



## Carátula para entrega de prácticas

Facultad de Ingeniería

Laboratorio de docencia

# Laboratorios de computación salas A y B

*Profesor:* M.I. Marco Antonio Martínez Quintana

*Asignatura:* Fundamentos de Programación

*Grupo:* 3

*No de Práctica(s):* 10. Depuración de programas.

*Integrante(s):* Hernández Vázquez Daniela

*No. de Equipo de  
cómputo empleado:* No Aplica

*No. de Lista o Brigada:* 26

*Semestre:* 1

*Fecha de entrega:* Lunes 7 de diciembre de 2020

*Observaciones:*

**CALIFICACIÓN:** \_\_\_\_\_

## Práctica 10: Depuración de programas.

### 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

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.

Antes de continuar, es necesario conocer las siguientes definiciones (extraídas del Glosario IEEE610) ya que son parte latente del proceso de Desarrollo de Software:

- **Error.** Se refiere a una acción humana que produce o genera un resultado incorrecto.
- **Defecto (Fault).** Es la manifestación de un error en el software. Un defecto es encontrado porque causa una Falla (failure).
- **Falla (failure).** Es una desviación del servicio o resultado esperado.

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".

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

- Ejecutar el programa: se procede a ejecutar el programa en la herramienta de depuración ofreciendo diversas opciones para ello.
- Mostrar el código fuente del programa: muestra cuál fue el código fuente del programa con el número de línea con el fin de emular la ejecución del programa sobre éste, es decir, se indica qué parte del código fuente se está ejecutando a la hora de correr el programa.
- 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.
- Continuar: continúa con la ejecución del programa después del punto de ruptura.
- Ejecutar la siguiente instrucción: cuando la ejecución del programa se ha detenido por medio del depurador, esta función permite ejecutar una instrucción más y detener el programa de nuevo. Esto es útil cuando se desea estudiar detalladamente una pequeña sección del programa. Si en la ejecución existe una llamada a función se ingresará a ella.
- Ejecutar la siguiente línea: es muy similar a la función anterior, pero realizará todas las instrucciones necesarias hasta llegar a la siguiente línea de código. Si en la ejecución existe una llamada a función se ignorará.
- Ejecutar la instrucción o línea anterior: deshace el efecto provocado por alguna de las funciones anteriores para volver a repetir una sección del programa.
- Visualizar el valor de las variables: permite conocer el valor de alguna o varias variables.

Dependiendo de la herramienta usada para compilar el programa, si es de consola o de terminal, su uso y las funciones disponibles variarán.

En las IDE (Entornos de Desarrollo Interactivo), suelen existir herramientas de depuración integradas de manera gráfica. Es muy común que existan dos modos de desarrollar un programa y producir el archivo ejecutable que son "Debug" y "Release". El primer modo se recomienda exclusivamente durante el desarrollo del

programa para poder depurarlo continuamente durante cualquier prueba de ejecución. El segundo modo se establece cuando el programa ha sido terminado y totalmente probado.

## Depuración de programas escritos en C con GCC y GDB

Para depurar un programa usando las herramientas desarrolladas por GNU, éste debe compilarse con información para depuración por medio del compilador GCC.

Para compilar, por ejemplo, un programa llamado *calculadora.c* con GCC con información de depuración, debe realizarse en una terminal con el siguiente comando:

```
gcc -g -o calculadora calculadora.c
```

El parámetro `-g` es quien indica que el ejecutable debe producirse con información de depuración.

Una vez hecho el paso anterior, debe usarse la herramienta GDB, la cual, es el depurador para cualquier programa ejecutable realizado por GCC.

Para depurar un ejecutable debe invocarse a GDB en la terminal indicando cuál es el programa ejecutable a depurar, por ejemplo, para depurar *calculadora*:

```
gdb ./calculadora
```

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*.

## Depuración de programas escritos en C con Dev-C++ 5.0.3.4

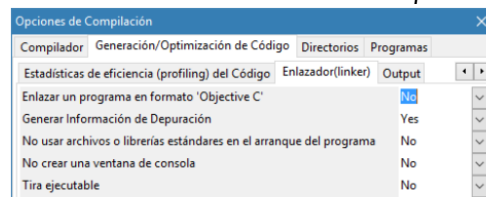
Dev-C++, es una IDE especializada para desarrollar programas escritos en C o C++. Si bien incorpora un editor de textos y un compilador integrados, también posee un depurador. Cabe destacar que por defecto Dev-C++ se basa en el compilador GCC y el depurador en GDB, aunque de manera gráfica ello es transparente para el usuario ya que todo simula una sola herramienta.

