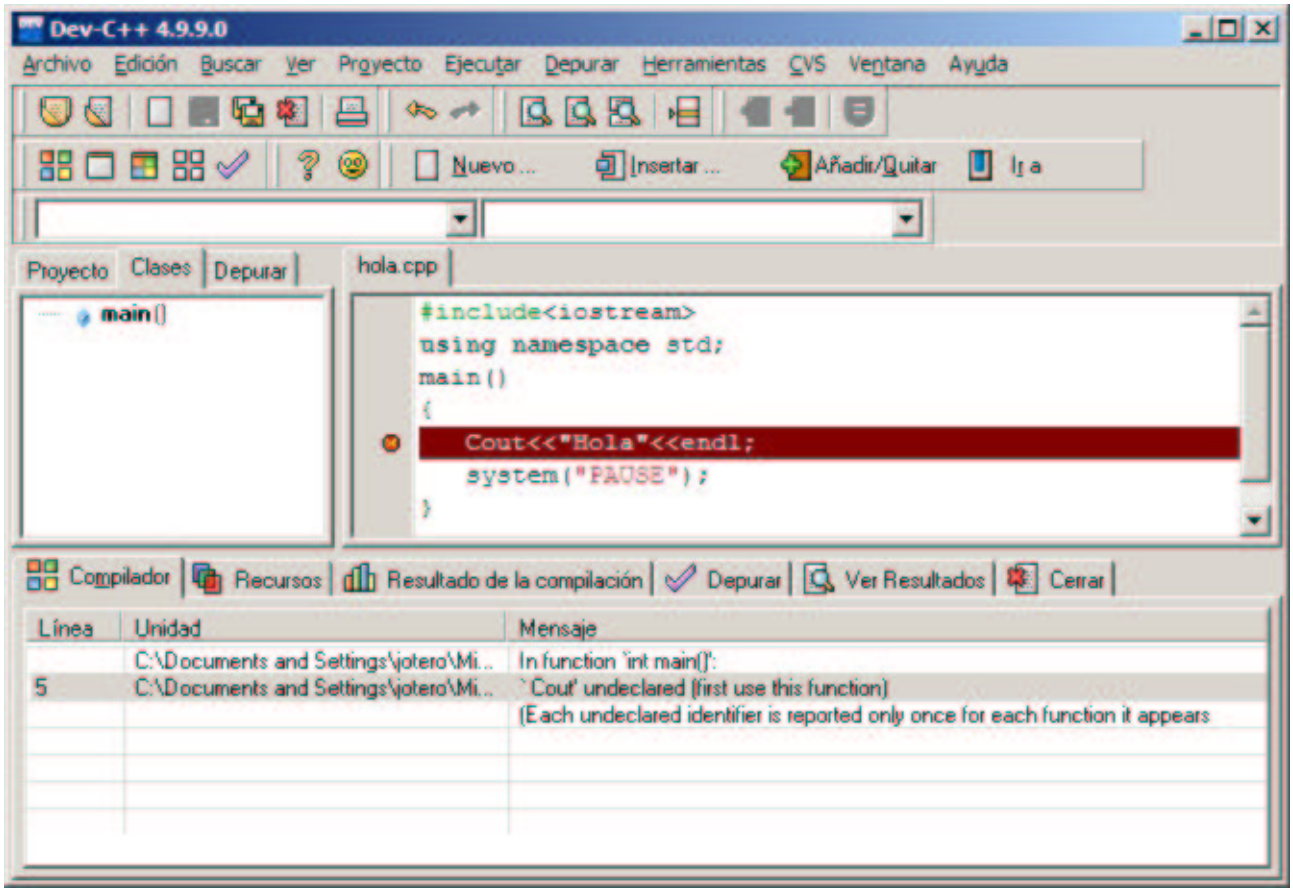

Figura 3.2: El mismo programa que el de la figura 3.1, después de corregir el primer error

