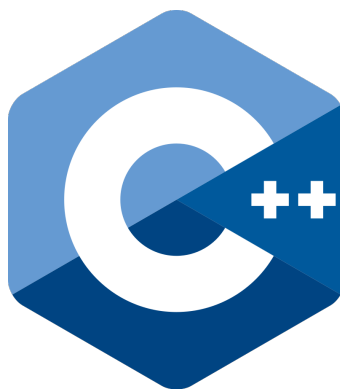




# Metodología de la Programación

DGIM

Curso 2020/2021



## **Guion de prácticas** *Sesiones de depuración*

*Marzo 2021*



# Índice

<b>1. Descripción</b>	<b>5</b>
1.1. Control de la ejecución . . . . .	6
1.2. Visualización de datos . . . . .	7
<b>2. Depurar paso a paso</b>	<b>7</b>
2.1. Un caso de ejemplo . . . . .	8



## 1. Descripción

En esta sesión se van a practicar los conceptos sobre depuración que se han introducido en el guión de Netbeans, por lo que se recomienda haber leído antes éste.

Hay disponible una serie de videotutoriales sobre este tema en la carpeta pública de Google Drive ([Abrir en navegador →](#))

La Figura 1 se va a utilizar como referencia para introducir estos conceptos y muestra un programa en fase de depuración.

