

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA Y SISTEMAS DE TELECOMUNICACIÓN

PROGRAMACIÓN I

Manual de uso del depurador de VSCode

ÍNDICE

1.- INTRODUCCIÓN	3
2.- PREPARACIÓN DEL ENTORNO PARA USAR EL DEPURADOR.....	4
3.- PRIMER EJEMPLO: PROGRAMA PRINCIPAL SIN FUNCIONES AUXILIARES	4
3.1.- EJECUCIÓN SIN DEPURACIÓN	6
3.2.- EJECUCIÓN CON DEPURACIÓN	7
4.- EJEMPLO DE DEPURACIÓN CUANDO HAY VARIAS FUNCIONES.....	10

1.- Introducción

Como se explica en las clases de teoría, cuando hacemos un programa seguimos las siguientes fases:

1. Diseño
2. Codificación
3. Compilación
4. Depuración y pruebas
5. Ejecución

VSCode ofrece una herramienta, llamada *depurador*, que facilita la tarea de hacer pruebas y comprobar el correcto funcionamiento del programa.

En inglés, al depurador se le llama *debugger*. En español, lo podríamos traducir como *eliminador de bichos* o algo parecido. Proviene del término *bug* (*bicho*), que es como se denomina a los errores en la lógica del programa. En general, se puede considerar que hay dos tipos de errores en los programas:

- **Errores sintácticos:** son errores en la sintaxis del código. Los hay de muchos tipos: podría tratarse del nombre de un identificador escrito de manera incorrecta, o del intento de utilizar una variable que no se ha declarado correctamente, o de la llamada a funciones de una librería que hemos olvidado poner en los *include* u otras causas. En muchos de estos errores, el propio editor de VSCode nos avisará subrayando el código en rojo. Este tipo de errores provocan que el programa no compile. Al intentar compilar, recibiremos mensajes del motivo por el que no compila el programa. Suelen ser los errores más fáciles de corregir.
- **Errores en la lógica del programa:** son los más difíciles de corregir. En este tipo de errores, el programa se puede compilar y ejecutar, pero los resultados que proporciona no son los que esperábamos. A este tipo de errores se les suele denominar *bugs* y de ahí el nombre *debugger* con el que se suele llamar a las herramientas de depuración.

El depurador integrado en VSCode proporciona herramientas que permiten ejecutar los programas paso a paso, examinando en cada momento el valor que toman las variables y observando por donde se mueve el flujo de instrucciones (si se pasa dentro de un *if*, o de un *while*, o se accede al *else*, etc.). De esta manera, podremos localizar más fácilmente los errores de funcionamiento: el programa no fluye por donde debería, las variables no toman el valor correcto, en qué línea se ha producido la desviación, etc.

Para poder utilizar el depurador es necesario compilar los programas de una forma especial. Si has instalado el compilador de C y el editor VSCode siguiendo las instrucciones del manual que se proporciona con la asignatura, no deberías tener problemas para ejecutar el depurador. En particular, es necesario haber instalado en VSCode la extensión *Intellisense, debugging and Code Browsing*, tal como se explica en dicho manual.

2.- Preparación del entorno para usar el depurador

En los próximos apartados vamos a utilizar dos programas para probar las distintas opciones que ofrece el depurador. Debes crear una carpeta para el proyecto llamada, por ejemplo, *PruebasDepurador_1*, colgando de la carpeta raíz que ya se utilizó dentro del manual de instalación de VSCode (Prog_I).

El primer programa será un ejemplo sencillo que conste de una única función, la función `main()`. Es un ejemplo adecuado para las primeras clases del curso. En concreto, en la Práctica 1 se piden resultados que se obtienen de manera similar a lo que se hace en este ejemplo. El segundo programa contendrá, además de la función `main()`, una función auxiliar. Es un ejemplo que necesita comprender cómo se hacen funciones en C y, por ello, se aconseja retomarlo cuando se explique la teoría correspondiente en el curso.

Para poder depurar el código de un programa es necesario que el proyecto disponga de los ficheros *launch.json* y *tasks.json* dentro de una carpeta *.vscode* que debería colgar de la carpeta raíz, encontrándose pues a la misma altura que las carpetas de los distintos proyectos. Como se explicó en el manual de instalación de VSCode, podemos crear dichos ficheros haciendo *click* con el botón derecho del ratón sobre el código de alguno de los archivos *.c* del proyecto y seleccionando la opción *Agregar configuración de depuración*. Entonces, se creará la carpeta *.vscode*, si no existía, y dentro de ella, los ficheros *tasks.json* y *launch.json*.

