

# Duración, alcance y vinculación de variables. Usos de static.

UTN FRH – INFO II – Ing. Weber



**Duración (storage duration)**

# Duración (storage duration)

La **duración** describe cuánto tiempo una variable permanece en la memoria durante la ejecución del programa.

La duración puede ser:

- **Automática**
- **Estática**
- **Dinámica**

A continuación vamos a analizar y “clasificar” la duración de las **variables locales**.

Tipo	Ejemplo	Duración
local		

# Variables locales: duración (storage duration)

```
10  ▾ int main()  
11  {  
12  |   int x;  
13  |  
14  | }
```

Es creada cuando la función ejecuta la declaración

**Crear es, entre otras cosas, asignarle memoria**

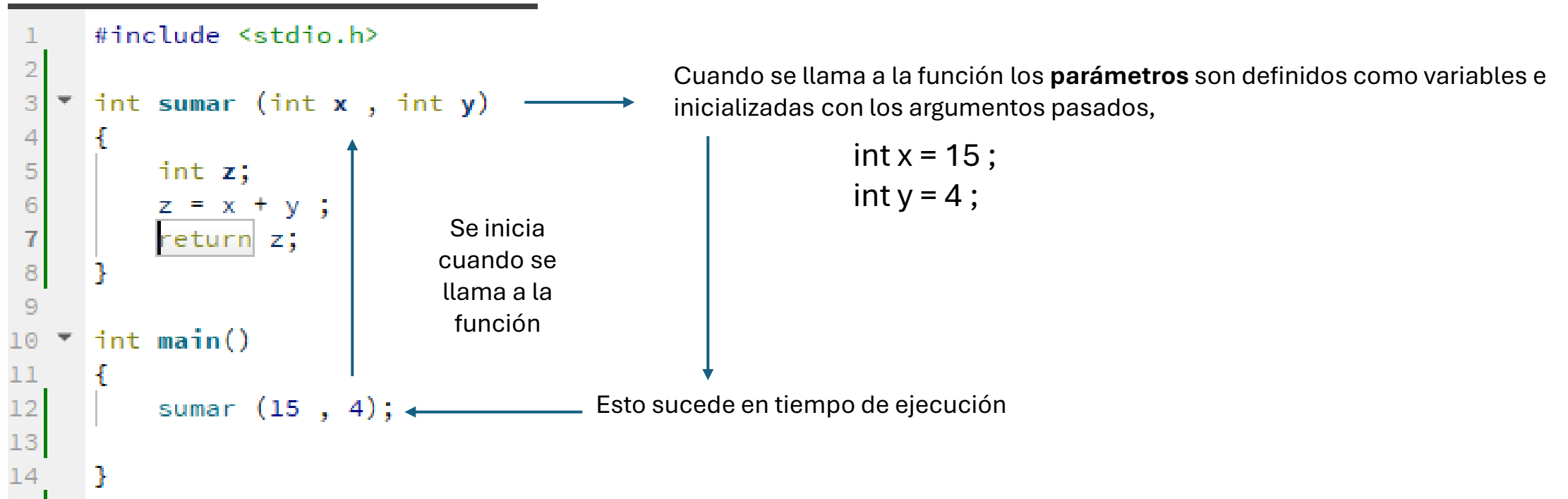
# Variables locales: duración (storage duration)

```
int sumar (int x , int y)
{
    int z;
    z = x + y ;
    return z;
}
```

Los **parámetros** de función “x” e “y” son variables locales.

“z” es también una variable local

# Variables locales: duración (storage duration)



# Variables locales: duración (storage duration)

```
1  #include <stdio.h>
2
3  int sumar (int x , int y)
4  {
5      int z;
6      z = x + y ;
7      return z;
8  }
9
10 int main()
11 {
12     sumar (15 , 4);
13
14 }
```

¿Qué sucede con estas variables cuando la función “**sumar()**” finaliza?

**Las variables locales de la función “sumar” son destruidas.**

**Destruir es “quitarle” la memoria que le había sido asignada en la declaración**

- Tanto la creación como la destrucción de las variables fueron en tiempo de ejecución.
- La duración de las variables es una propiedad determinada en tiempo de ejecución.