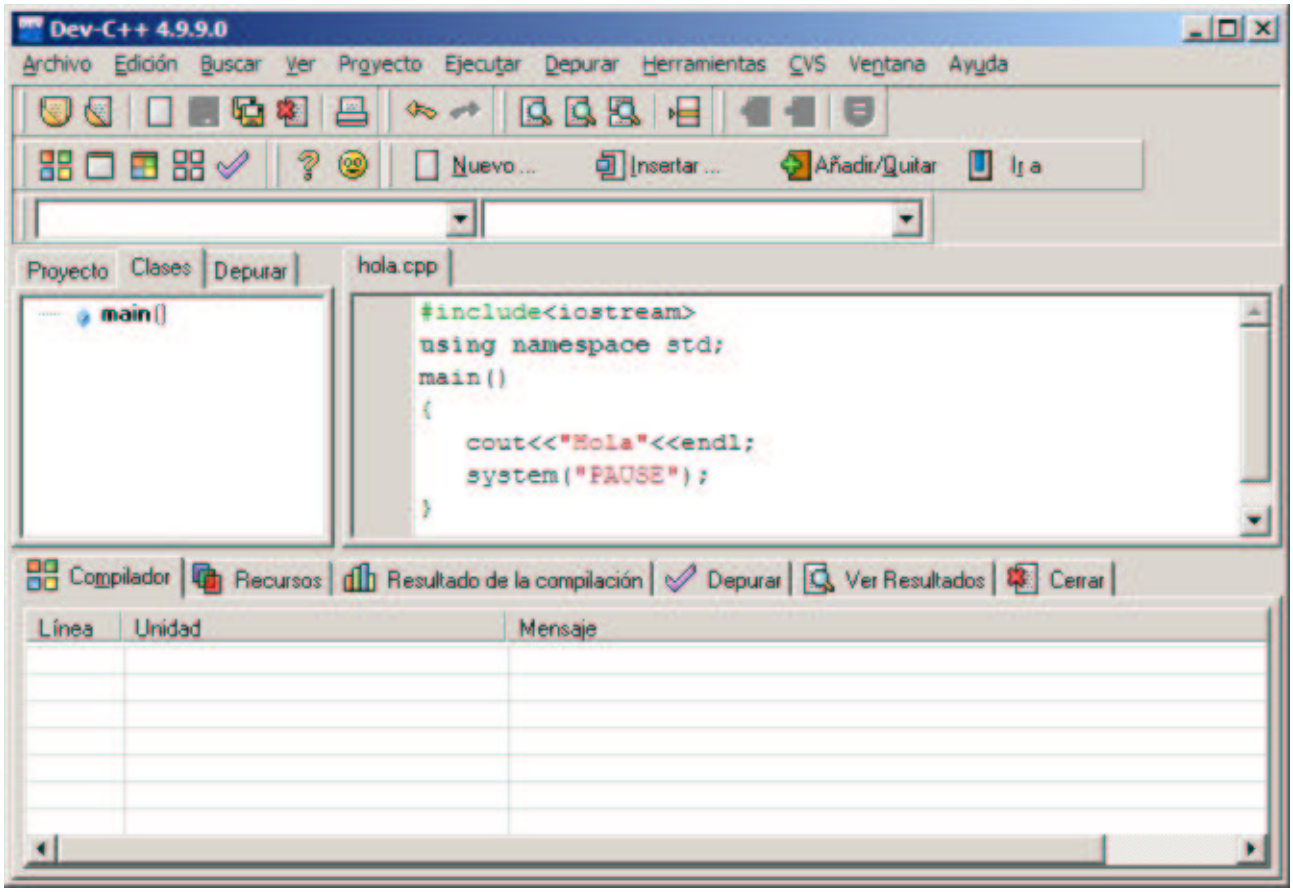

Figura 3.3: El mismo programa que el de la figura 3.2, después de corregir todos los errores

