

Contenido

[**Objetivo** 3](#_Toc528705220)

[**Actividades:** 3](#_Toc528705221)

[**Introducciòn:** 3](#_Toc528705222)

[**Actividades** 10](#_Toc528705223)

[**Conclusión:** 12](#_Toc528705224)

# **Objetivo**

Aprender las técnicas básicas de depuración de programas en C para revisar de manera precisa el flujo de ejecución de un programa y el valor de las variables; en su caso, corregir posibles errores.

# **Actividades:**

* Revisar, a través de un depurador, los valores que va tomando una variable en un programa escrito en C, al momento de ejecutarse.
* Utilizando un depurador, revisar el flujo de instrucciones que se están ejecutando en un programa en C, cuando el flujo depende de los datos de entrada.

# **Introducción:**

Para depurar un programa tiene que estar compilado.

Para la depuración, la compilación debe ser:

* Eficiente
* Óptimo

La depuración de un programa es útil cuando:

* El programa tiene algún fallo:
* el programa no muestra los resultados que se esperan para cierta entrada de datos debido a que el programador cometidó algún error durante el proceso de diseño.

Error:

Es humano cuando el mismo la se equivoca.

El programa tiene error de ejecución o defecto:

Cuando el programa está ejecutándose, este se detiene inesperadamente.

Algunas funciones básicas que tienen en común la mayoría de los depuradores son las siguientes:

* Punto de ruptura: también conocido por su traducción al inglés *breakpoint*, sirve para detener la ejecución del programa en algún punto indicado previamente por medio del número de línea. Como la ejecución del programa es más rápida de lo que podemos visualizar y entender, se suelen poner puntos de ruptura para conocer ciertos parámetros de la ejecución como el valor de las variables en determinados puntos del programa. También sirve para verificar hasta qué punto el programa se ejecuta sin problemas y en qué parte podría existir el error, esto es especialmente útil cuando existe un error de ejecución.

Depurar un programa significa someterlo a un ambiente de ejecución controlado por medio de herramientas dedicadas a ello. Este ambiente permite conocer exactamente el flujo de ejecución del programa, el valor que las variables adquieren, la pila de llamadas a funciones, entre otros aspectos. Es importante poder compilar el programa sin errores antes de depurarlo.

***Glosario IEEE610***

**Error**. Se refiere a una acción humana que produce o genera un resultado incorrecto.

**Defecto (Faul).** Es la manifestación de un error en el software. Un defecto es encontrado porque causa una Falla (afilare).

**Falla (afilare).** Es una desviación del servicio o resultado esperado.

Depurador o “dejugar”.

El depurador es un ejecutable cuya misión es permitir la ejecución controlada de un segundo ejecutable. Se comporta como un envoltorio dentro del cual se desarrolla una ejecución normal de un programa, pero a la vez permite realizar una serie de operaciones específicas para visualizar el entorno de ejecución en cualquier instante.

Permite:

* Ejecutar un programa línea a línea
* Detener la ejecución temporalmente en una línea de código concreta
* Detener temporalmente la ejecución bajo determinadas condiciones
* Visualizar el contenido de las variables en un determinado momento de la ejecución
* Cambiar el valor del entorno de ejecución para poder ver el efecto de una corrección en el programa

Uno de los depuradores más utilizados en entornos Linux es cd (Dejugar de GNU). En este documento se describen los comandos más relevantes de este depurador para ser utilizados con un programa escrito en C.

La depuración de un programa es útil cuando:

* Se desea optimizar el programa:

no basta que el programa se pueda compilar y se someta a pruebas que demuestren que funciona correctamente. Debe realizarse un análisis exhaustivo del mismo en ejecución para averiguar cuál es su flujo de operación y encontrar formas de mejorarlo (reducir el código, utilizar menos recursos llegando a los mismos resultados, hacer menos rebuscado al algoritmo), o bien, encontrar puntos donde puede fallar con ciertos tipos de entrada de datos.

* El programa tiene algún fallo:

el programa no muestra los resultados que se esperan para cierta entrada de datos debido a que el programador cometió algún error durante el proceso de diseño. Muchas veces encontrar este tipo de fallos suele ser difícil, ya sea porque la percepción del programador no permite encontrar la falla en su diseño o porque la errata es muy pequeña, pero crucial. En este caso es de mucha utilidad conocer paso a paso cómo se ejecutan las estructuras de control, qué valor adquieren las variables, etc.

* El programa tiene un error de ejecución o defecto:

cuando el programa está ejecutándose, éste se detiene inesperadamente. Suele ocurrir por error en el diseño o implementación del programa en las que no se contemplan las limitaciones del lenguaje de programación o el equipo donde el programa se ejecuta. Como el programa se detiene inesperadamente, no se conoce la parte del programa donde se provoca el defecto, teniendo que recurrir a la depuración para encontrarlo. El más común de este tipo de defecto es la “violación de segmento”.

Arranque y parada del “Dejugar”.