# Variables locales: duración (storage duration)

```
1  #include <stdio.h>
2
3  int sumar (int x , int y)
4  {
5      int z;
6      z = x + y ;
7      return z;
8  }
9
10 int main()
11 {
12
13     int k ;
14
15     sumar (15 , 4);
16
17     return 0;
18 }
```



Cuando se ejecuta **main()** comienza el tiempo de ejecución



# Variables locales: duración (storage duration)

```
1  #include <stdio.h>
2
3  int sumar (int x , int y)
4  {
5      int z;
6      z = x + y ;
7      return z;
8  }
9
10 int main()
11 {
12     int k ;
13     sumar (15 , 4);
14
15     return 0;
16
17
18 }
```

La **duración** de “k” empieza acá.



# Variables locales: duración (storage duration)


```
1  #include <stdio.h>
2
3  int sumar (int x , int y)
4  {
5      int z;
6      z = x + y ;
7      return z;
8  }
9
10 int main()
11 {
12
13     int k ;
14
15     sumar (15 , 4);
16
17     return 0;
18 }
```

“k” sigue en memoria mientras se ejecuta la función “**sumar()**”



# Variables locales: duración (storage duration)

```
1  #include <stdio.h>
2
3  int sumar (int x , int y)
4  {
5      int z;
6      z = x + y ;
7      return z;
8  }
9
10 int main()
11 {
12
13     int k ;
14
15     sumar (15 , 4);
16
17     return 0;
18 }
```



“**k**” sigue en memoria mientras se ejecuta la función “**sumar()**”

# Variables locales: duración (storage duration)

```
1  #include <stdio.h>
2
3  int sumar (int x , int y)
4  {
5      int z;
6      z = x + y ;
7      return z;
8  }
9
10 int main()
11 {
12
13     int k ;
14
15     sumar (15 , 4);
16
17     return 0;
18 }
```

“k” sigue en memoria mientras se ejecuta la función “**sumar()**”

Acá empieza la **duración** de “z”

Acá finaliza la **duración** de “z”

# Variables locales: duración (storage duration)

```
1  #include <stdio.h>
2
3  int sumar (int x , int y)
4  {
5      int z;
6      z = x + y ;
7      return z;
8  }
9
10 int main()
11 {
12
13     int k ;
14
15     sumar (15 , 4);
16
17     return 0;
18 }
```



Cuando finaliza el **main()** “k” se elimina junto con la finalización del programa.

Por lo tanto, “z” existió durante la ejecución de **sumar()** y “k” durante la ejecución del **main()** .


*La variables locales existen únicamente durante la ejecución del bloque de código en la que se definieron.*

Esto se resume en la siguiente definición:

**Las variables locales tienen duración automática.**

# Variables locales: duración (storage duration)

Tipo	Ejemplo	Duración
local	int y;	automática

A close-up photograph of the eyepiece section of a surveying instrument, likely a theodolite or level. The instrument is white and has two large, circular eyepieces with black lenses. The background is a blurred view of a body of water with a distant, hazy landmass on the horizon. The text "Alcance (scope)" is overlaid in white, bold font across the center of the eyepieces.

**Alcance (scope)**

# Alcance (scope)

El **alcance** se refiere a la región del código en la que una variable puede ser referenciada.

Dicho de otra forma, desde que partes del código se puede acceder a una variable.

El alcance puede ser:

- **Alcance de Bloque:** La variable es visible solo dentro del bloque `{ ... }` donde se declara.
- **Alcance de Archivo:** La variable es visible desde el punto de su declaración hasta el final del archivo.

A continuación vamos a analizar y “clasificar” el **alcance** de las variables locales.