Cabe señalar, que si se desea utilizar Dev-C++ como herramienta de desarrollo de programas en C, debe estar instalado adecuadamente en el equipo para que funcione tanto el compilador como las herramientas de depuración. Si se usa sistema operativo Windows,

se recomienda encarecidamente usar la versión que se proporciona en <http://lcp02.fi-b.unam.mx> en la sección de Servicios.

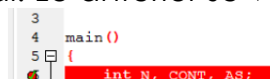
Antes de iniciar la depuración debe tenerse a la mano el archivo con el programa escrito en C o proceder a escribirlo en la misma IDE. Para ello debe usarse el menú *Archivo* → *Nuevo* → *Código Fuente* si se piensa usar la IDE para escribirlo o en su lugar *Archivo* → *Abrir Proyecto* o *Archivo* si ya existía el código fuente.

Una vez que se tiene el programa, debe activarse la opción de compilación generando información para el depurador. Para activar esta opción debe abrirse el menú *Herramientas* → *Opciones del Compilador* y acceder a la pestaña *Generación/Optimización de Código* y finalmente, en la subpestaña *Enlazador* (linker), activar la opción *Generar Información de Depuración*:



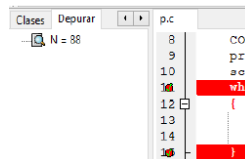
Después de realizar lo anterior, el programa realizado puede compilarse y ejecutarse con lo que ofrece el menú *Ejecutar*.

Para agregar puntos de ruptura, debe hacerse clic en la línea de código donde se desea colocar y ésta se volverá en color rojo. Para retirarlo se hace clic de nuevo en la línea y volverá a su color normal. Lo anterior se ve en la imagen siguiente:



The screenshot shows the Visual Studio Code interface. The top panel displays the source code of a C program. The second line, `printf("TECLAS UN NUMERO IMPAR: ");`, is highlighted in red, indicating the current execution point. The bottom panel shows the 'Run and Debug' window. The 'Breakpoints' tab is active, showing a breakpoint at line 6. The 'Console' tab shows the output of the program: `TECLAS UN NUMERO IMPAR:`  followed by a newline character.

Finalmente, para estudiar el valor de cada variable, se puede recurrir a la función *Añadir Watch* y escribir el nombre de la variable. En un cuadro a la izquierda, se verá el nombre de la variable y su valor hasta el punto donde se está ejecutando el programa:

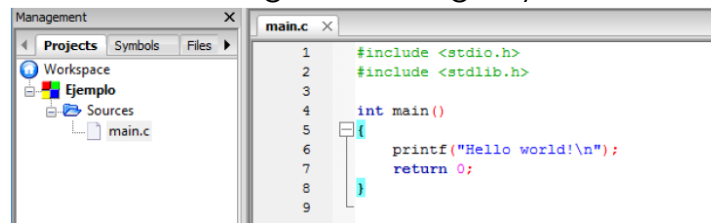


## Depuración de programas escritos en C con Code::Blocks 13.12

Para poder depurar, es necesario crear un nuevo proyecto desde el menú *File* → *New* → *Project* y elegir en el cuadro que aparece *Console Application* (recordar que, por ahora, todos los programas son desarrollados en modo de consola). Seguir el asistente para crear el proyecto eligiendo que se usará el lenguaje C, luego seleccionar un título adecuado para el proyecto, la ruta donde se creará y el título del archivo asociado al proyecto. Posteriormente, en el mismo asistente seleccionar *Create “Debug” Configuration* y *Create “Release” Configuration* en el mismo asistente con compilador *GNU GCC Compiler*. Nótese que existen dos carpetas asociadas que son

`/bin/debug` y `/bin/release` que son donde se crearán los ejecutables de depuración y el final respectivamente. Se debe finalizar el asistente.