Para que un programa escrito en C pueda ser manipulado por **cd** es preciso realizar una compilación que incluya como parte del ejecutable, un conjunto de datos adicionales. Esto se consigue incluyendo la opción **-g** al invocar el compilador:

gcc -Wall -g -o gdb\_use gdb\_use.c

Una vez creado el fichero ejecutable se invoca el depurador con el comando:

gdb gdb\_use

Tras arrancar el depurador se muestra por pantalla un mensaje seguido del pronto (cd):

$ gdb gdb\_use

GNU gdb (GDB) 7.0-ubuntu

Copyright (C) 2009 Free Software Foundation, Inc.

License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>

This is free software: you are free to change and redistribute it.

There is NO WARRANTY, to the extent permitted by law. Type "show copying"

and "show warranty" for details.

This GDB was configured as "i486-linux-gnu".

For bug reporting instructions, please see:

<http://www.gnu.org/software/gdb/bugs/>...

Reading symbols from /home/test/gdb\_use...done.

(gdb)

En este instante, el programa depurador ha arrancado, pero la ejecución del programa gdb\_use (que se ha pasado como primer argumento) todavía no. La interacción con el depurador se realiza a través de comandos introducidos a continuación del prompt, de forma similar a como se proporcionan comandos a un shell o intérprete de comandos en Linux.

Para arrancar la ejecución del programa se utiliza el comando run (o su abreviatura r). Tras introducir este comando, el programa se ejecuta de forma normal. Si se produce un error o una interrupción en la ejecución, el programa se detiene y el control vuelve al depurador, por lo que se muestra por pantalla de nuevo el prompt (gdb). Por ejemplo:

(gdb) r

Starting program: /home/test/gdb\_use

Program received signal SIGSEGV, Segmentation fault.

0x08048441 in check (table=0x804a008) at gdb\_use.c:29

29 if ((table[y].data + 1) != table[y].next->data)

(gdb)

Cuando se produce un error, el depurador muestra el lugar en el código responsable de ese error. Es posible que el error se produzca en una rutina interna del sistema. En tal caso, no quiere decir que el error esté en tal librería, sino que se ha manifestado en la librería por causas debidas al código del programa.

Si se desea detener un programa mientras se está ejecutando se debe pulsar Crtl-C (la tecla control, y mientras se mantiene pulsada, se pulsa C). La interrupción del programa es capturada por el depurador, y el control lo retoma su intérprete de comandos. En este instante, la ejecución del programa ha sido detenida pero no terminada. Prueba de ello, es que la ejecución puede continuarse mediante el comando continue (que se puede abreviar simplemente con la letra c).

Para salir del depurador se utiliza el comando quit (abreviado por la letra q). Si se pretende terminar la sesión del depurador mientras el programa está en ejecución se pide confirmación para terminar dicha ejecución.

(gdb) q

The program is running. Exit anyway? (y or n) y

$

El comando help muestra la información referente a todos los comandos y sus opciones. Si se invoca sin parámetros, se muestran las categorías en las que se clasifican los comandos. Si el comando va seguido del nombre de una categoría, proporciona información detallada sobre los comandos que contiene. Si se invoca seguido de un comando, describe su utilización.

Al correr GDB se entra a una línea de comandos. De acuerdo al comando es posible realizar distintas funciones de depuración:

*list* o *l:*

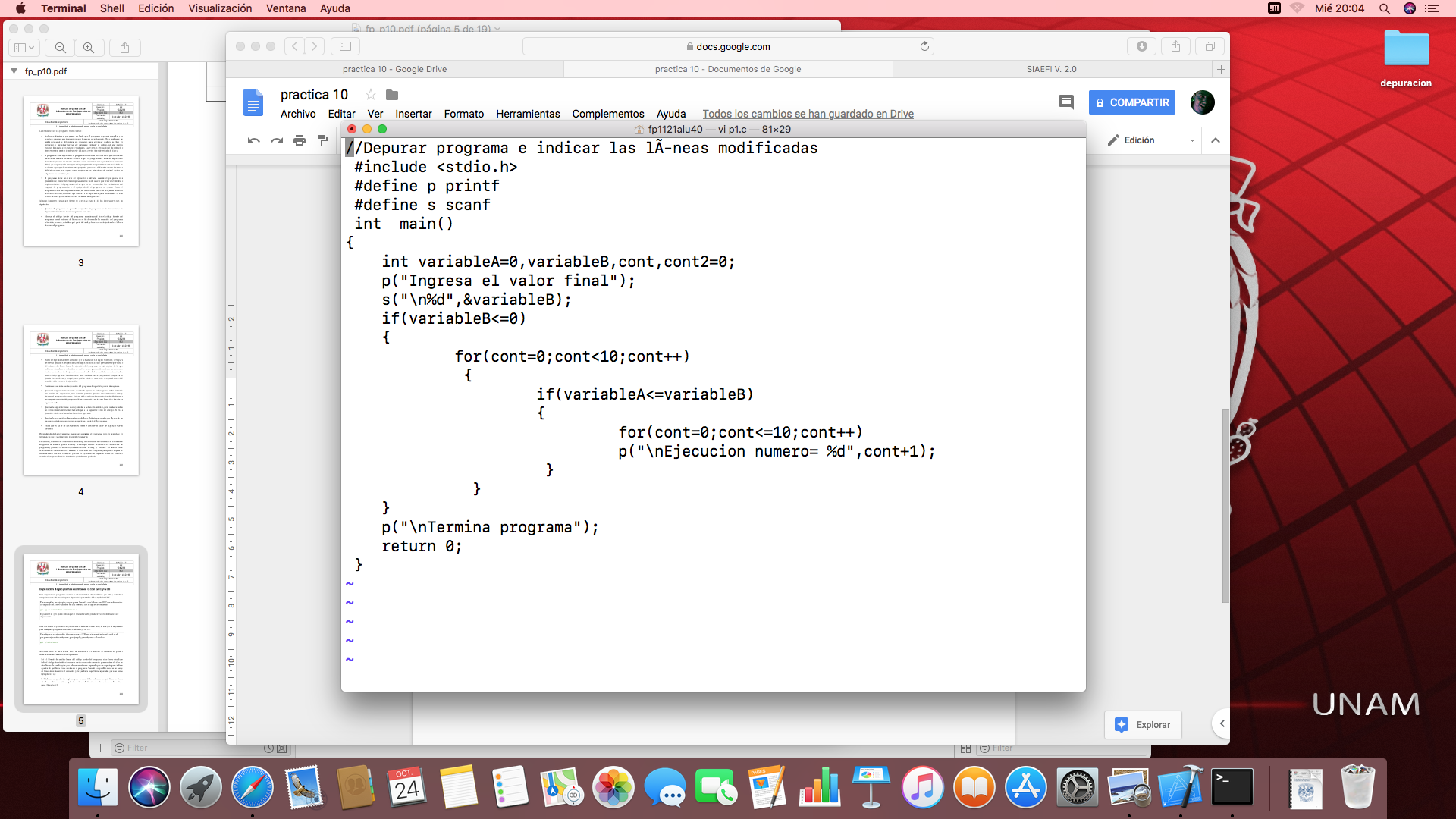
Permite listar diez líneas del código fuente del programa, si se desea visualizar todo el código fuente debe invocarse varias veces este comando para mostrar de diez en diez líneas. Se puede optar por colocar un número separado por un espacio para indicar a partir de qué línea desea mostrarse el programa. También es posible mostrar un rango de líneas introduciendo el comando y de qué línea a qué línea separadas por una coma.