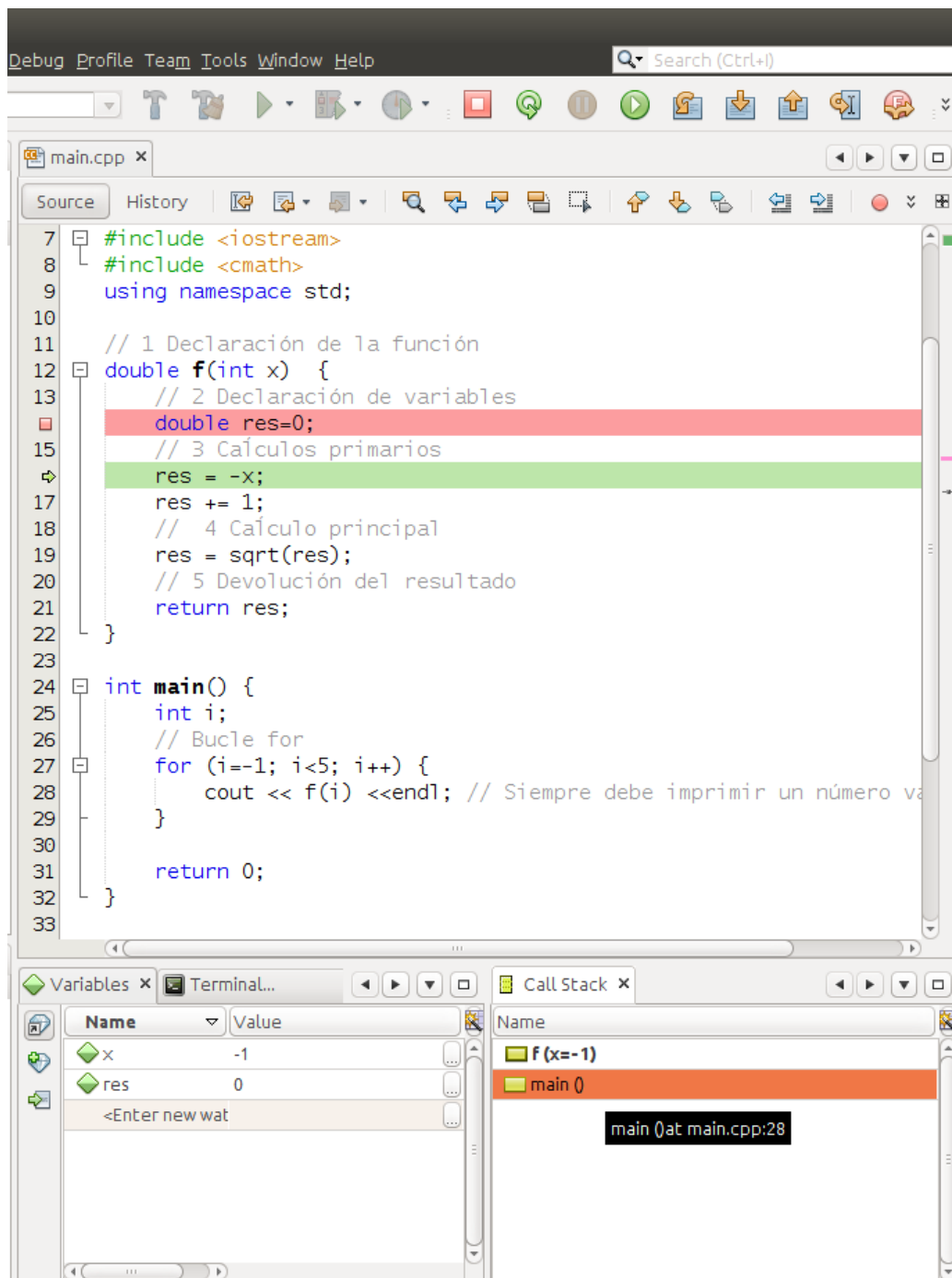


Figura 1: Un programa cuya ejecución se está depurando

## 1.1. Control de la ejecución

- **Punto de ruptura - breakpoint**

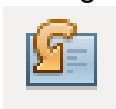
Es una línea de código en la que queremos que el programa interrumpa su ejecución. Se introduce haciendo clic en el número de línea del editor de código, o bien botón derecho - Toggle breakpoint. Una vez que se establece un punto de ruptura, éste aparece marcado como una línea sombreada de rojo en el editor.

- **Inicio de la depuración** (debe haber al menos un punto de ruptura definido)

La línea que está a punto de ejecutarse aparece sombreada en verde. Debug - Debug Project



- **Ejecución paso a paso** Ejecuta el siguiente paso. Si es una llamada a una función o método, entonces la ejecuta de golpe, sin entrar a ella  
Debug - Step over



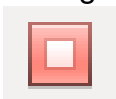
- **Ejecución paso a paso** Ejecuta el siguiente paso. Si es una llamada a una función o método, entonces entra dentro de ella y también la ejecuta paso a paso  
Debug - Step into



- **Continuar hasta el cursor** Ejecuta el programa hasta que llegue a la línea en la que se ha marcado el cursor.  
Debug - Run to cursor



- **Parar** Interrumpe el proceso de depuración  
Debug - Terminate



- **Continuar** Ejecuta el programa hasta el final o hasta que pase por otro punto de ruptura  
Debug - Continue



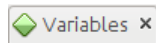
- Si se quiere lanzar el depurador con redirección de datos de entrada o parámetros de entrada, éstos se deberán especificar en las opciones del proyecto. Ver guión NetBeans, **Project - Properties- C++ Compiler - Run - Run arguments**

## 1.2. Visualización de datos

La principal herramienta para ver y monitorizar el estado de las variables del programa, exclusivamente aquellas cuyo ámbito y visibilidad coincida con la línea de código que se está ejecutando.

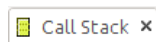
- **Ventana de variables**

Visualiza el contenido de las variables visibles desde el punto de ejecución en el que se encuentra el programa y permite incluso su modificación. Window - Debugging - Variables



- **Ventana de la pila**

Visualiza la pila de llamadas y los puntos del código desde los que se han ido produciendo éstas Window - Debugging - Call Stack



- **Visualización directa de una variable**

Cualquier variable que aparezca en el ámbito de ejecución actual se puede visualizar sin más que colocar el cursor sobre el identificador de la variable.