Tipo	Ejemplo	Duración	Alcance
local	int y;	automática	



# Variables locales: alcance (scope)

```
1  #include <stdio.h>
2
3  int sumar (int x , int y)
4  {
5      int z;
6      z = x + y ;
7      return z;
8  }
9
10 int main()
11 {
12
13     int k ;
14
15     sumar (15 , 4);
16
17     return 0;
18 }
```

“k” no esta al alcance de la función “**sumar()**”

Una vez declarada; “k” esta al alcance de la función **main()**

“k” sale del alcance y deja de poder ser usado

# Variables locales: alcance (scope)

## ¿funciona?

```
1  #include <stdio.h>
2
3  int sumar_cuadrados (int x, int y)
4  {
5      x *= x;
6      y *= y;
7
8      return (x + y);
9  }
10
11 int main(void)
12 {
13
14     int x , y ;
15
16     x = 2  ;
17     y = 3  ;
18
19
20     printf ("La suma de los cuadrados de %d y %d es %d", x , y , sumar_cuadrados(x,y));
21
22     return 0;
23 }
24
```

Funciona debido a que “**x**” e “**y**” tienen diferentes **alcances** en “**main()**” y la función “**sumar\_cuadrados()**”

Podríamos pensar que las variables locales tienen **alcance** de función.

Pero esto no es así desde el estándar **C99**

# Variables locales: alcance (scope)

```
1 #include <stdio.h>
2
3
4 int main()
5 {
6     int x = 10, z;
7     {
8         int y = 11;
9         y++;
10    }
11
12    z = x + y;
13
14 }
15
```

Variable 'y' set but not us...

Use of undeclared identifier 'y'

No funciona

Las variables locales tienen alcance de bloque

# Variables locales: alcance (scope)

```
1  #include <stdio.h>
2
3
4  int main(void)
5  {
6
7      int x , y ;
8
9      x = 2 ;
10     y = 3 ;
11     {
12         int x , y ;
13         x = 5 ;
14         y = 6 ;
15         printf ("Dentro del bloque %d y %d \n", x, y);
16     }
17
18
19     printf ("Fuera del bloque %d y %d \n", x , y );
20
21     return 0;
22 }
23
```

Dentro del bloque 5 y 6  
Fuera del bloque 2 y 3

Las variables locales tienen alcance  
de bloque

Al tener el mismo identificador las variables dentro del bloque  
anidado ocultan al compilador las variables definidas en el




# Variables locales: alcance (scope)

```
1  #include <stdio.h>
2
3
4  int main(void)
5  {
6
7      int i=121;
8      printf ("El valor de i del main %d \n", i);
9
10     for (int i=0; i<3; i++)
11     |     printf ("El valor de i dentro del bloque for %d \n", i);
12
13
14     printf ("El valor de i del main %d \n", i);
15
16     return 0;
17 }
18
```

```
El valor de i del main 121
El valor de i dentro del bloque for 0
El valor de i dentro del bloque for 1
El valor de i dentro del bloque for 2
El valor de i del main 121
```

**Las variables locales tienen alcance de bloque**

# Variables locales: duración y alcance

- **Duración:** propiedad de la variable en tiempo de **ejecución**.
- **Alcance:** en tiempo de **compilación**  Llamar a una variable fuera de su alcance nos dará un **error de compilación**.

Tipo	Ejemplo	Duración	Alcance
local	int y;	automática	De bloque



A yellow combine harvester is shown from a low angle, moving through a vast field of golden wheat. The harvester is positioned in the upper right quadrant of the frame. The field stretches out to the horizon, with a few trees visible in the distance. The sky is a deep, hazy blue, suggesting the time is either dawn or dusk. The overall mood is peaceful and industrious.

# Repaso de algunos temas

# Repaso de algunos temas

- A continuación repasaremos unos temas que tenemos que tener en cuenta para comprender lo que sigue.