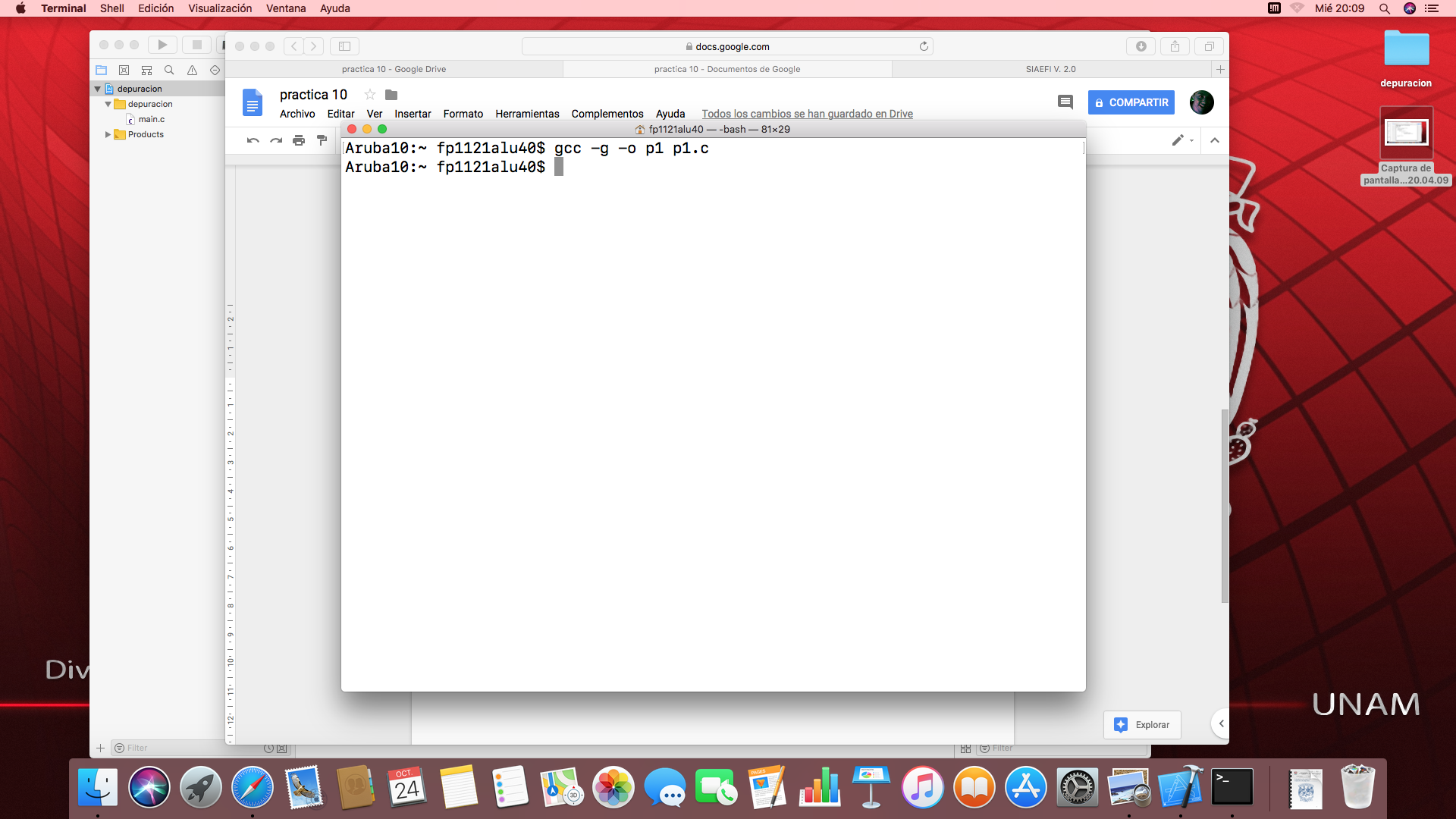
Ejemplo: *list 4,6*

*b:* Establece un punto de ruptura para lo cual debe indicarse en qué línea se desea establecer o bien también acepta el nombre de la función donde se desea realizar dicho paso.