En la parte izquierda, se encontrará el nombre del archivo fuente del proyecto. Para ello navegar como se indica en la siguiente imagen y dar clic en `main.c`:

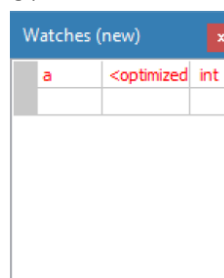


Dicho archivo debe editarse para formar el programa deseado. Cuando se está desarrollando es importante que esté seleccionado el modo de depuración, localizado en la barra de herramientas que se muestra:

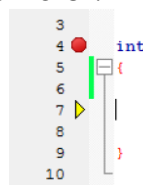


Como puede observarse, hay un menú desplegable que tiene la opción *Debug* y *Release*, debe estar siempre seleccionada la primera opción hasta no haber terminado el programa a desarrollar. Cuando todo esté listo, se cambiará a la segunda opción y se usará el archivo ejecutable que se localiza en la `/bin/release`, jamás el localizado en `/bin/debug` que solo tiene efectos de desarrollo.

A la izquierda del menú, se encuentran las opciones de compilación y ejecución del programa en el modo seleccionado. A la derecha se encuentran las opciones de depuración. La primera opción, *Debug/Continue*, permite ejecutar el programa en modo de depuración y reanudar la ejecución después de un punto de ruptura. La opción *Stop debugger*, detiene la depuración y permite continuar editando. Existen otras herramientas adicionales, entre ellas correr el programa hasta donde se encuentre el cursor en el texto, ejecutar la siguiente línea o la siguiente instrucción, todas ellas estudiadas anteriormente.



Para visualizar una variable, debe hacerse clic sobre ella con el depurador corriendo, y dar clic en *Watch 'variable'*, aparecerá un pequeño cuadro con la tabla de las variables que se desean visualizar y su valor:



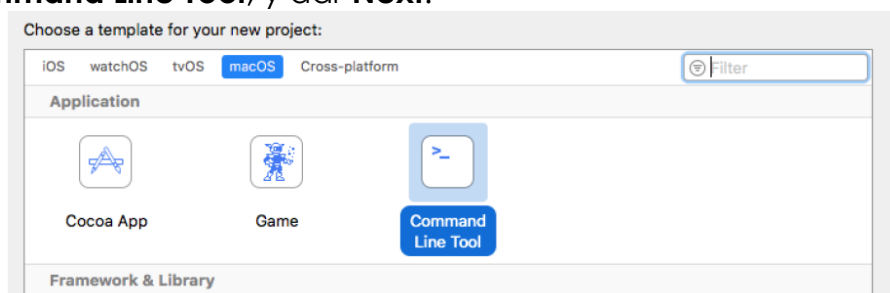
Finalmente, para agregar un punto de ruptura, se tiene que hacer clic derecho sobre el número de línea de código y agregarlo, aparecerá un punto rojo en la línea, para quitarlo se tiene que hacer el mismo procedimiento.

La parte del código que se está ejecutando por el depurador, se indica por medio de una flecha amarilla en el número de línea correspondiente.

## Depuración de programas escritos en C con Xcode de Mac

Xcode es otro Entorno de Desarrollo Interactivo (IDE) que, al igual que los antes mencionados en esta guía, además de contener herramientas de depuración, cuenta con un editor y un compilador entre otros. Es por esto que para aplicar las actividades de depuración de un programa en C en equipos MAC utilizando esta herramienta, debemos realizar lo siguiente.

Abrir la aplicación Xcode, crear un nuevo proyecto seleccionando **Create a new Xcode project**; a lo cual aparecerá la siguiente pantalla donde se deberá seleccionar **macOS** y **Command Line Tool**, y dar **Next**.



Posteriormente dar las características del proyecto, seguido de **Next**. Por ejemplo:

Product Name:

Team:

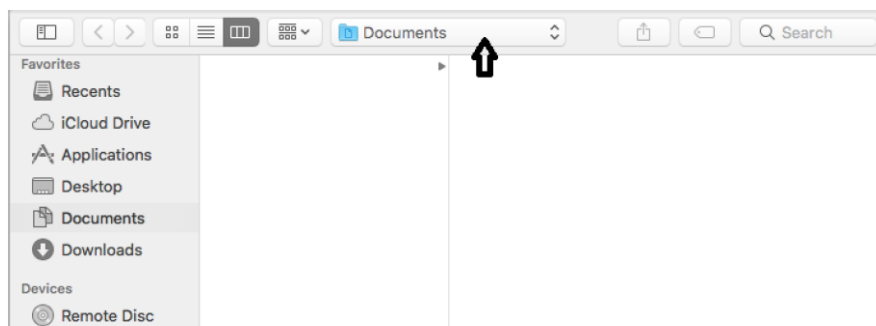
Organization Name:

Organization Identifier:

Bundle Identifier:

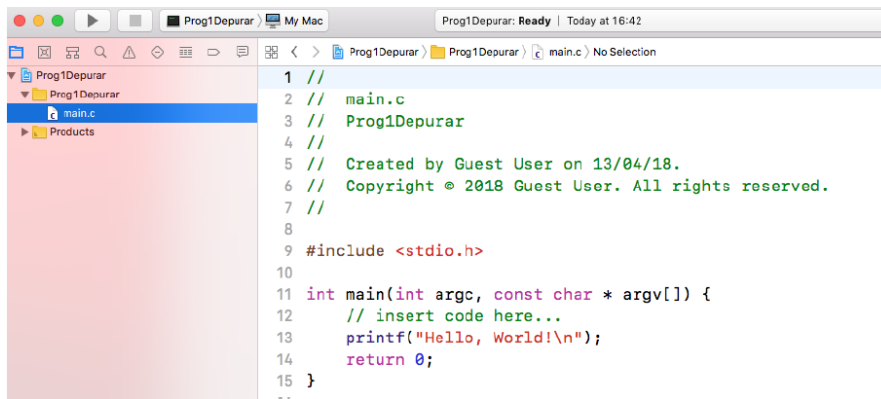
Language:

Se presentará una ventana donde hay que indicar dónde se grabará el proyecto. Por ejemplo en Documents. Después dar **Create**.



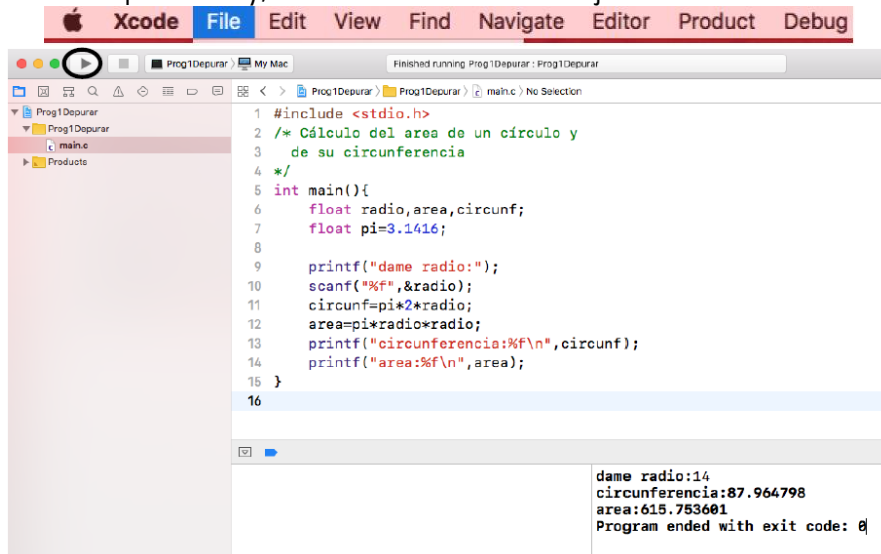
A partir de este momento ya tenemos el área de trabajo para editar y depurar programas:



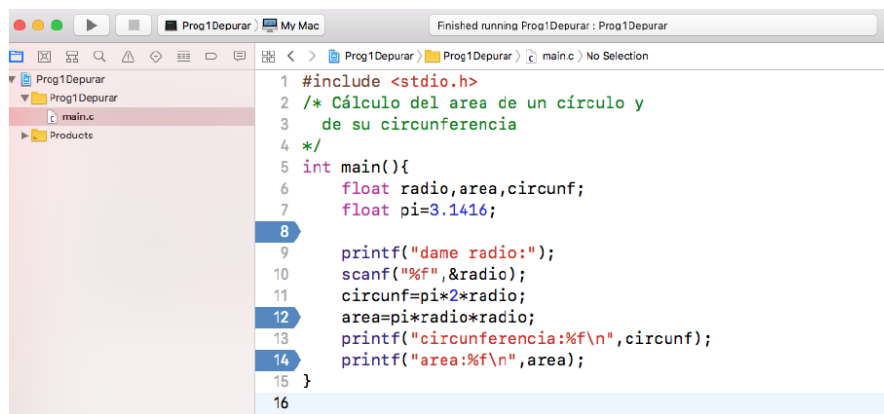


Seleccionando main.c en la jerarquía de archivos, mostrado en el lado izquierdo, podremos observar que nos presenta un “esqueleto” de un programa en C, el cual se deberá sustituir por el que se vaya a depurar. Como Xcode es un IDE, podemos aquí mismo editar y compilar antes de iniciar las actividades de depuración. Para ello nos podemos auxiliar de la barra superior de menús de Xcode:

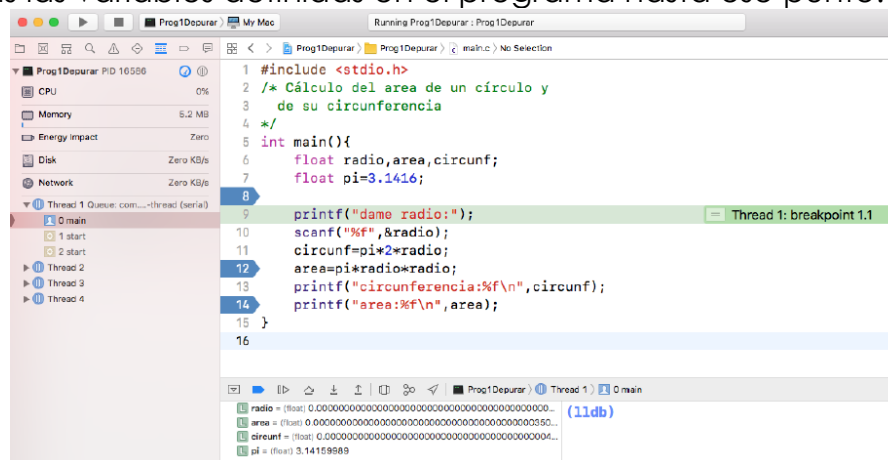
Una vez que se tiene el programa a depurar editado, primeramente se recomienda compilarlo antes de realizar su depuración; esto se realiza usando las teclas Cmd +B. Posteriormente para ejecutarlo (o compilarlo y ejecutarlo si no se ha hecho anteriormente la compilación), en la barra de trabajo utilizar



Para iniciar con algunas actividades de depuración, vamos a indicar puntos de ruptura. Esto se realiza haciendo clic, con el botón derecho del mouse, sobre el número que indica la línea de código donde se desea detener y aparecerá una flecha azul:

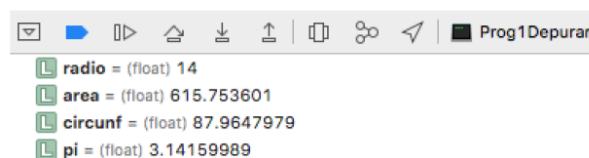


Una vez definidos los puntos de ruptura, al momento de ejecutar el programa se detendrá al encontrar el primer punto de ruptura, indicando automáticamente el valor de todas las variables definidas en el programa hasta ese punto. Por ejemplo:



Como se puede observar, en la parte inferior se muestran las variables con su valor y tipo de dato declarado. Si a una variable aún no se le asigna un valor, éste será indeterminado.

La continuidad y control de la ejecución del programa, se realiza a través de los íconos de la barra de resultados:



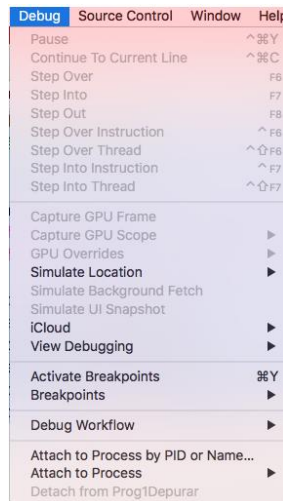
Desactiva todos los puntos de ruptura

Continúa la ejecución del programa hasta la siguiente línea de ruptura

Ejecuta sólo la siguiente instrucción y se detiene

Sigue ejecutando el programa hasta la siguiente línea de ruptura

Se recomienda revisar las opciones que se tienen en el menú Debug para realizar otras actividades de depuración.