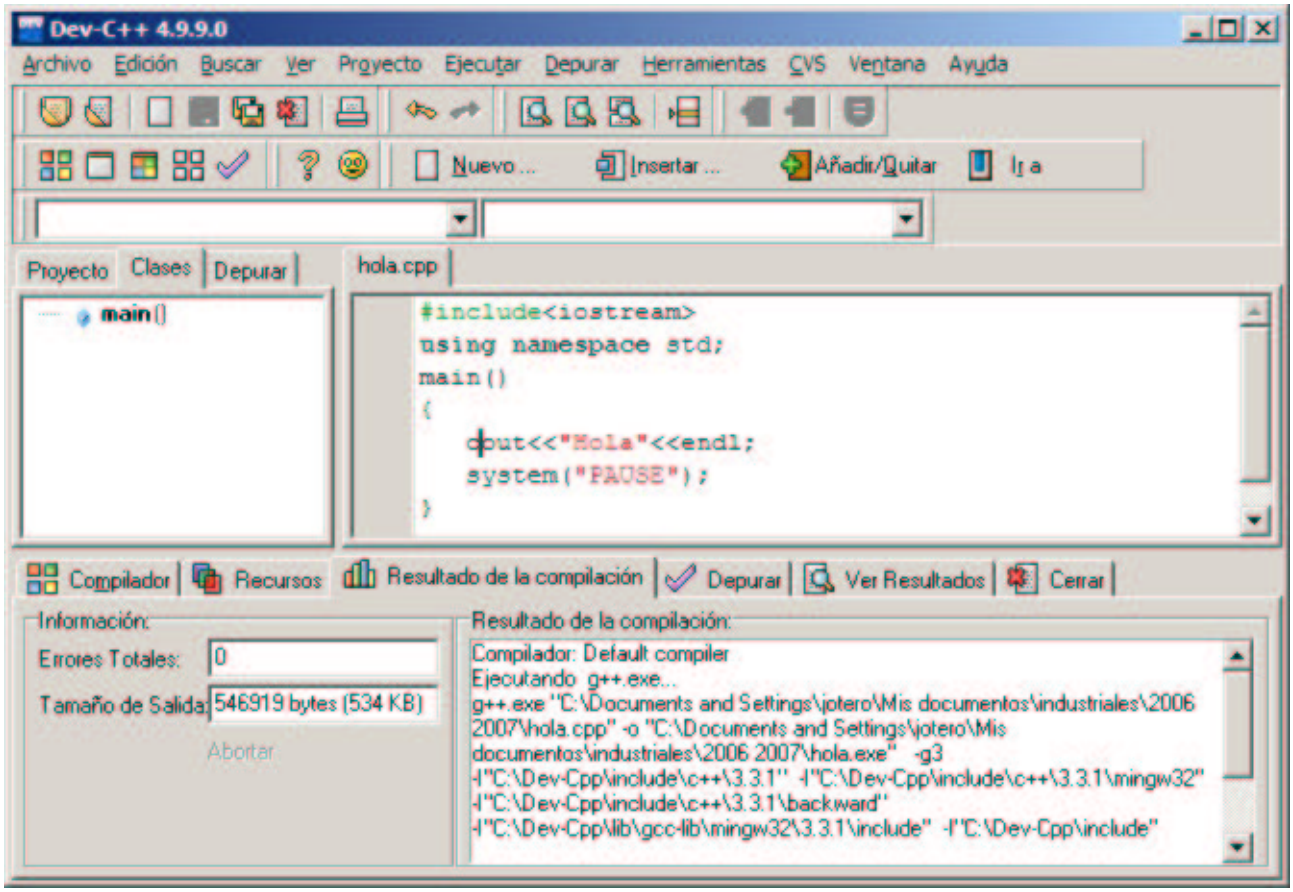

Figura 3.4: Traza del compilador en la pestaña 'Resultado de la compilación'