# Prototipos de función – declaración anticipada

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



# Prototipos de función – declaración anticipada

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

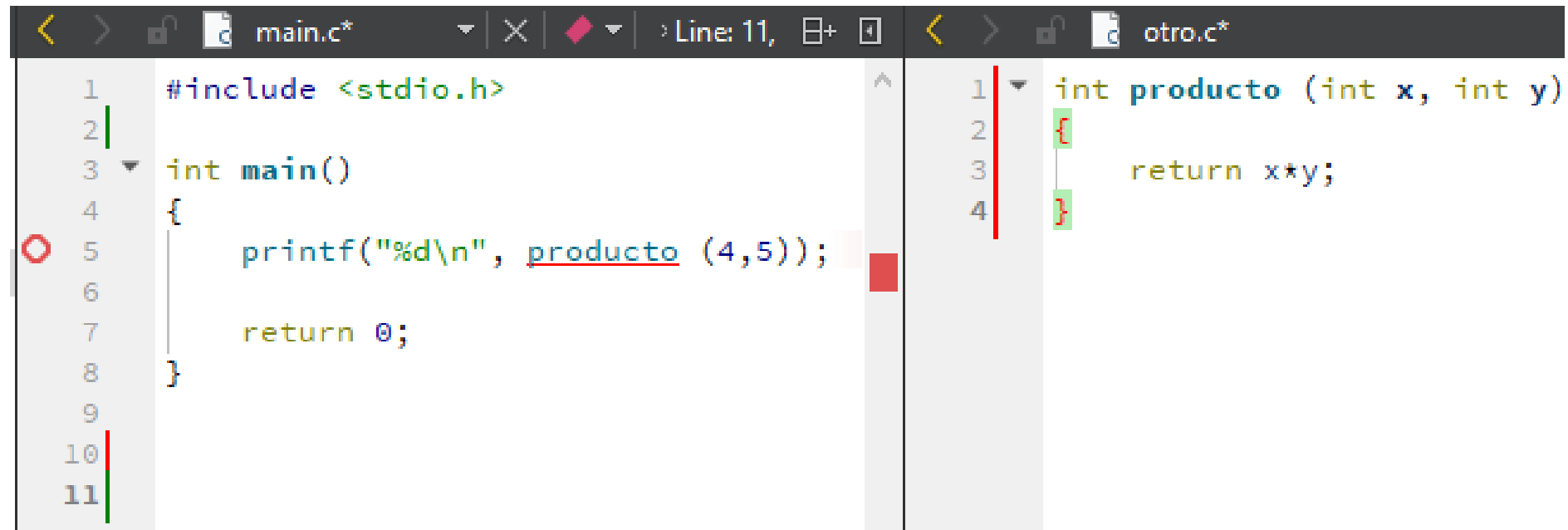


# Prototipos de función – declaración anticipada

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



# Programa dividido en varios archivos



```
main.c*
1  #include <stdio.h>
2
3  int main()
4  {
5      printf("%d\n", producto (4,5));
6
7      return 0;
8  }
9
10
11

otro.c*
1  int producto (int x, int y)
2  {
3      return x*y;
4  }
```

Los archivos fueron creados correctamente desde el IDE, y corresponden al mismo proyecto. ¿Esto funciona?

NO

# Programa dividido en varios archivos

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

No funciona por exactamente la misma razón que no funcionaba este ejemplo.  
Falta el prototipo

# Programa dividido en varios archivos

```
main.c
1  #include <stdio.h>
2
3  int producto (int , int );
4
5  int main()
6  {
7      printf("%d\n", producto (4,5));
8
9      return 0;
10 }
```

```
otro.c
1  int producto (int x, int y)
2  {
3      return x*y;
4  }
5
```





# **Variables globales – duración y alcance**

# Variables globales – duración y alcance

A continuación vamos a analizar y “clasificar” el alcance y la duración de las variables globales.

Tipo	Ejemplo	Duración	Alcance
local	int y;	automática	De bloque
global			



# Variables globales – duración y alcance

```
1  #include <stdio.h>
2
3  int g_com ;
4
5  void sumaVariable_g ()
6  {
7      g_com ++ ;
8  }
9  int main()
10 {
11     printf("%d\n", g_com);
12     sumaVariable_g();
13     printf("%d\n", g_com);
14     return 0;
15 }
16
```

Se declaran después de los **#include** al principio del archivo

Es buena práctica diferenciarlas, por ejemplo “g\_”

Automáticamente se inicializan en 0

**Tienen alcance de archivo**

Se inician cuando empieza a ejecutarse el archivo y duran hasta que finaliza.