3.- Primer ejemplo: Programa principal sin funciones auxiliares

El primer ejemplo nos va a servir para mostrar cómo arrancar el depurador y los elementos de su interface.

Crea un fichero llamado *test_debug_1.c* dentro de la carpeta del proyecto y copia en él el siguiente código:

```
// test_debug_1.c
// Pruebas con el depurador de VSCode
#include <stdio.h>

int main() {
    int x = 0;

    for(int i=0; i<3; i++) {
        x = i*i;
        printf("%d - %d \n", i, x);
    }

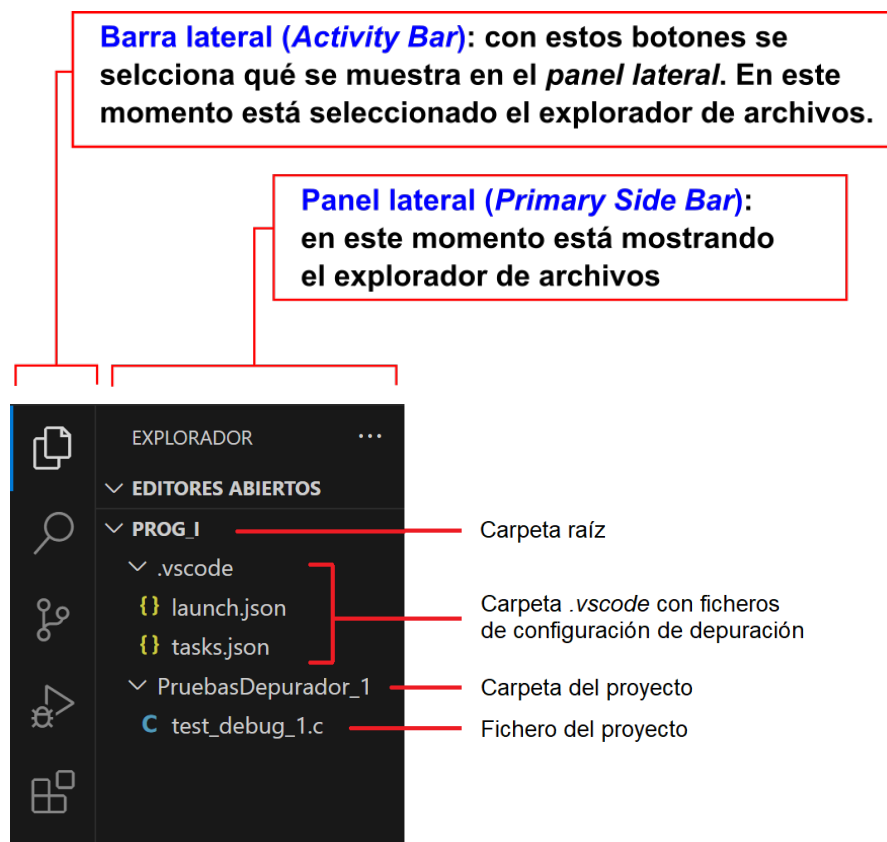
    return 0;
}
```

Cuando termines de teclear el código, grábalo en el disco con CTRL+S.

Se trata de un programa que tiene una única función, la función `main()`, que, como sabes, es la función por la que arrancan los programas escritos en C. Cuando compilamos un programa y lo ejecutamos, la ejecución comienza siempre en la función `main()` del programa, que se suele denominar también el *programa principal*.

Ahora tendrás que crear la carpeta *.vscode* con los ficheros *tasks.json* y *launch.json*, que son los que permitirán a VSCode utilizar el depurador. Sáltate este paso si ya dispones de dichos ficheros de compilaciones anteriores.

Si no, pincha con el botón derecho del ratón en el código del programa y selecciona la opción *Agregar configuración de depuración*. VSCode te pedirá que selecciones el depurador; debes elegir C/C++: *gcc.exe Compilar y depurar el archivo activo*. En la parte izquierda de VSCode, deberías ver algo parecido a lo siguiente:



Por su parte, en el editor de texto de VSCode, deberías ver el código de *test_debug_1.c*, que es el fichero que está seleccionado en el *panel lateral*, como muestra la siguiente figura:

```
C test_debug_1.c X
C test_debug_1.c > ...
1 // test_debug_1.c
2 // Pruebas con el depurador de VSCode
3 #include <stdio.h>
4
5 int main() {
6     int x = 0;
7
8     for(int i=0; i<3; i++) {
9         x = i*i;
10        printf("%d - %d \n", i, x);
11    }
12
13    return 0;
14 }
15
```

At the bottom, the status bar shows: Lín. 3, col. 1 Espacios: 4 UTF-8 CRLF {} C Win32

Como puedes ver, el programa declara una variable llamada x a la que asigna un valor inicial 0. A continuación, hace un bucle cuya variable contador llamada i se mueve entre 0 y 3. En cada iteración del bucle, se asigna a la variable x el cuadrado de la variable contador y muestra en pantalla una línea con: el valor de la i , un guión, el valor que tiene x en ese momento, y un cambio de línea. Cuando termina la instrucción iterativa, el programa termina devolviendo el valor 0, para indicar que todo ha ido bien.