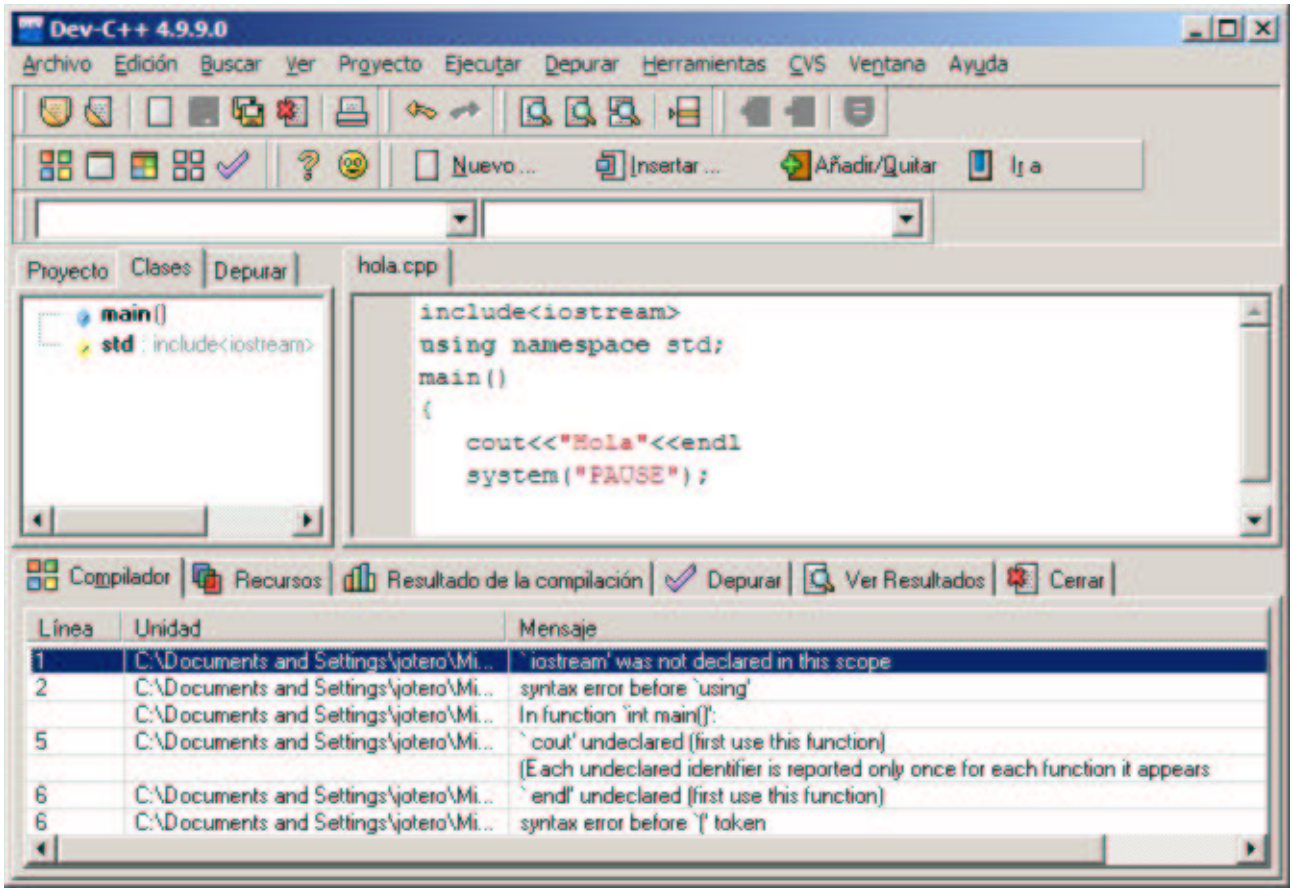

Figura 3.5: En este programa faltan varios caracteres

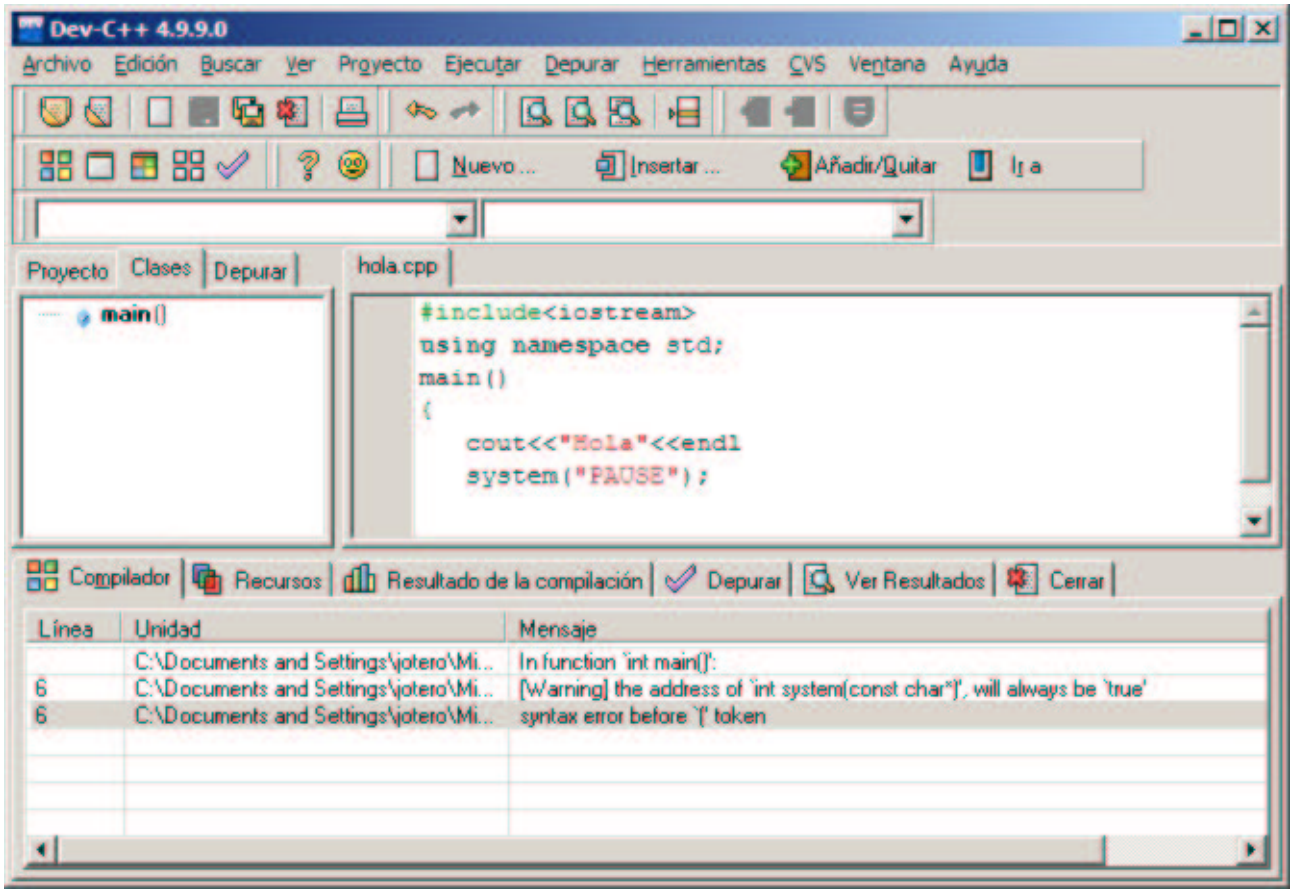

Figura 3.6: El mismo programa que en la figura 3.5 después de añadir el # que faltaba

conveniente comenzar a corregir los errores por el primero que se produce ya que este puede enmascarar otros más abajo, como se puede ver en la imagen anterior, no aparecen mensajes relacionados con la ausencia de al '}' final. Si se corrige la ausencia del # y se recompila, aparece un error antes del paréntesis de 'system'. Puesto que desde ese punto hasta el principio de la línea todo está correcto, el error estará en la línea anterior. Se trata del punto y coma final de la línea anterior. Esta es una circunstancia habitual al compilar programas con errores.

De nuevo se corrige este error y se recompila. Aparece un error al final del programa (ver figura 3.7), en este caso se trata de la '}' que cierra el mismo. Si se corrige este error, añadiendo la '}' en la última línea, el programa compila sin errores.

3.1.3. Instrucciones escritas en un lugar incorrecto

El siguiente tipo de error se ilustra el ejemplo de la figura 3.8 en el que la última de las instrucciones se escribe fuera de la llave final, aparecen varios errores debidos a esta cuestión que desaparecen al corregir el programa de forma adecuada.

En este caso la interpretación del error es más difícil, es mas, aparece una línea de un fichero fuente que no hemos escrito nosotros, forma parte de las librerías que se distribuyen junto con el compilador. Si hacemos doble clic en esa línea se abriría ese fichero e incluso podríamos modificarlo (si lo permiten los permisos del archivo), de modo que no es conveniente hacerlo.