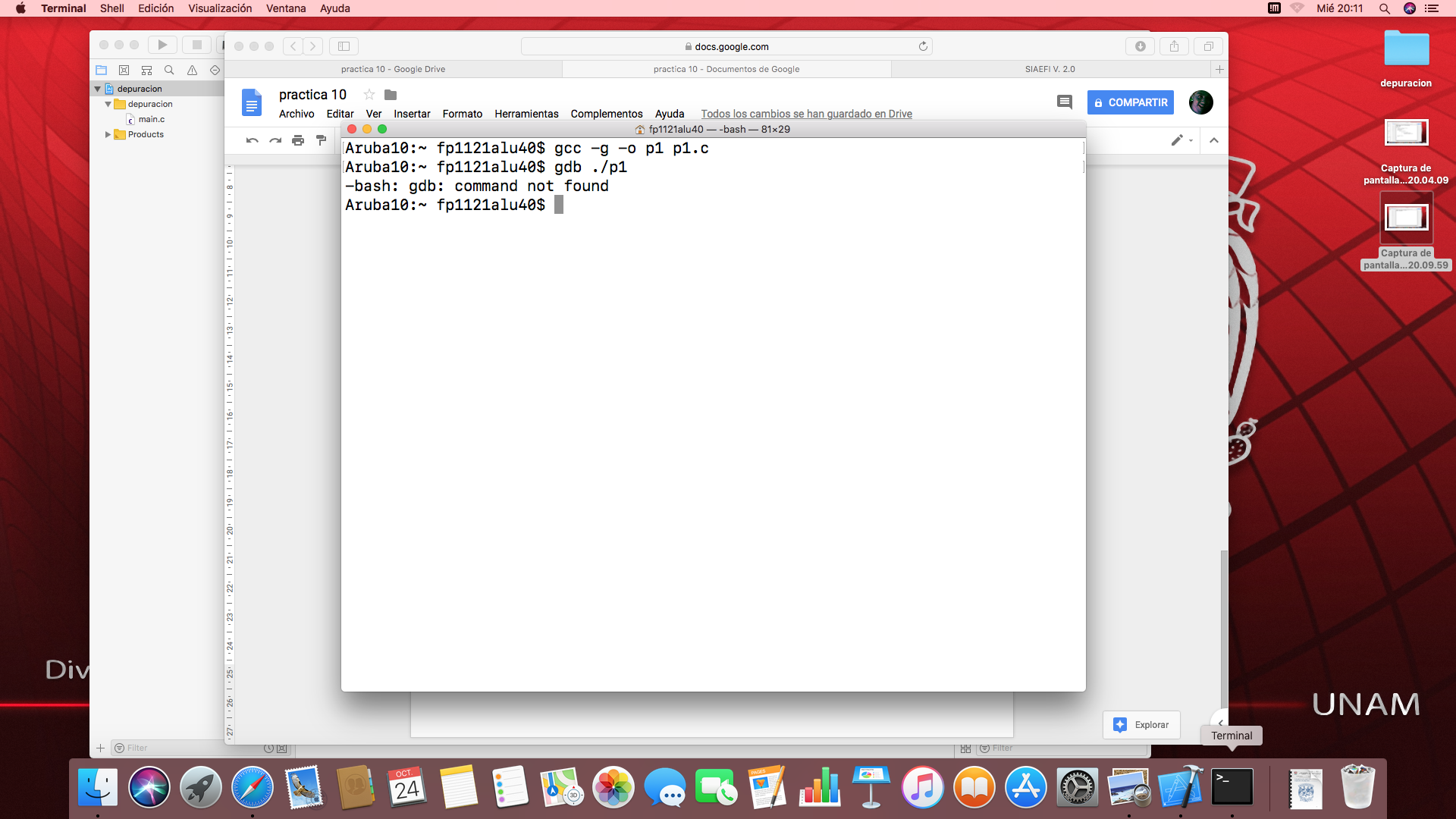
Ejemplo: *b 5*

* d o delete: Elimina un punto de ruptura, indicando cuál es el que debe eliminarse usando el número de línea. Ejemplo: d 5
* clear: Elimina todos los puntos de ruptura. Ejemplo: clear
* info line: Permite mostrar información relativa a la línea que se indique después del comando. Ejemplo: info line 8
* run o r: Ejecuta el programa en cuestión. Si el programa tiene un punto de ruptura se ejecutará hasta dicho punto, de lo contrario se ejecutará todo el programa.
* c: Continúa con la ejecución del programa después de un punto de ruptura.
* s: Continúa con la siguiente instrucción después de un punto de ruptura.
* n: Salta hasta la siguiente línea de código después de un punto de ruptura.
* p o print: Muestra el valor de una variable, para ello debe escribirse el comando y el nombre de la variable separados por un espacio. Ejemplo: p suma\_acumulada
* ignore: Ignora un determinado punto de ruptura indicándolo con el número de línea de código. Ejemplo: ignore 5
* q o quit: Termina la ejecución de GDB.
* GDB tiene más opciones disponibles que pueden consultarse con comandos como help o invocando desde la terminal del sistema man gdb.