NOTA: Buenas prácticas de programación

¡Acostúmbrate a teclear el código (a picar tecla) usa lo menos posible el copy-paste!

Cada vez que modifiques algo en el código fuente, hay que guardar el fichero en el disco con las modificaciones. Podrías usar la opción de menú *Archivo -> Guardar*, pero es mucho mejor que te acostumbres a utilizar el atajo de teclado CTRL+S (¡S de Save!).

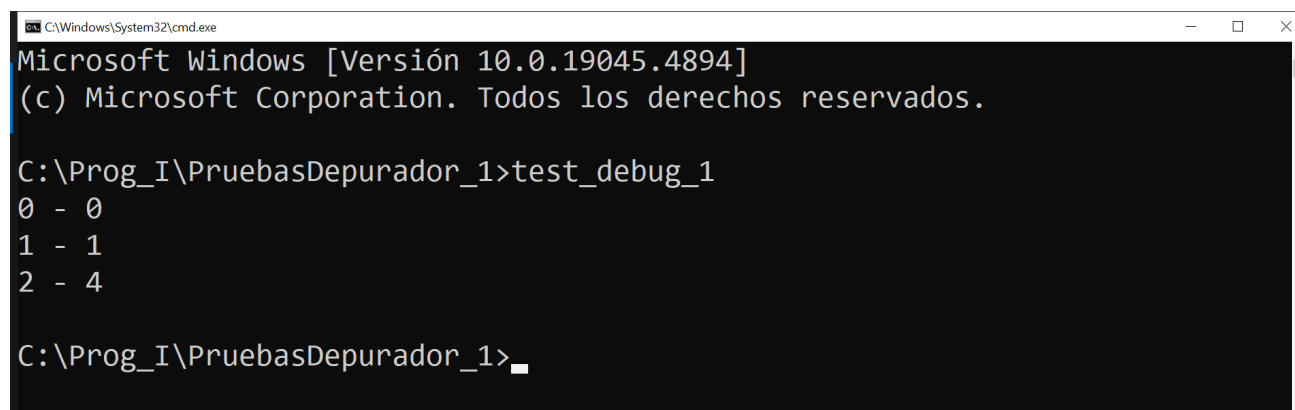
3.1.- Ejecución sin depuración

Vamos a compilar el programa `test_debug_1.c` para obtener el ejecutable que, en Windows se llamará `test_debug_1.exe` y en Mac o Linux simplemente `test_debug_1`.

Para compilar, puedes elegir la opción de menú *Terminal -> Ejecutar tarea de compilación* o, mejor aún, utiliza el atajo de teclado CTRL+SHIFT+B (B, de *build*). Con ello, el fichero que esté activo en ese momento en el editor se compilará. Debes seleccionar el compilador `gcc.exe`, si se te solicita. Se debería abrir la zona del terminal en la parte inferior de VSCode y, tras otros mensajes, indicar que la compilación finalizó correctamente. Si te marca algún error de compilación, seguramente será algún error sintáctico en el código que deberás corregir.

Una vez consigas compilar correctamente el código, pinchando en el terminal y pulsando una tecla se cerrará el área de terminales. Ahora, deberías ver en el explorador de archivos el fichero ejecutable.

Ya estamos preparados para ejecutar el programa en una consola externa. Para ello, pincha con el ratón en el código, para que el editor de texto adquiera el foco y pulsa CTRL+SHIFT+C (C, de *console*). Con ello, se debería abrir la consola Command Prompt de Windows. Asegúrate que te encuentras en el directorio del programa (si no cambia a él utilizando "cd") y teclea `test_debug_1`: se ejecutará el programa y deberías ver algo parecido a la siguiente figura:



```
C:\Windows\System32\cmd.exe
Microsoft Windows [Versión 10.0.19045.4894]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Prog_I\PruebasDepurador_1>test_debug_1
0 - 0
1 - 1
2 - 4


C:\Prog_I\PruebasDepurador_1>
```

NOTA: Terminal por defecto en Windows

En el Manual de instalación de VSCode que tienes disponible en el Moodle de la asignatura, se explica cómo configurar VSCode para que, en Windows, cuando abres un terminal, el terminal que se abra sea la consola (Command Prompt) y no otro.