3.1.4. Errores producidos por codificación incorrecta del algoritmo

Para comprender mejor el último de los tipos de error, se acompaña la explicación de un ejemplo en el que se pretende escribir un programa que pida dos números y muestre su producto. Supongamos que el programador/a confunde el signo '+' con el '*', lo cual no es difícil, al estar en el mismo lugar del teclado.

Como se ve (figura 3.9) el compilador no muestra ningún error, ya que este obviamente no puede adivinar la intención del programador/a.

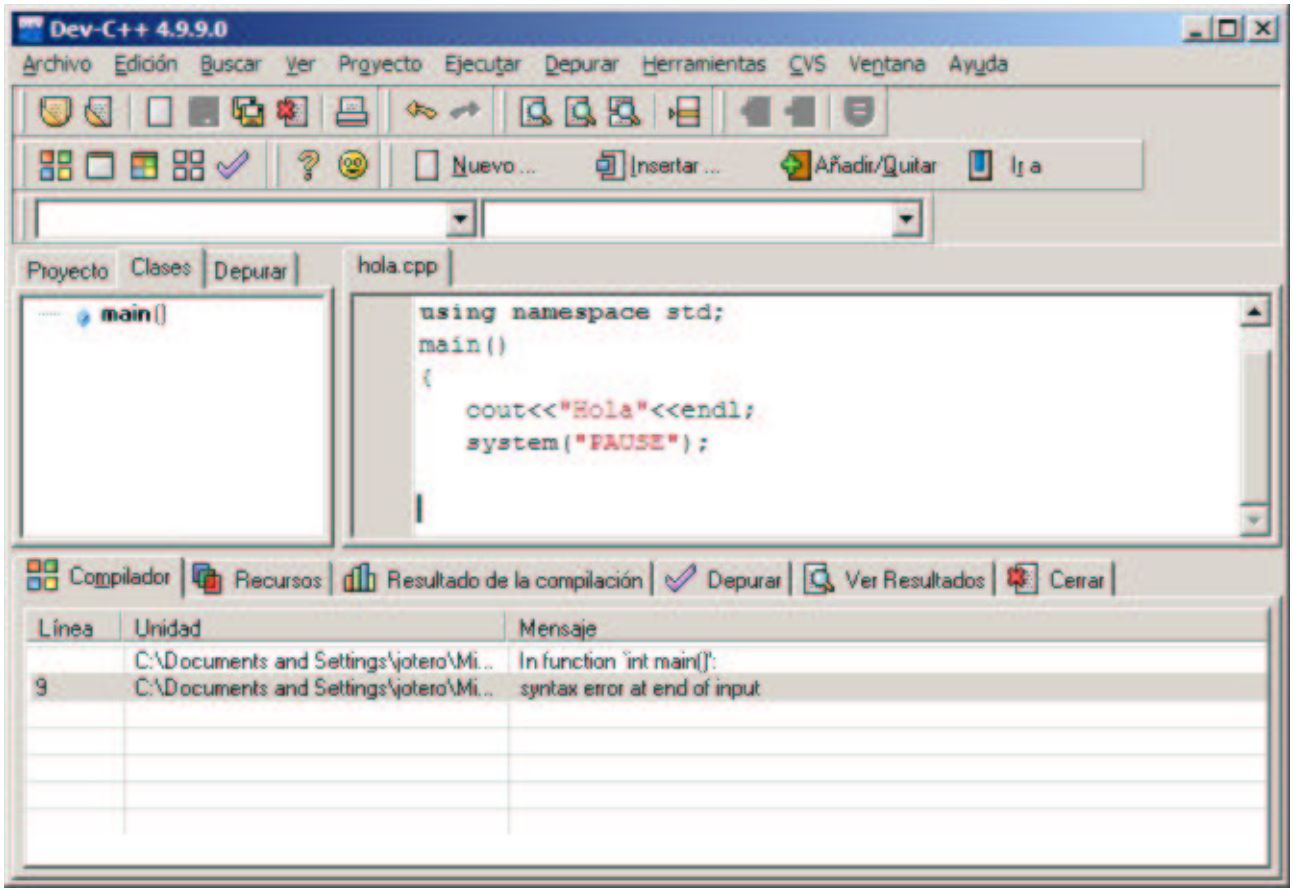

Figura 3.7: El mismo programa que en la figura 3.6, observar el error que produce la llave que falta