Depurando

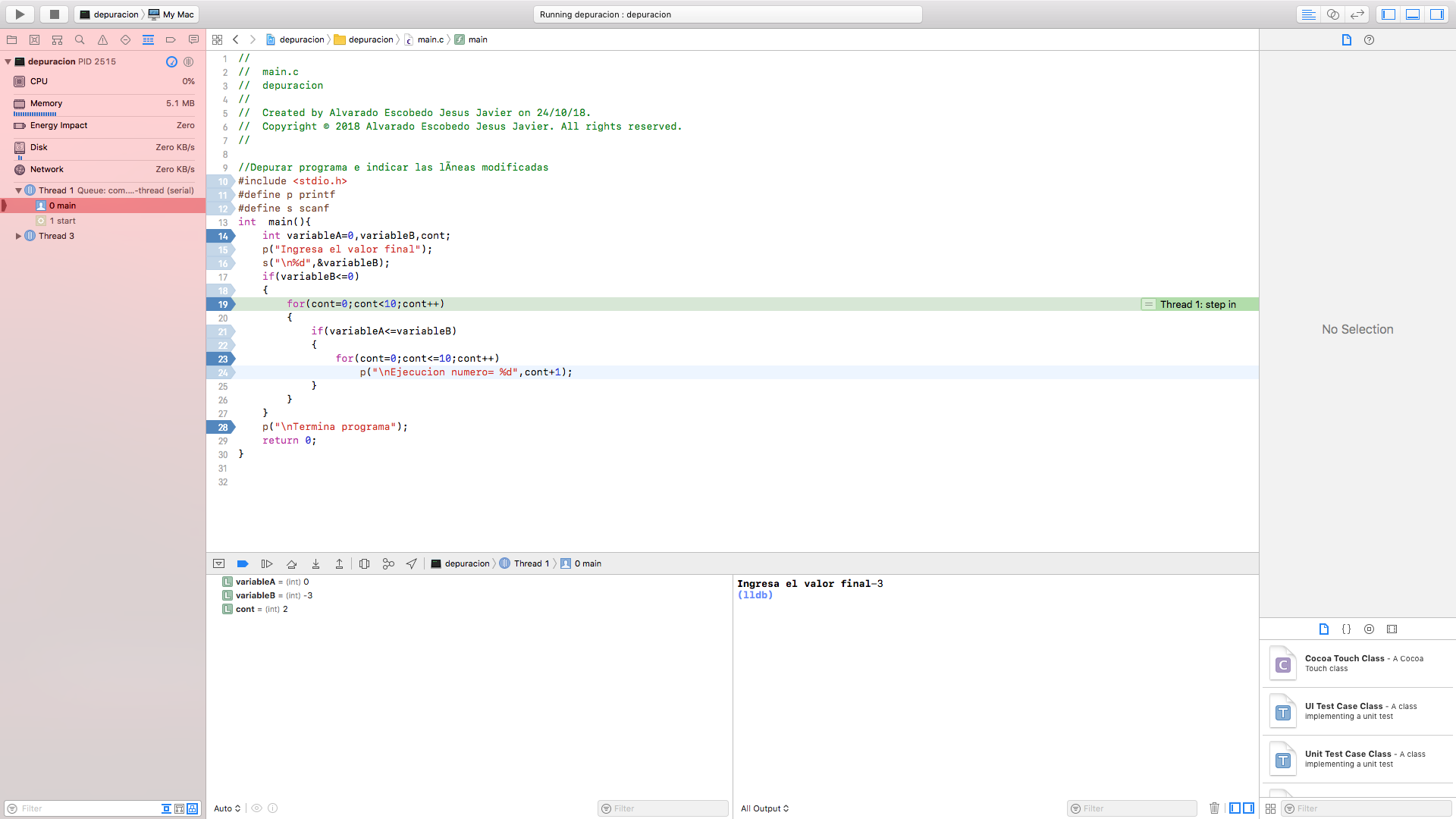


Activar depurador



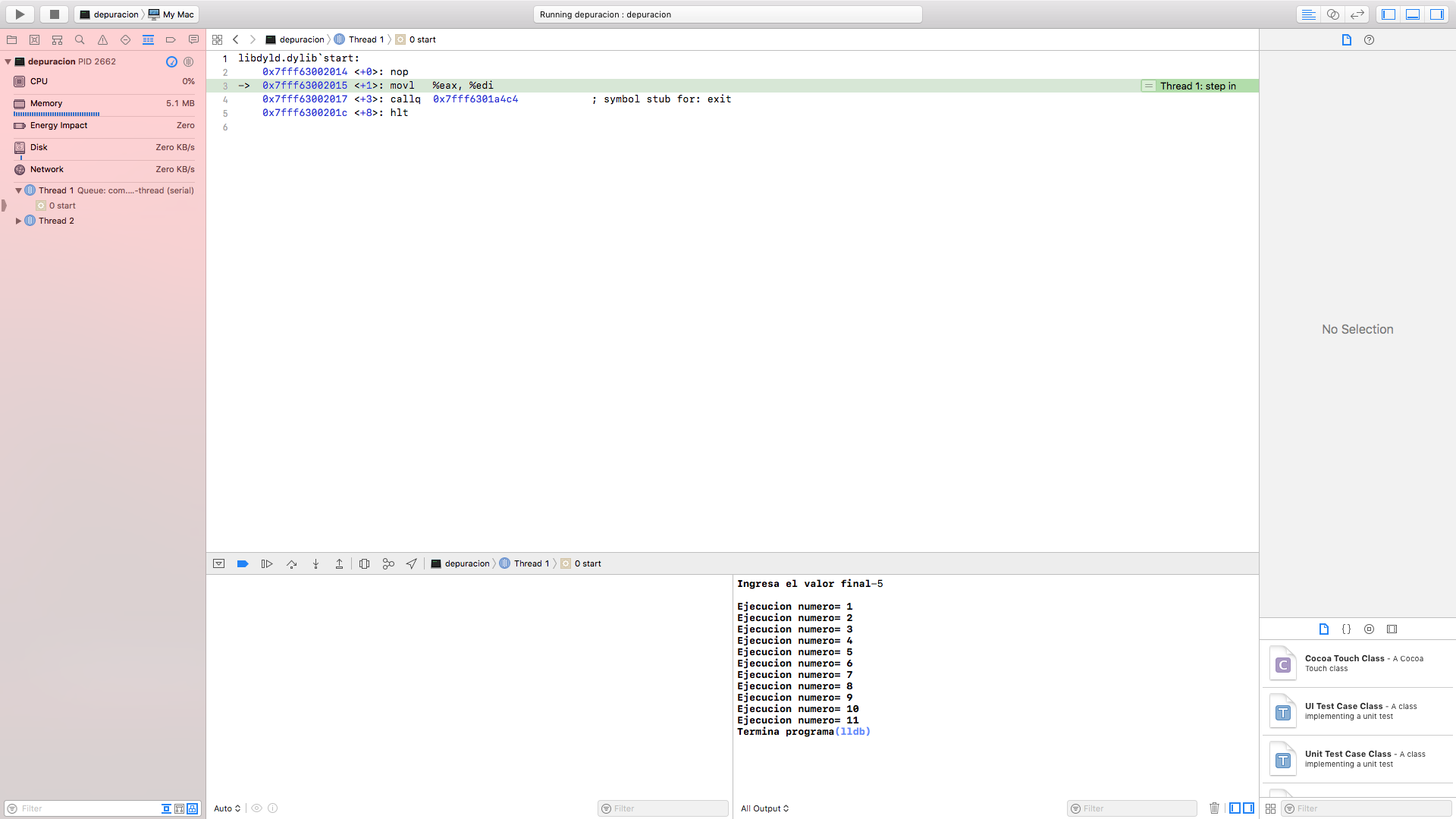
!!!!No tiene instalado el depurados¡¡¡¡¡¡

Uso de depurador en xcode



Contador aumentara hasta llegar a 10, con numeros negativos

Cambiamos de < a > en el condicial if.



# **Actividades**

Para el siguiente código fuente, utilizar algún entorno de depuración para encontrar la utilidad del programa y la funcionalidad de los principales comandos de depuración, como puntos de ruptura, ejecución de siguiente línea o instrucción.

#include <stdio.h>

void main()

{

int N, CONT, AS;

AS=0;

CONT=1;

printf("TECLEA UN NUMERO: ");

scanf("%i",&N);

while(CONT<=N)

{

AS=(AS+CONT);