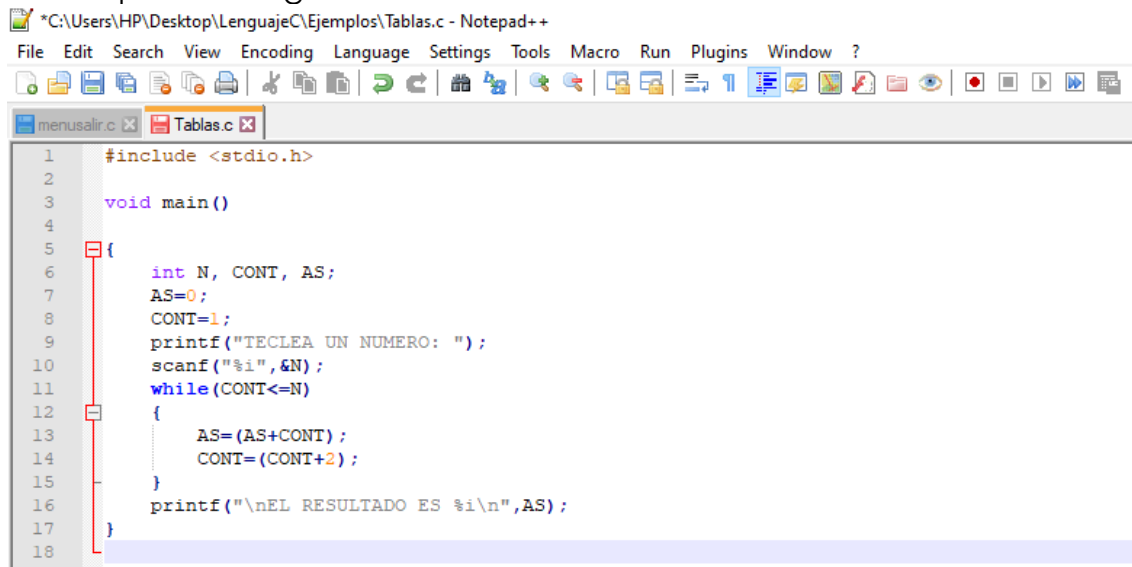
## Ejercicios propuestos

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);
}
```

Primero se copia el código en un editor de texto



Se ejecuta procurando que no tenga errores

```
Símbolo del sistema

C:\Users\HP\Desktop\LenguajeC\Ejemplos>Tablas.exe
TECLEA UN NUMERO: 2

EL RESULTADO ES 1

C:\Users\HP\Desktop\LenguajeC\Ejemplos>gcc Tablas.c -o Tablas.exe

C:\Users\HP\Desktop\LenguajeC\Ejemplos>Tablas.exe
TECLEA UN NUMERO: 3

EL RESULTADO ES 4

C:\Users\HP\Desktop\LenguajeC\Ejemplos>gcc Tablas.c -o Tablas.exe

C:\Users\HP\Desktop\LenguajeC\Ejemplos>Tablas.exe
TECLEA UN NUMERO: 4

EL RESULTADO ES 4

C:\Users\HP\Desktop\LenguajeC\Ejemplos>gcc Tablas.c -o Tablas.exe

C:\Users\HP\Desktop\LenguajeC\Ejemplos>Tablas.exe
TECLEA UN NUMERO: 6

EL RESULTADO ES 9

C:\Users\HP\Desktop\LenguajeC\Ejemplos>
```

Ahora procedo a depurar el programa con GCC y GDB

```
Símbolo del sistema - gdb ./Tablas

C:\Users\HP\Desktop\LenguajeC\Ejemplos>gcc -g -o Tablas Tablas.c

C:\Users\HP\Desktop\LenguajeC\Ejemplos>gdb ./Tablas
GNU gdb (GDB) 7.6.1
Copyright (C) 2013 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 "mingw32".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from C:\Users\HP\Desktop\LenguajeC\Ejemplos\Tablas.exe...done.
(gdb) list 6,10
   6             int N, CONT, AS;
   7             AS=0;
   8             CONT=1;
   9             printf("TECLEA UN NUMERO: ");
  10             scanf("%i",&N);
(gdb) b 8
Breakpoint 1 at 0x401426: file Tablas.c, line 8.
(gdb) d 8
No breakpoint number 8.
(gdb)
```

```
Símbolo del sistema

<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from C:\Users\HP\Desktop\LenguajeC\Ejemplos\Tablas.exe...done.
(gdb) list 6,10
   6             int N, CONT, AS;
   7             AS=0;
   8             CONT=1;
   9             printf("TECLEA UN NUMERO: ");
  10             scanf("%i",&N);
(gdb) b 8
Breakpoint 1 at 0x401426: file Tablas.c, line 8.
(gdb) d 8
No breakpoint number 8.
(gdb) clear
No source file specified.
(gdb) info line 12
Line 12 of "Tablas.c" is at address 0x401450 <main+64> but contains no code.
(gdb) info line 13
Line 13 of "Tablas.c" starts at address 0x401450 <main+64> and ends at 0x401458 <main+72>.
(gdb) p suma acumulada
No symbol "suma" in current context.
(gdb) q