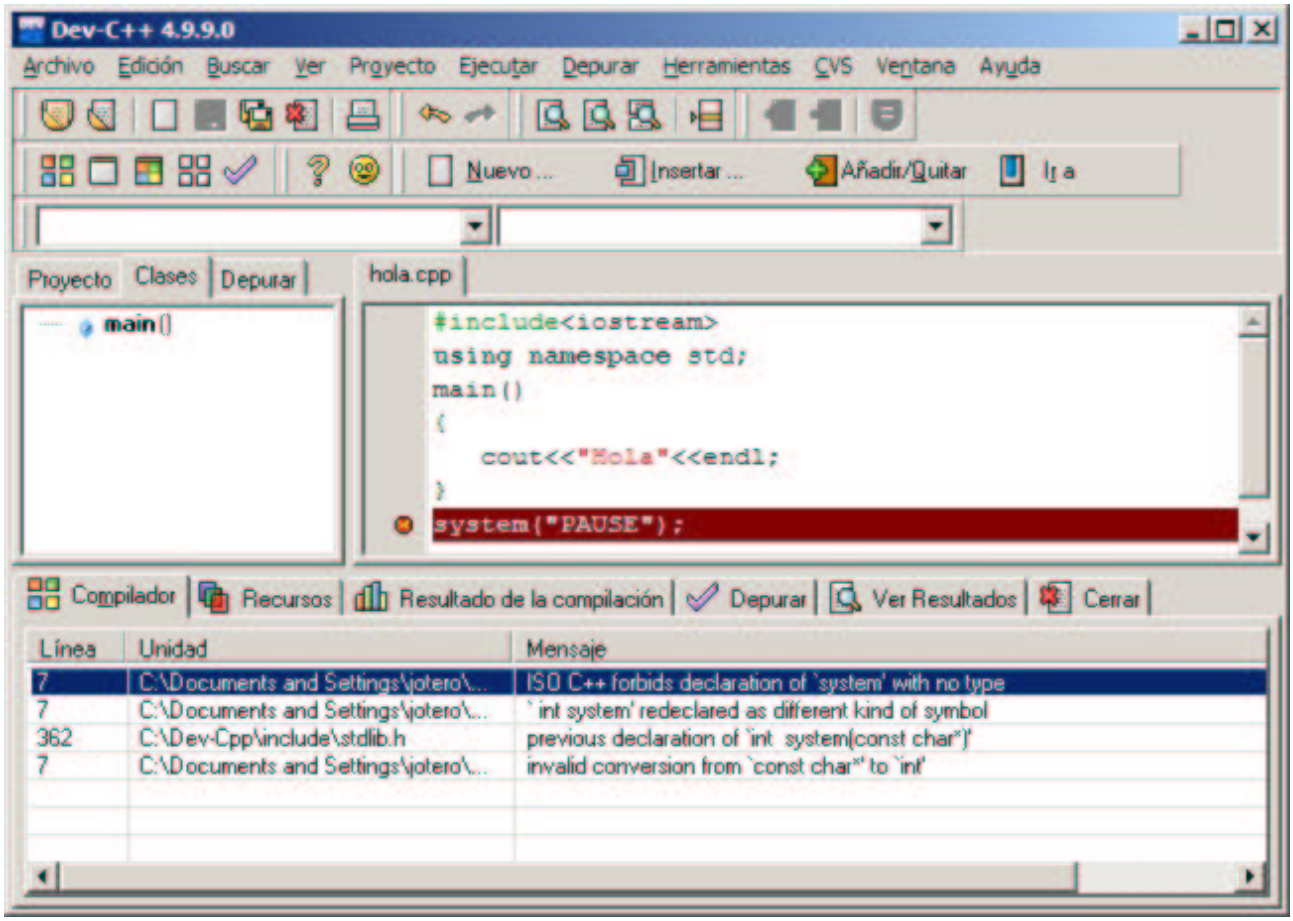

Figura 3.8: Error producido por una instrucción escrita en un lugar incorrecto.