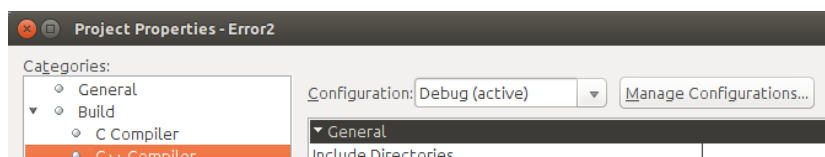
```
res += 1;
// 4 Cálculo principal res=0
res = sqrt(res);
```

Pero esta visualización desaparece nada más mover el cursor, de forma que si se quiere dejar fija esta visualización, se puede anclar en la página para que se pueda monitorizar haciendo clic en el pin que aparece en la ventana de visualización de la variable.

```
res += 1;
// 4 Cálculo principal res = 0
res = sqrt(res);
// 5 Evaluación del resultado
```

## 2. Depurar paso a paso

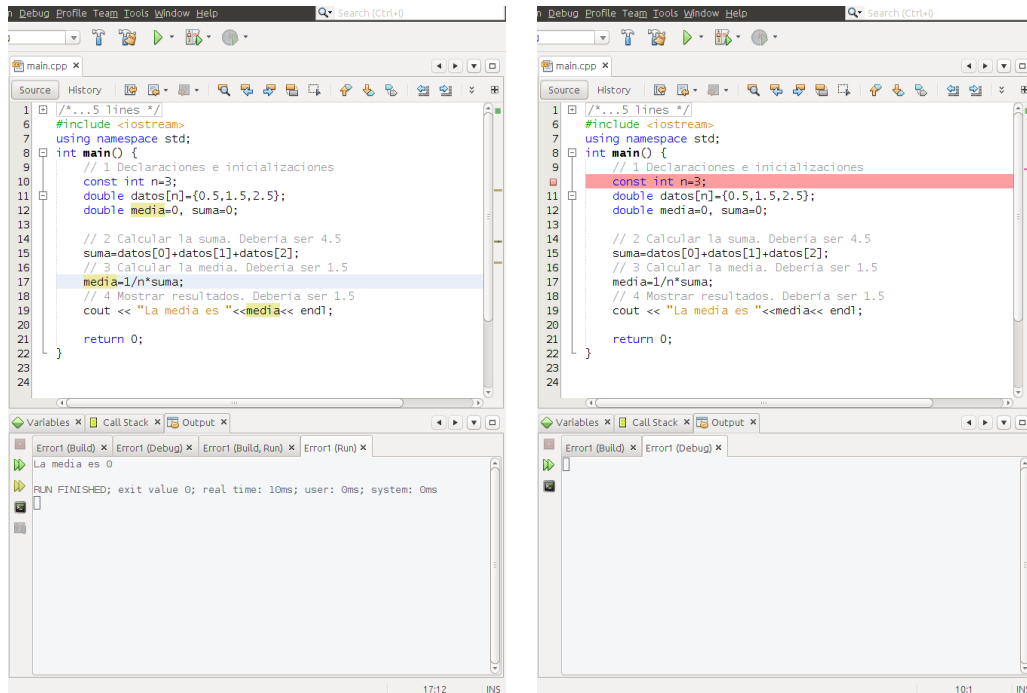
1. En primer lugar, se debe asegurar que el programa compila y enlaza y que se ha compilado en la configuración de **DEBUG**, que es la alternativa por omisión.



2. Estudiar el comportamiento del programa durante un error e intentar localizar el código que pueda estar fallando.
3. Colocar un punto de ruptura

## 2.1. Un caso de ejemplo

El siguiente programa calcula la media de tres valores almacenados en el vector `datos`.



```
1 //...5 lines ...
2
3 #include <iostream>
4 using namespace std;
5
6 int main() {
7     // 1 Declaraciones e inicializaciones
8     const int n=3;
9     double datos[n]={0.5,1.5,2.5};
10    double media=0, suma=0;
11
12    // 2 Calcular la suma. Debería ser 4.5
13    suma=datos[0]+datos[1]+datos[2];
14    // 3 Calcular la media. Debería ser 1.5
15    media=suma/n;
16    // 4 Mostrar resultados. Debería ser 1.5
17    cout << "La media es " << media << endl;
18
19    return 0;
20 }
```

Variables | Call Stack | Output | Error (Build) | Error (Debug) | Error (Run) | Error (Build, Run) | Error (Run)