**Las variables globales son de duración estática.**

# Variables globales – duración y alcance

Tipo	Ejemplo	Duración	Alcance
local	int y;	Automática	De bloque
global	int g_y;	Estática	De archivo



**Vinculación (linkage)**

# Vinculación (linkage)

Se refiere a si una variable (o función) puede ser llamada desde otros archivos del proyecto.

- **Vinculación Externa (External Linkage)**: Referenciable desde otros archivos.
- **Vinculación Interna (Internal Linkage)**: Solo referenciable dentro del mismo archivo.
- **Sin Vinculación (No Linkage)**: No referenciable fuera del bloque de código.

A continuación vamos a analizar y “clasificar” la **vinculación** de las variables locales y globales

Tipo	Ejemplo	Duración	Alcance	Vinculación
local	int y;	Automática	De bloque	
global	int g_y;	Estática	De archivo	

# Vinculación (linkage) de una variable local

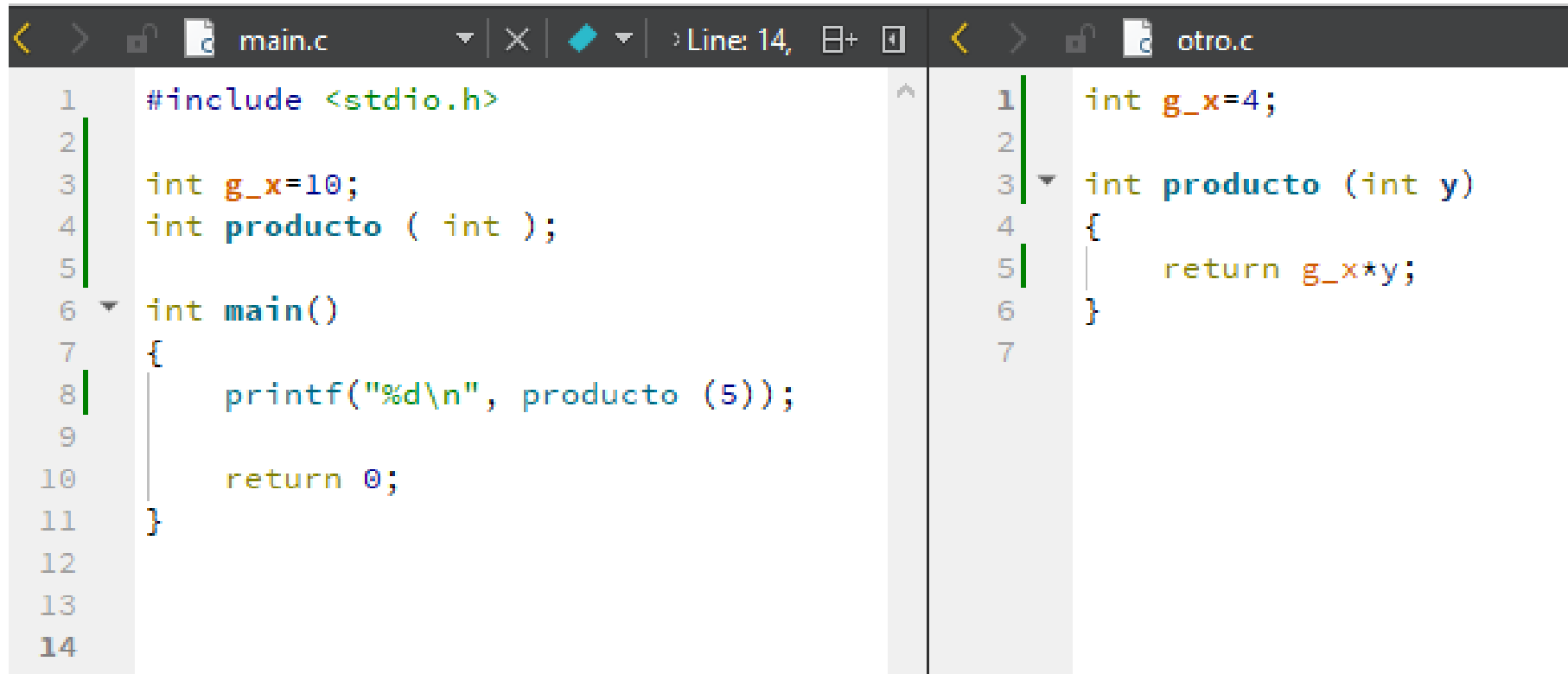
```
1 #include <stdio.h>
2
3
4 int main()
5 {
6     int x = 10, z;
7     {
8         int y = 11;
9         y++;
10    }
11
12    z = x + y;
13
14 }
15
```