3.2.- Ejecución con depuración

Para depurar el programa, hay que establecer al menos un *punto de interrupción* (*break point*). Se trata de puntos en los que el programa se detiene y podemos inspeccionar los valores que tienen las variables en ese momento. Un buen sitio para establecer el primer *break point* es al principio de la función `main()`, como se ha hecho en la siguiente figura:

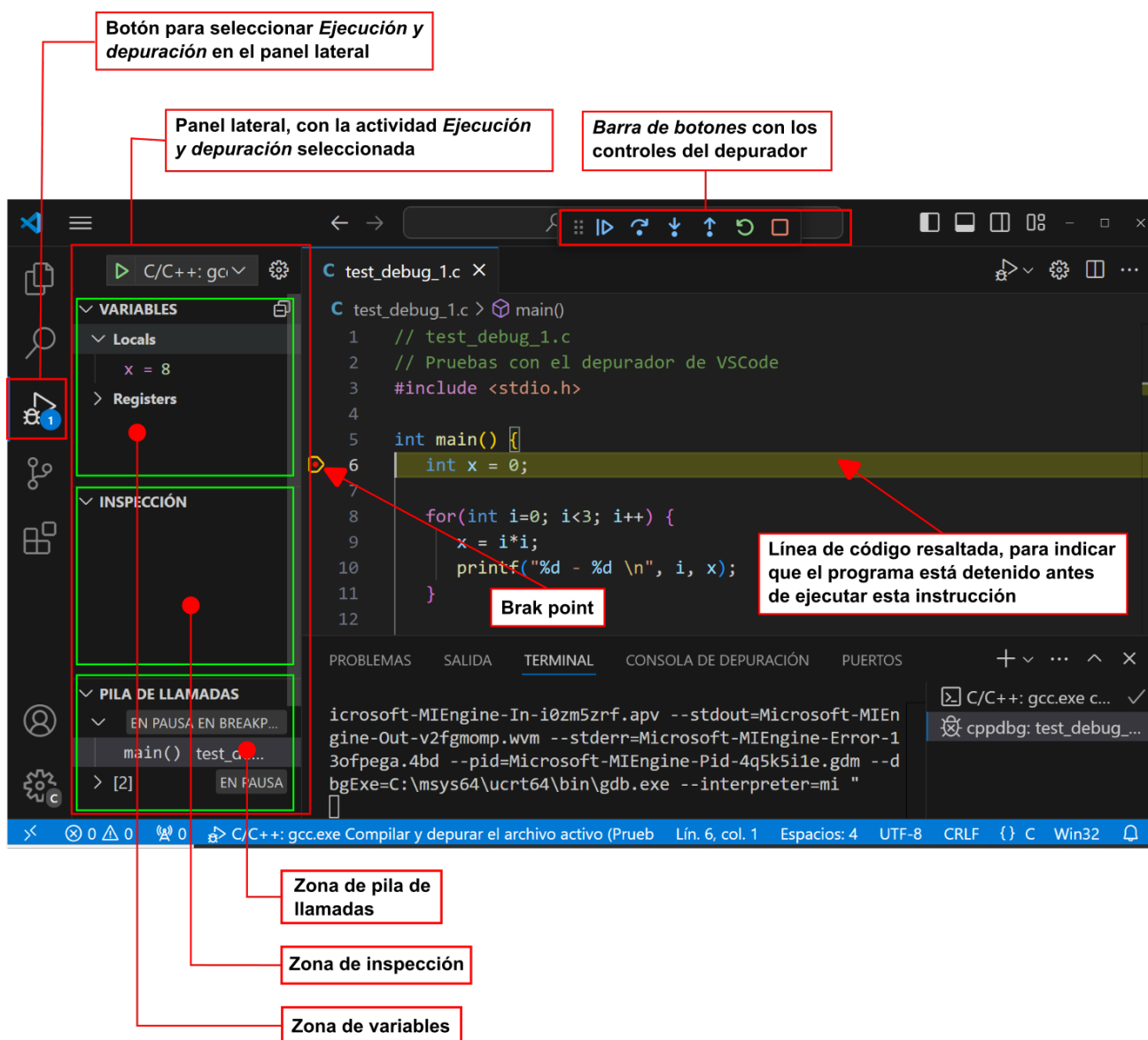


```
1 // test_debug_1.c
2 // Pruebas con el depurador de VSCode
3 #include <stdio.h>
4
5 int main() {
6     int x = 0;
7
8     for(int i=0; i<3; i++) {
9         x = i*i;
10        printf("%d - %d \n", i, x);
11    }
12
13    return 0;
14 }
15
```

Si sobrevuelas con el ratón la zona del editor de texto a la izquierda de los números de línea, verás que van apareciendo puntos rojos. Pincha a la izquierda del número de línea 6, que es la instrucción en la que se crea y se inicializa la variable `x`.