La media es 0

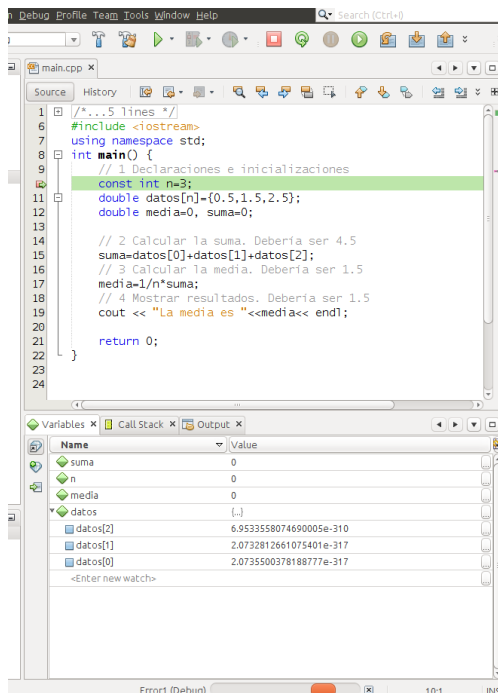
RUN FINISHED; exit value 0; real time: 10ms; user: 0ms; system: 0ms

17:12 INS

La media debería ser 1.5 pero se visualiza como 0 en la ejecución. Lo primero es intentar localizar con la mayor precisión, y de forma iterativa si fuese necesario, la sentencia que produce el error. Tenemos cuatro alternativas: 1) el bloque de declaraciones e inicializaciones, 2) El cálculo de la suma, 4) El cálculo de la media o 4) La visualización del resultado. Al menos una de ellas falla

El primer paso es poner un punto de ruptura (PR). Esto se hace haciendo clic sobre el número de la línea en la que queremos poner el PR. Aparecerá un recuadro rojo sobre el número de línea y la línea se sombrea de color rojo también. El PR hará que el programa se detenga al llegar a ese punto, justo antes de ejecutar esa línea.

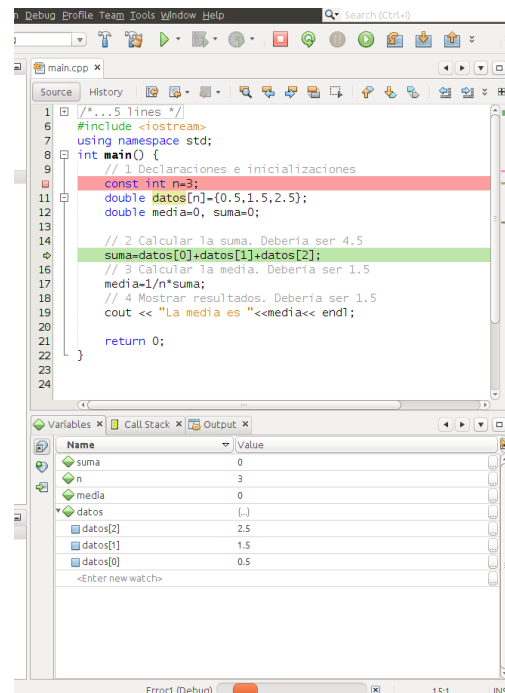




```
1  /*...5 lines */
2  #include <iostream>
3  using namespace std;
4  int main() {
5      // 1 Declaraciones e inicializaciones
6      const int n=3;
7      double datos[n]={0.5,1.5,2.5};
8      double media=0, suma=0;
9
10     // 2 Calcular la suma. Debería ser 4.5
11     suma=datos[0]+datos[1]+datos[2];
12     // 3 Calcular la media. Debería ser 1.5
13     media=1/n*suma;
14     // 4 Mostrar resultados. Debería ser 1.5
15     cout << "La media es " << media << endl;
16
17     return 0;
18 }
```

Name	Value
suma	0
n	0
media	0
datos	[...]
datos[2]	6.9533558074690005e-310
datos[1]	2.0732812661075401e-317
datos[0]	2.0735500378188777e-317