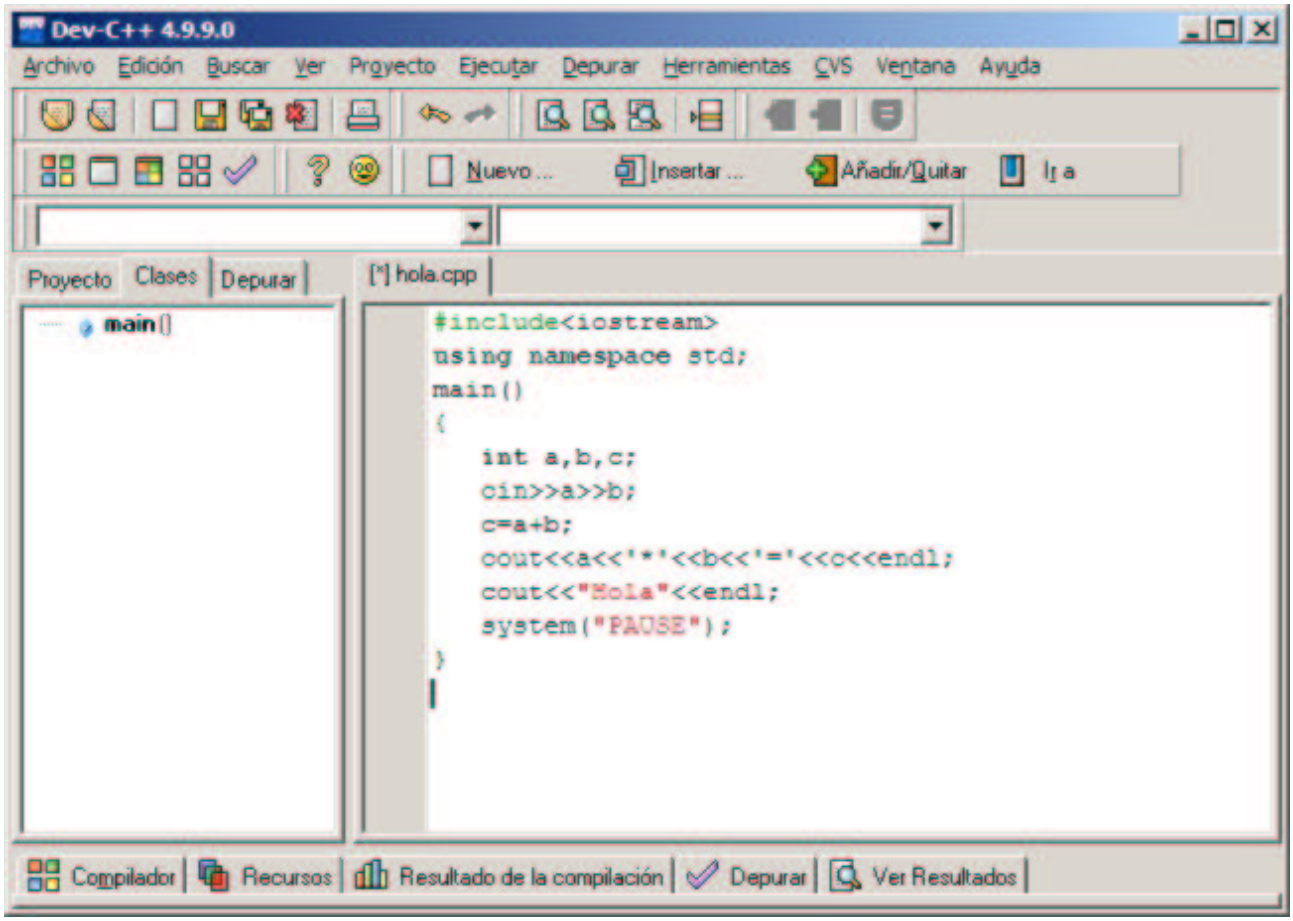

Figura 3.9: Un programa que compila pero que no produce el resultado deseado.

Al probar el programa sin embargo, este/a debería darse cuenta de que el resultado no es correcto:

```
4 5
4*5=9
Hola
Presione una tecla para continuar . . .
```

El programador/a deberá deducir que parte de los cálculos no han sido codificados en C++ correctamente. En este caso es muy sencillo pero no siempre será así.