Para lanzar el depurador tienes que seleccionar la opción de menú *Ejecutar -> Iniciar Depuración* o utilizar el atajo de teclado `F5`. También puedes lanzar la depuración con el botón que hay arriba y a la derecha del editor de texto (replicado en el panel lateral izquierdo), con un icono de *flecha + bicho*. Si lo haces, VSCode cambiará de aspecto y la ejecución queda detenida antes de ejecutar la instrucción en la que esté el primer *break point*.

La siguiente figura muestra el resultado tras lanzar el depurador. Observa los nuevos paneles y controles que ofrece VSCode mientras se está depurando el programa:



Hay bastantes cosas para aprender aquí, pero vamos por partes. Observa que en los botones de la parte izquierda está seleccionado *Ejecución y depuración*. Como resultado, el panel izquierdo ya no muestra el explorador de archivos, sino el entorno de depuración. El panel está dividido en varias zonas:

1. **Variables:** muestra el valor que tienen las variables existentes en ese momento. Ofrece dos zonas: la zona de variables locales y la zona de registros. Se denominan *variables locales* las existentes dentro de una función concreta. En cuanto a la zona de *Registros*, muestra el valor de los *registros* del procesador u otros. Se denominan *registros* a determinadas zonas de memoria destinadas a guardar valores. Por el momento no debes preocuparte de ellos, pues su uso está destinado a programadores avanzados.
2. **Inspección:** en esta zona podemos teclear cualquier expresión y nos mostrará el valor que toma en ese momento. Estas expresiones se pueden referir a operaciones realizadas con variables existentes, valores del entorno u otros en los que pudiéramos estar interesados.
3. **Pilas de llamadas:** por el momento, tampoco debes preocuparte por utilizar esta zona.
4. **Puntos de interrupción:** aparecerán los distintos puntos de interrupción que hayamos definidos.

En la parte superior del panel, pulsando sobre los puntos que aparecen, podemos seleccionar qué zonas queremos ver. Puede ser cómodo desactivar las zonas: *Pilas de llamadas* y *Puntos de interrupción*.

SABER MÁS: Creación y asignación de variables

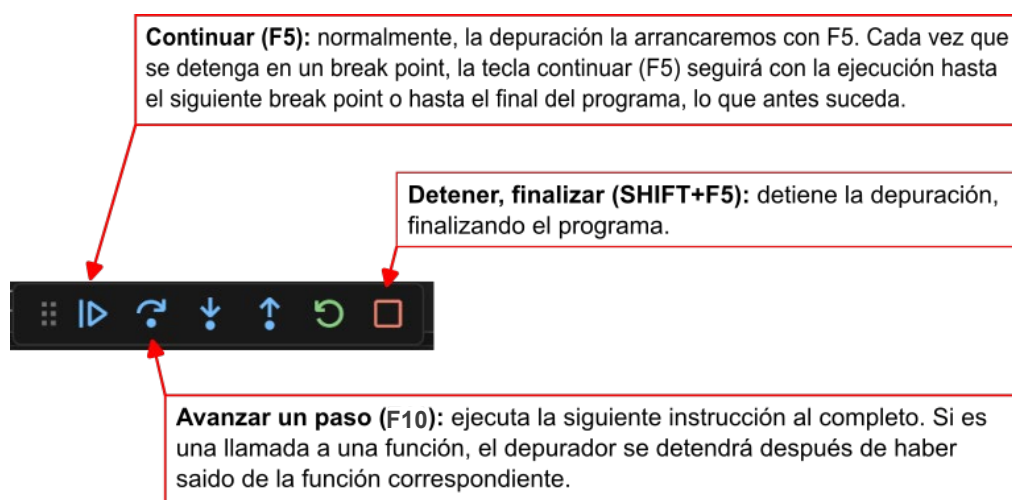
Podría surgirme una duda: ¿si el programa aún no ha ejecutado la instrucción en la que se crea x , ¿por qué aparece ya en la zona de variables como si existiera y, además, con un valor aparentemente incorrecto?