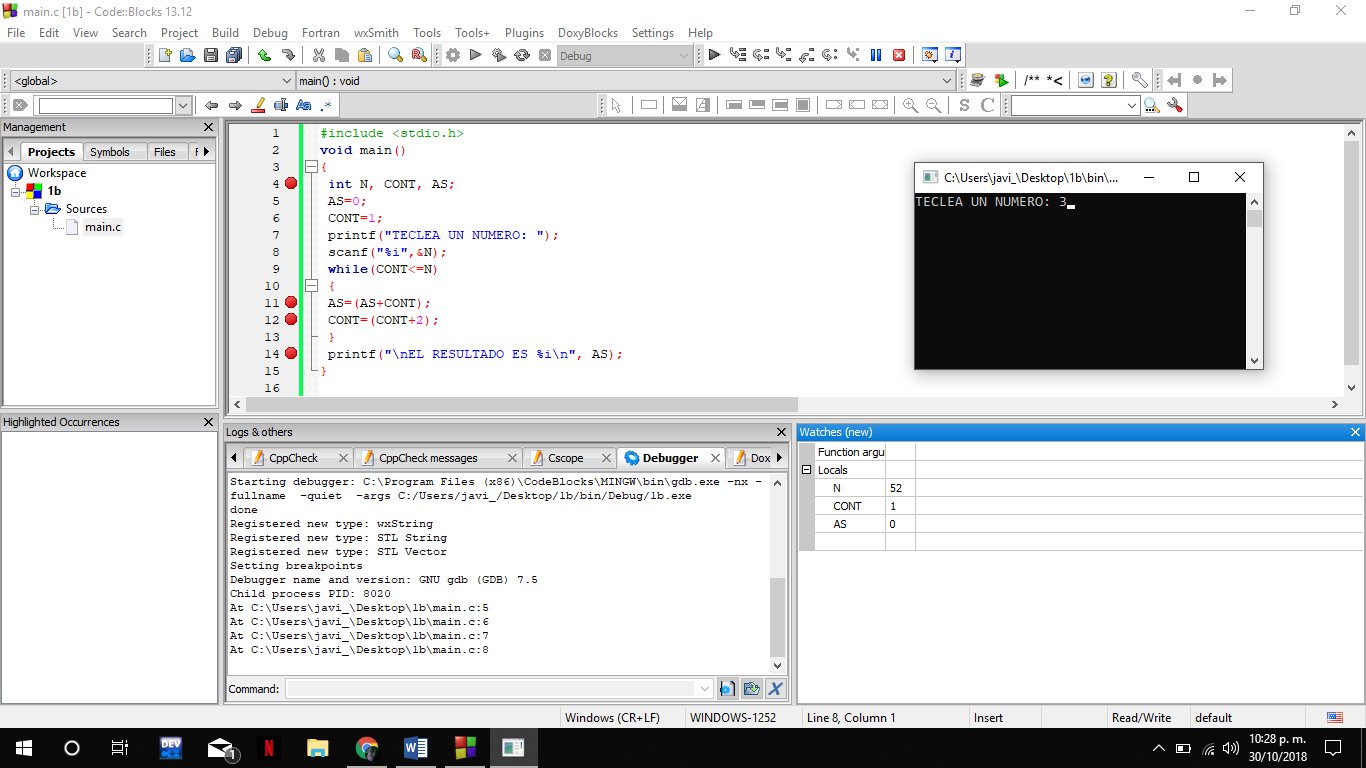
CONT=(CONT+2);