Se inicia la depuración. NetBeans marca la línea que está a punto de ejecutarse en color verde. En este caso, al iniciar la depuración, se empieza a ejecutar el programa hasta que llega al primer PR. Se debe poner atención de poner el PR en un punto adecuado, porque podría ser que la ejecución del programa no llegue a pasar por él, por lo que la ejecución se haría completa, sin interrumpirse. Se aprovecha para desplegar la pestaña de variables y observar los valores de las variables antes de inicializarse, como los elementos del vector, que contienen basura.



```
1  /*...5 lines */
2  #include <iostream>
3  using namespace std;
4  int main() {
5      // 1 Declaraciones e inicializaciones
6      const int n=3;
7      double datos[n]={0.5,1.5,2.5};
8      double media=0, suma=0;
9
10     // 2 Calcular la suma. Debería ser 4.5
11     suma=datos[0]+datos[1]+datos[2];
12     // 3 Calcular la media. Debería ser 1.5
13     media=1/n*suma;
14     // 4 Mostrar resultados. Debería ser 1.5
15     cout << "La media es " << media << endl;
16
17     return 0;
18 }
```

Name	Value
suma	0
n	3
media	0
datos	[...]
datos[2]	2.5
datos[1]	1.5
datos[0]	0.5

Se ejecuta (F8 - Step over) el bloque de declaraciones e inicializaciones y se observa en la pestaña de variables que estas se han producido correctamente.



```
1 /*...5 lines */
2 #include <iostream>
3 using namespace std;
4 int main() {
5     // 1 Declaraciones e inicializaciones
6     const int n=3;
7     double datos[n]={0.5,1.5,2.5};
8     double media=0, suma=0;
9
10    // 2 Calcular la suma. Debería ser 4.5
11    suma=datos[0]+datos[1]+datos[2];
12    // 3 Calcular la media. Debería ser 1.5
13    media=1/n*suma;
14    // 4 Mostrar resultados. Debería ser 1.5
15    cout << "La media es " << media << endl;
16
17    return 0;
18 }
```

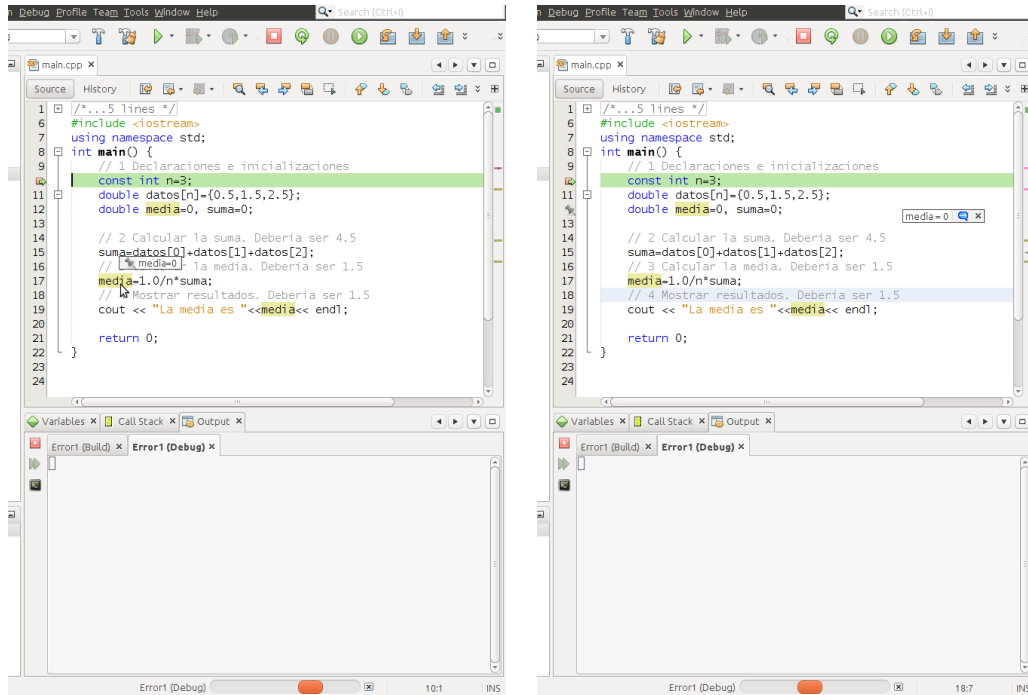
Name	Value
suma	4.5
n	3
media	0
datos	[...]
datos[2]	2.5
datos[1]	1.5
datos[0]	0.5