Solo pueden verse desde el bloque donde fueron creadas

Tipo	Ejemplo	Duración	Alcance	Vinculación
local	int y;	Automática	De bloque	Sin vinculación
global	int g_y;	Estática	De archivo	

# Vinculación (linkage) de una variable global

Pueden tener vinculación interna o externa, depende como las declaremos.



```
main.c
1  #include <stdio.h>
2
3  int g_x=10;
4  int producto ( int );
5
6  int main()
7  {
8      printf("%d\n", producto (5));
9
10     return 0;
11 }
12
13
14

otro.c
1  int g_x=4;
2
3  int producto (int y)
4  {
5      return g_x*y;
6  }
7
```

Aquí ocurre un error de linkeo. Ya que, declarada de esta forma, **la variable global tiene vinculación EXTERNA**. Estamos definiendo dos veces la misma variable, como son en archivos diferentes el que se queja es el **linker**.

# Vinculación (linkage) de una variable global

Tipo	Ejemplo	Duración	Alcance	Vinculación
local	int y;	Automática	De bloque	Sin vinculación
global	int g_y;	Estática	De archivo	Externa

# Vinculación (linkage) de una variable global

```
main.c*
1  #include <stdio.h>
2
3  int g_x=10;
4  int producto ( int );
5
6  int main()
7  {
8      printf("%d\n", producto (5));
9
10     return 0;
11 }

otro.c*
1  int producto (int y)
2  {
3      return g_x*y;
4  }
5
```

Entonces si declarada de esta forma, tiene **vinculación EXTERNA**. ¿Por qué esto no funciona?

Nuevamente, como en el caso de las funciones, falta una **definición anticipada**.



# Vinculación (linkage) de una variable global

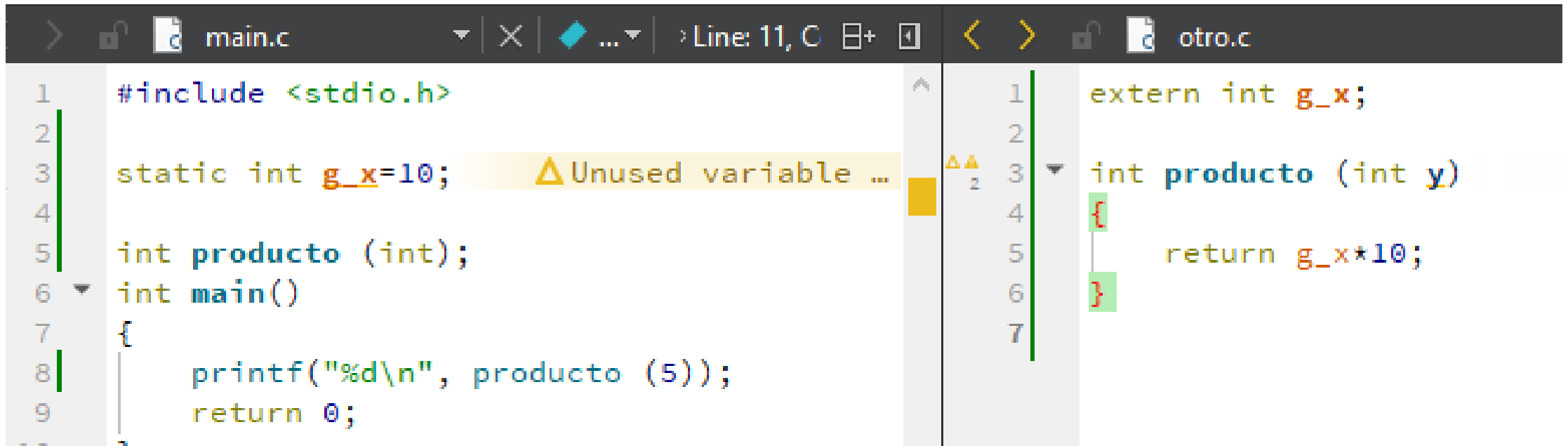
```
< > main.c Line: 14, + - < > otro.c
1  #include <stdio.h>
2
3  int g_x=10;
4  int producto ( int );
5
6  int main()
7  {
8      printf("%d\n", producto (5));
9
10     return 0;
11 }
12

1  extern int g_x;
2
3  int producto (int y)
4  {
5      return g_x*y;
6  }
7
```