El código creado por el compilador, lleva las declaraciones de variables al principio de la función, independientemente de en qué parte de la función las declaremos. El compilador hace eso para reservar memoria para todas las variables que se usen en la función. En cambio, el valor se asigna en el momento en el que suceda dicha asignación de valor. El valor de x que estás viendo es un valor arbitrario.

El caso de la variable contador i es un poco diferente. Esa variable no existe durante toda la función, sino que sólo existe durante la vida útil del bucle (al estar definida en su cabecera). Si haces avanzar una instrucción al depurador, pulsando F10, podrás observar que, cuando la ejecución está antes de la primera iteración, ya se crea la variable i y, cuando termina la ejecución del bucle, deja de existir.

Si has lanzado el programa, verás que se ha detenido en el *break point* de la línea 6, en concreto, antes de ejecutar la línea 6, que es donde el programa declara y asigna valor a la variable x .

Ahora puedes ir ejecutando el programa paso a paso. La siguiente figura te muestra los botones del depurador que puedes utilizar:



En cada paso, observa los valores que el depurador te muestra para las variables.

La zona de inspección la puedes utilizar para obtener el valor de cualquier expresión. Por ejemplo, podrías teclear una expresión `printf` diferente, o un cálculo basado en los valores de las variables, o cualquier otra cosa que te pueda interesar conocer en un momento concreto de la ejecución de un programa.

4.- Ejemplo de depuración cuando hay varias funciones

Dentro de la carpeta raíz, crea el proyecto (nueva subcarpeta) *PruebasDepurador_2*, y en su interior genera el fichero *test_debug_2.c* con el siguiente código:

```
#include <stdio.h>

int cuadrado(int n);

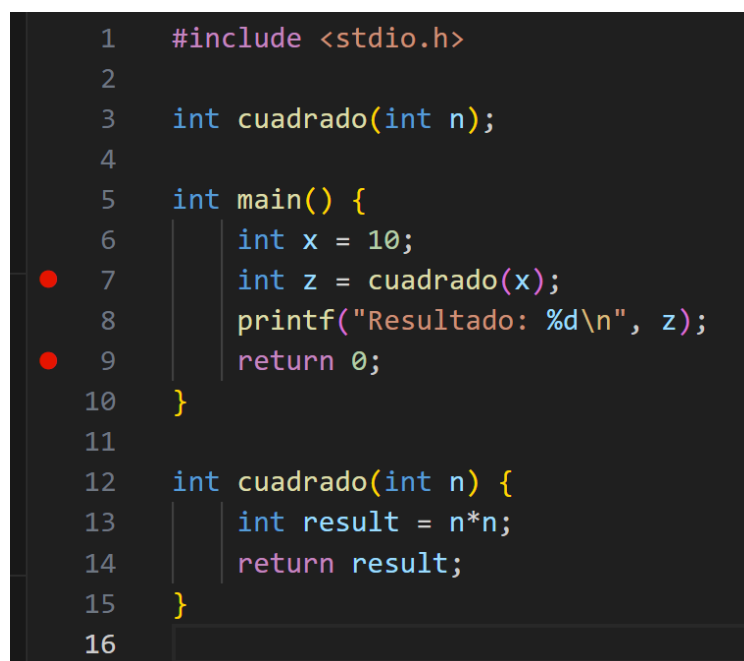
int main() {
    int x = 10;
    int z = cuadrado(x);
    printf("Resultado: %d\n", z);
    return 0;
}

int cuadrado(int n) {
    int result = n*n;
    return result;
}
```

El programa anterior consta de dos funciones:

1. `cuadrado()`: es una función que recibe como argumento un número entero y devuelve otro número entero con el cuadrado del valor recibido.
2. `main()`: es el programa principal de la aplicación. Primero, crea una variable `x` y le asigna el valor 10. A continuación, se crea una variable `z` a la que se le asigna el resultado de llamar a la función `cuadrado()` con el valor de `x` como argumento. Finalmente, se imprime el valor de `z`.