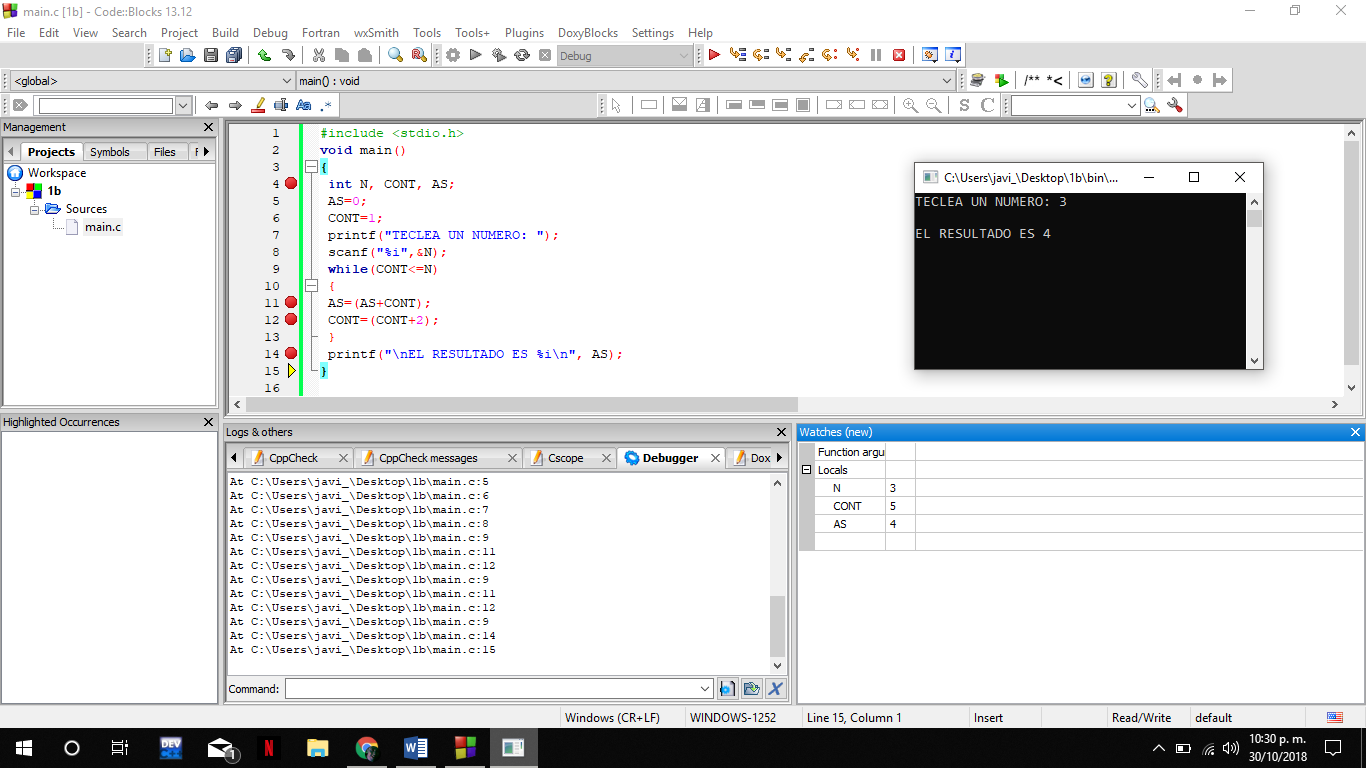
}

printf("\nEL RESULTADO ES %i\n", AS);

}

Funcionalidad del programa:





El siguiente programa debe mostrar las tablas de multiplicar desde la del 1 hasta la del 10. En un principio no se mostraba la tabla del 10, luego después de intentar corregirse sin un depurador dejaron de mostrarse el resto de las tablas. Usar un depurador de C para averiguar el funcionamiento del programa y corregir ambos problemas.

#include <stdio.h>

void main()

{

Int i,j;

for(i=1;i<10;i++){

printf(“\nTablas del %i\n”,i);

for(“j=1;j==10;j++);

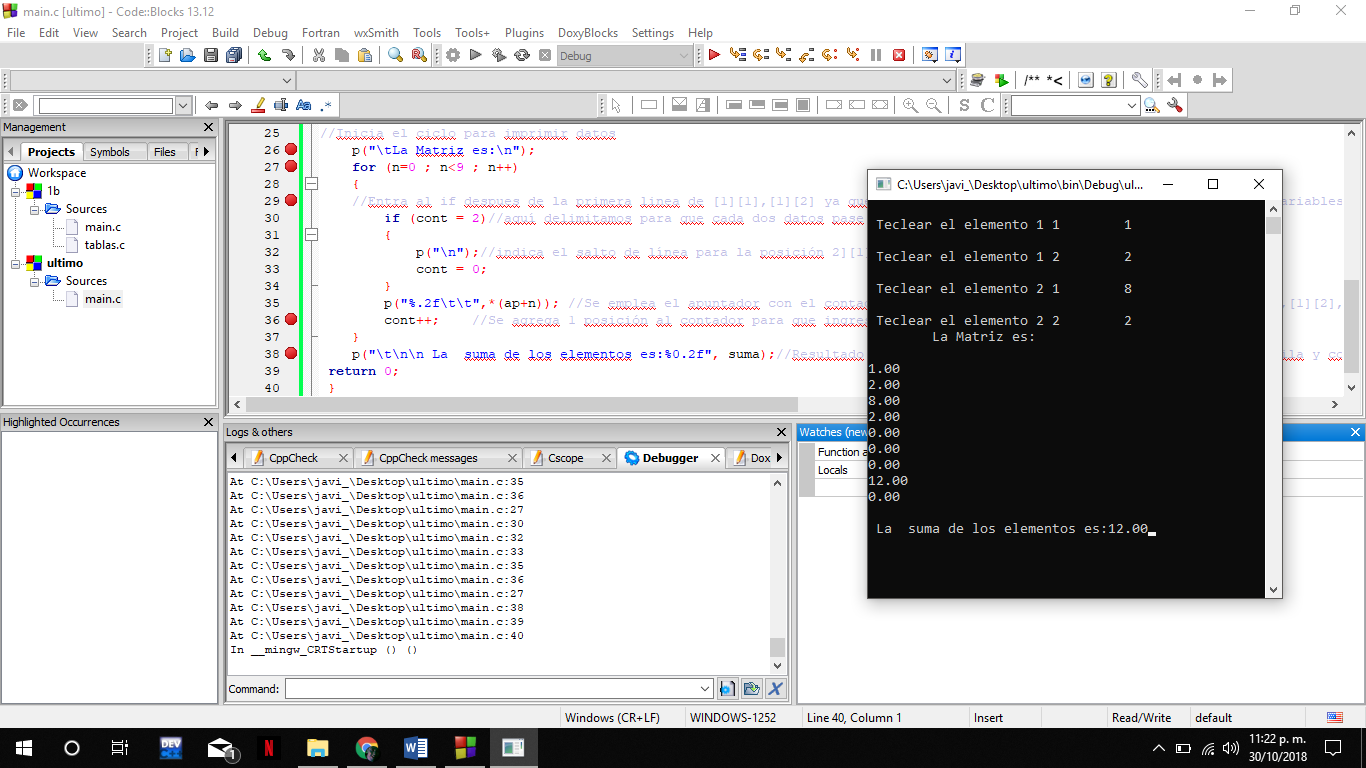
}

Printf(“%i X %i = %i\n”, i, j, i\*j );

}

}

}



# **Conclusión:**

Con el depurador pudimos hacer uso de puntos de ruptura y optimizar el programa

eficiencia

liberamos espacio de memoria

eliminamos variables sin uso

Todo eso lo logramos al poder ver valores de variables en cada Break-point y en el paso a paso.