La declaración anticipada lleva la palabra reservada “**extern**”.  
No se le debe asignar ningún valor en la declaración anticipada.

50

# Vinculación (linkage) de una variable global



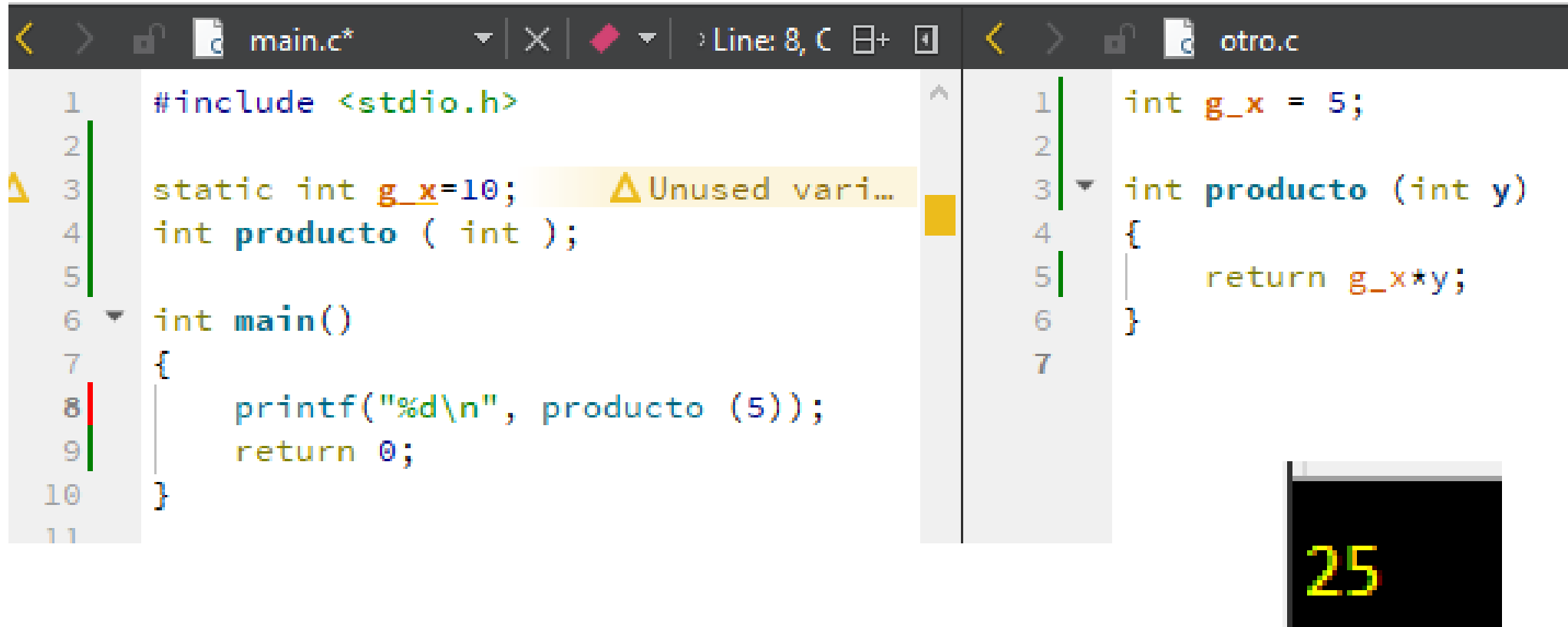
```
main.c
1 #include <stdio.h>
2
3 static int g_x=10;  Δ Unused variable ...
4
5 int producto (int);
6 int main()
7 {
8     printf("%d\n", producto (5));
9     return 0;
10 }

otro.c
1 extern int g_x;
2
3 int producto (int y)
4 {
5     return g_x*10;
6 }
7
```