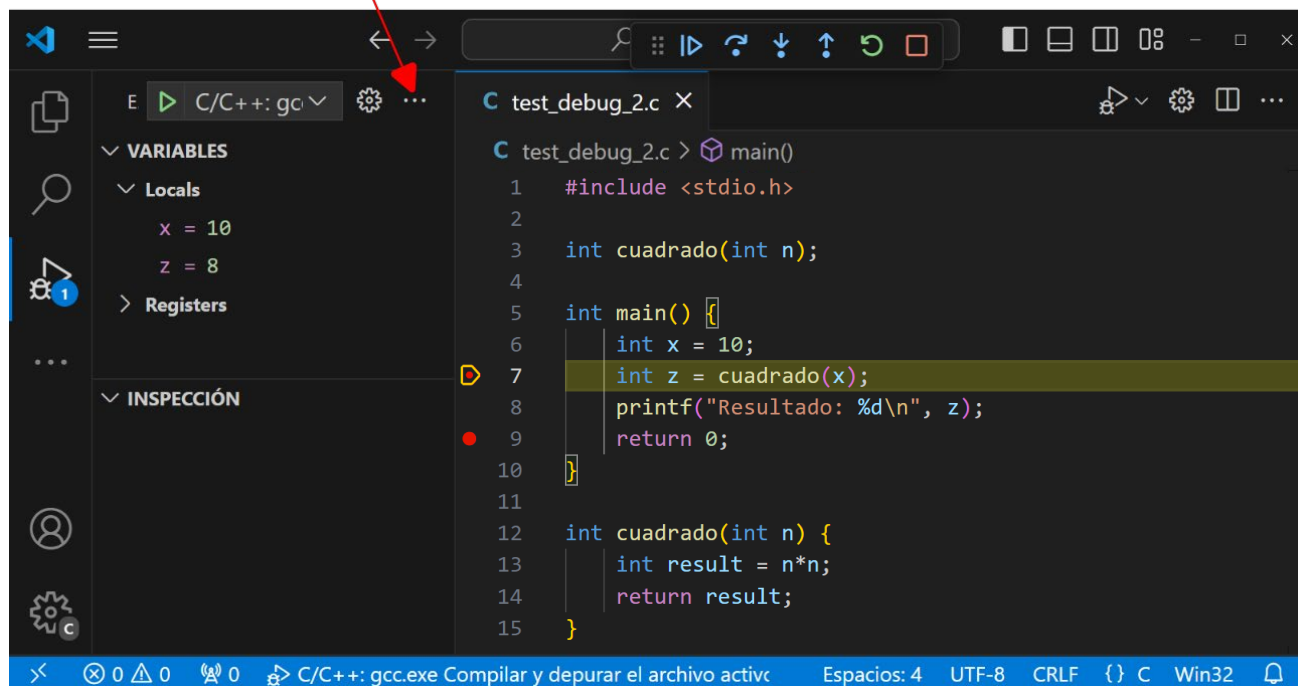
Vamos a poner dos puntos de interrupción: uno en la línea 7 de código, para que el programa se detenga antes de la instrucción en la que se realiza la llamada a la función `cuadrado()` y otro en la línea 9, antes de terminar el programa. La siguiente figura muestra el resultado:



```
1  #include <stdio.h>
2
3  int cuadrado(int n);
4
5  int main() {
6      int x = 10;
7      int z = cuadrado(x);
8      printf("Resultado: %d\n", z);
9      return 0;
10 }
11
12 int cuadrado(int n) {
13     int result = n*n;
14     return result;
15 }
16
```

Si lanzas el depurador, deberías ver algo parecido a lo que muestra la figura inferior. Observa que hemos ocultado las secciones *Pilas de llamadas* y *Puntos de interrupción* en el panel lateral, actuando sobre las opciones que se ofrecen al pinchar los puntos de la parte superior. También hemos ocultado el área de terminales de la parte inferior de la ventana de VSCode.

Botón con opciones para mostrar u ocultar secciones en el panel lateral





Observa que las variables *x* y *z* ya aparecen en la zona de variables, aunque la *z* tiene un valor arbitrario, pues aún no se ha ejecutado la instrucción en la que se realiza la asignación.

La ejecución del programa está detenida antes de la instrucción:

```
int z = cuadrado(x);
```

Si se continúa la ejecución (F5), el programa entraría en la función `cuadrado()` con el valor de *x*. En la función, el valor se elevaría al cuadrado y se devolvería el resultado. El programa principal asigna este valor devuelto a la variable *z*.