Se ejecuta (F8 - Step over) la línea que calcula la suma y se comprueba que el valor calculado es el correcto

```
1 /*...5 lines */
2 #include <iostream>
3 using namespace std;
4 int main() {
5     // 1 Declaraciones e inicializaciones
6     const int n=3;
7     double datos[n]={0.5,1.5,2.5};
8     double media=0, suma=0;
9
10    // 2 Calcular la suma. Debería ser 4.5
11    suma=datos[0]+datos[1]+datos[2];
12    // 3 Calcular la media. Debería ser 1.5
13    media=1/n*suma;
14    // 4 Mostrar resultados. Debería ser 1.5
15    cout << "La media es " << media << endl;
16
17    return 0;
18 }
```

Name	Value
suma	4.5
n	3
media	0
datos	[...]
datos[2]	2.5
datos[1]	1.5
datos[0]	0.5

Se ejecuta (F8 - Step over) la línea de la media y se observa que el valor calculado (0.0) difiere del valor esperado (1,5) por lo que aquí aparece el primer error. Puede haber otros errores, pero pueden ser dependientes de este, por lo que es buena idea interrumpir el programa ahora e intentar reparar este error. Si se observa la línea que falla es una suma, la cual ya está comprobada que funciona, y una división. Por lo tanto el error debe estar en la división. Se observa que, efectivamente, es una división entre números enteros, razón por la cual el resultado es 0.0.



Se repara este error y se vuelve a iniciar la depuración (para lo cual NetBeans recompila de nuevo el proceso tras los cambios introducidos). Esta vez, para monitorizar los valores se va a usar otra funcionalidad. Si se coloca el cursor sobre una variable del programa, NetBeans mostrará una pequeña ventana mostrando su valor.

Esta ventana es temporal, si se mueve el cursor, la ventana desaparece. No obstante, se puede hacer fija pulsando sobre el pin de la ventana, lo que ancla esta ventana de forma definitiva, pudiéndose colocar en una zona bien visible para monitorizar su valor conforme se ejecuta el programa.



```
1 /*...5 lines */
2 #include <iostream>
3 using namespace std;
4 int main() {
5     // 1 Declaraciones e inicializaciones
6     const int n=3;
7     double datos[n]={0.5,1.5,2.5};
8     double media=0, suma=0;
9     // 2 Calcular la suma. Debería ser 4.5
10    suma=datos[0]+datos[1]+datos[2];
11    // 3 Calcular la media. Debería ser 1.5
12    media=1.0/n*suma;
13    // 4 Mostrar resultados. Debería ser 1.5
14    cout << "La media es " << media << endl;
15    return 0;
16 }
```

Justo antes de ejecutar el cálculo de la media

```
1 /*...5 lines */
2 #include <iostream>
3 using namespace std;
4 int main() {
5     // 1 Declaraciones e inicializaciones
6     const int n=3;
7     double datos[n]={0.5,1.5,2.5};
8     double media=0, suma=0;
9     // 2 Calcular la suma. Debería ser 4.5
10    suma=datos[0]+datos[1]+datos[2];
11    // 3 Calcular la media. Debería ser 1.5
12    media=1.0/n*suma;
13    // 4 Mostrar resultados. Debería ser 1.5
14    cout << "La media es " << media << endl;
15    return 0;
16 }
```

Y justo después, se puede observar que, ahora, el valor esperado y el real coinciden, por lo que se puede seguir ejecutando el programa y se comprobaría que la salida de datos también es correcta, lo que da por terminada la sesión de depuración.

### 3. Videotutoriales

1. Primera sesión de depuración ([Abrir en navegador →](#))
2. Segunda sesión de depuración ([Abrir en navegador →](#))