Si queremos que una variable global tenga **vinculación INTERNA**, la debemos declarar agregándole la palabra “**static**”

**NO ANDA**

# Vinculación (linkage) de una variable global



```
main.c*
1  #include <stdio.h>
2
3  static int g_x=10;  Δ Unused vari...
4  int producto ( int );
5
6  int main()
7  {
8      printf("%d\n", producto (5));
9      return 0;
10 }
11

otro.c
1  int g_x = 5;
2
3  int producto (int y)
4  {
5      return g_x*y;
6  }
7
```

25

Si queremos que una variable global tenga **vinculación INTERNA**, la debemos declarar agregándole la palabra **“static”**

La variable `g_x` del `main()` con vinculación interna esta oculto a la variable global con vinculación externa. Practica no recomendada, pero posible.

# Vinculación (linkage) de una variable global

Tipo	Ejemplo	Duración	Alcance	Vinculación
local	int y;	Automática	De bloque	Sin vinculación
global	int g_y;	Estática	De archivo	Externa
global interna	static int g_y;	Estática	De archivo	Interna

# Vinculación (linkage) de funciones

Como ya dijimos el **linkage** también es un atributo de las funciones.

Por default todas las funciones tienen vinculación externa, por eso, si usamos los prototipos de función podemos llamar a las funciones desde cualquier archivo del proyecto.

Al igual que con las variables globales, podemos crear funciones con vinculación interna agregando la palabra reservada **static**.

```
main.c*
1  #include <stdio.h>
2
3  int producto ( int, int );
4
5  int main()
6  {
7      printf("%d\n", producto (5 , 6 ));
8      return 0;
9  }
10

otro.c
1
2
3  static int producto (int x, int y)
4  {
5      return x*y;
6  }
7
```

NO COMPILA

# Variables locales estáticas

Tipo	Ejemplo	Duración	Alcance	Vinculación
local	int y;	Automática	De bloque	Sin vinculación
local estática	static int y;			
global	int g_y;	Estática	De archivo	Externa
global interna	static int g_y;	Estática	De archivo	Interna

# Variables locales estáticas

```
1  #include <stdio.h>
2
3  int suma ()
4  {
5      int i=0;
6      printf ("%d\n", i );
7      i++;
8  }
9
10
11 int main()
12 {
13     suma();
14     suma();
15     suma();
16     suma();
17     getchar();
18
19 }
```



# Variables locales estáticas

```
1  #include <stdio.h>
2
3  int suma ()
4  {
5      static int i=0;
6      printf ("%d\n", i );
7      i++;
8  }
9
10
11 int main()
12 {
13     suma();
14     suma();
15     suma();
16     suma();
17     getchar();
18 }
```



```
0
1
2
3

```



# Variables locales estáticas

Tipo	Ejemplo	Duración	Alcance	Vinculación
local	<code>int y;</code>	Automática	De bloque	Sin vinculación
local estática	<code>static int y;</code>	Estática	De bloque	Sin vinculación
global	<code>int g_y;</code>	Estática	De archivo	Externa
global interna	<code>static int g_y;</code>	Estática	De archivo	Interna

**OJO: Solo modifica la duración, no la vinculación ni el alcance.**

# Próximo tema

- Cuando definimos la duración, dijimos que podía ser:
  - **Automática**
  - **Estática**
  - **Dinámica**
- Nos faltó ver el tercero de los ítems. Para esto vamos a seguir con **Gestión de Memoria dinámica**.