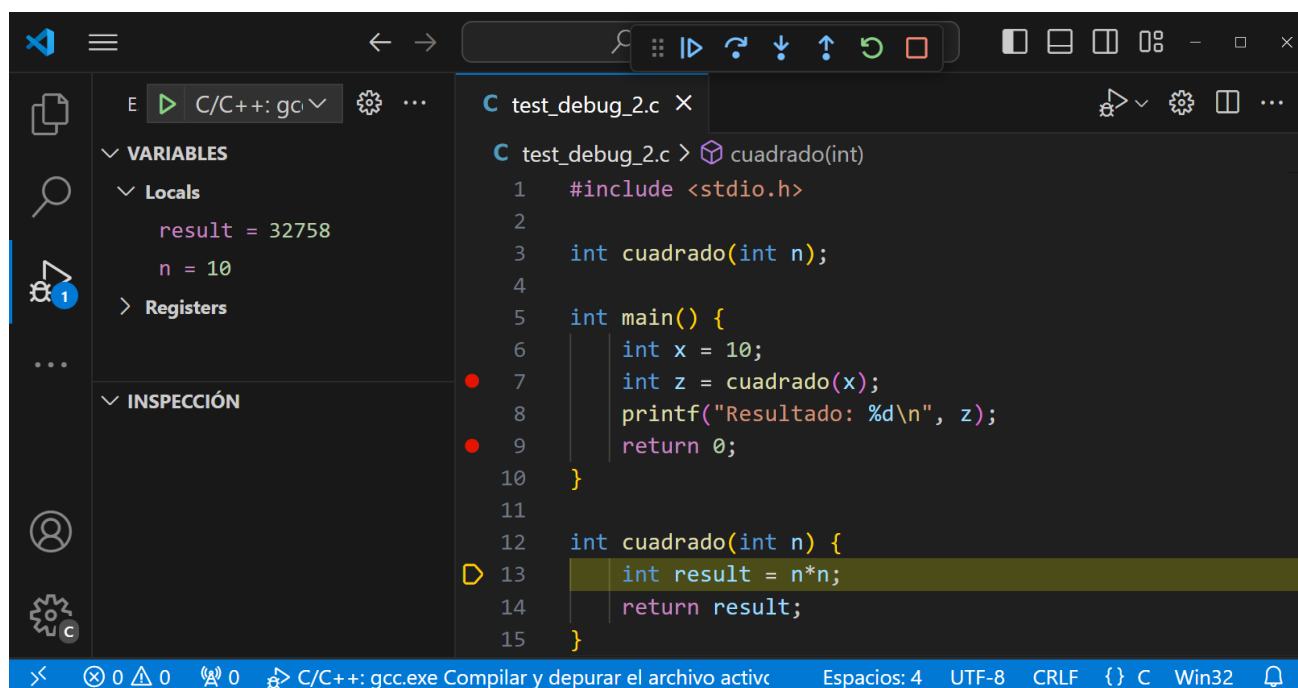
También podemos ordenar al depurador que avance paso a paso de dos maneras:

- Botón  (F10): Pasando de la línea 7 a la línea 8 de la función `main()`. Esto es, el depurador no se detiene en las instrucciones de dentro de la función `cuadrado()`, sino que ejecuta toda la función de una tacada, realiza la asignación del valor devuelto a *z* y se detiene antes de ejecutar la línea 8 del `main()`.
- Botón  (F11): Pasando a la línea 13 del código, que es la primera línea de la función `cuadrado()`, siguiendo el flujo exacto del programa.

Dependiendo de la situación unas veces nos interesará más una forma de avanzar u otra.

Si elegimos entrar en la función, podemos recorrer sus instrucciones paso a paso y, cuando ejecutemos la instrucción `return`, el siguiente paso será completar la línea 7 de `main()`, realizando la asignación del valor devuelto a la variable `z`.


En la siguiente figura se muestra el depurador detenido en la primera instrucción dentro de la función `cuadrado()`. Observa ahora la zona de variables en el panel lateral: las variables que se muestran son las variables locales de la función `cuadrado()`. En concreto, como aún no se ha realizado la asignación de valor a la variable `result`, observa que esta tiene un valor arbitrario.



Cuando estamos dentro de la función `cuadrado()`, las variables accesibles son sólo las de dicha función (`n` y `result`), en cambio, las variables del `main()`, por ejemplo la `z`, no son visibles desde dentro de la función `cuadrado()`. Este es un concepto importantísimo y que tienes que asimilar bien:

Las variables solo son visibles dentro del ámbito en el que han sido declaradas.

Es recomendable que practiques con el depurador, pues te permitirá comprender bien el orden de ejecución de las instrucciones de los programas y los diferentes ámbitos en los que son visibles unas variables u otras. Por ejemplo, podrías crear una variable llamada `z` dentro de la zona *Inspección*. Podrás comprobar que, cuando estés en instrucciones de `main()`, la variable `z` tiene su valor. En cambio, cuando estés dentro de `cuadrado()`, el depurador te indicará que la variable no es accesible.

Cuando estamos con el depurador detenidos dentro de una función, podemos pulsar . Esto provocará que se ejecute el código restante de la función y se salga de ella, devolviendo el control a la instrucción de la función superior en la que se había hecho la llamada.