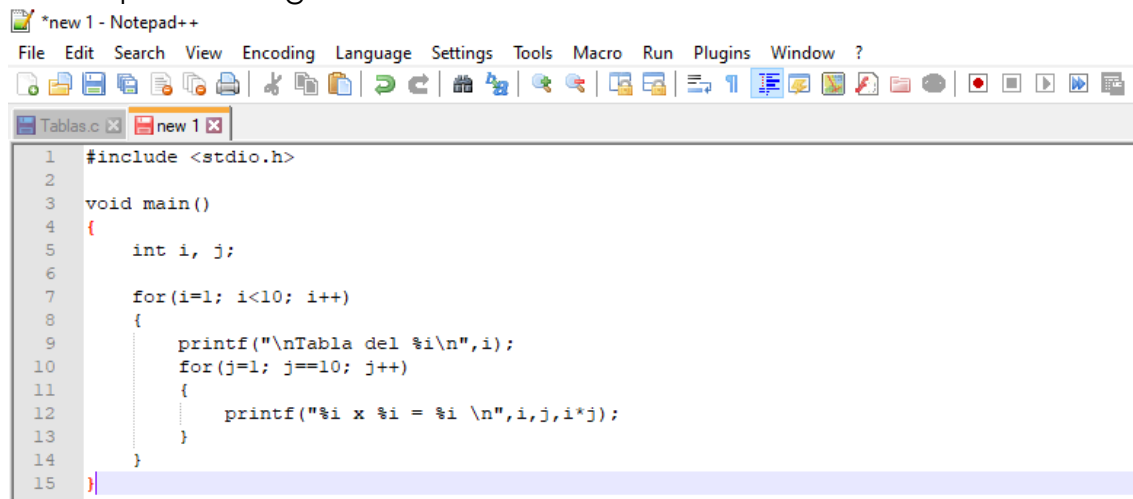
C:\Users\HP\Desktop\LenguajeC\Ejemplos>
```

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("\nTabla del %i\n", i);
        for(j=1; j==10; j++)
        {
            printf("%i X %i = %i\n", i, j, i*j);
        }
    }
}
```

Primero se copia el código en un editor de texto

A screenshot of the Notepad++ text editor. The title bar says '\*new 1 - Notepad++'. The menu bar includes File, Edit, Search, View, Encoding, Language, Settings, Tools, Macro, Run, Plugins, Window, and ?. The toolbar contains various icons for file operations and editing. The code is pasted into a window titled 'Tablas.c'. The code is as follows:

```
1 #include <stdio.h>
2
3 void main()
4 {
5     int i, j;
6
7     for(i=1; i<10; i++)
8     {
9         printf("\nTabla del %i\n",i);
10        for(j=1; j==10; j++)
11        {
12            printf("%i x %i = %i \n",i,j,i*j);
13        }
14    }
15 }
```

Se ejecuta y se ven los errores. El programa no nos muestra errores y se ejecuta correctamente, sin embargo, no se muestran los resultados esperados.

A screenshot of a Windows command prompt window titled 'Símbolo del sistema'. The command prompt shows the following sequence of commands and output:

```
C:\Users\HP\Desktop\LenguajeC\Ejemplos>gcc Tablas.c -o Tablas.exe
C:\Users\HP\Desktop\LenguajeC\Ejemplos>Tablas.exe
Tabla del 1
Tabla del 2
Tabla del 3
Tabla del 4
Tabla del 5
Tabla del 6
Tabla del 7
Tabla del 8
Tabla del 9
C:\Users\HP\Desktop\LenguajeC\Ejemplos>
```

El programa no es capaz de mostrarnos las tablas de multiplicar desde la del 1 hasta la del 10, es más, solo es capaz de mostrar los títulos de estas.

```
Símbolo del sistema
C:\Users\HP\Desktop\LenguajeC\Ejemplos>gcc Tablas.c -o Tablas.exe
C:\Users\HP\Desktop\LenguajeC\Ejemplos>Tablas.exe

Tabla del 1
Tabla del 2
Tabla del 3
Tabla del 4
Tabla del 5
Tabla del 6
Tabla del 7
Tabla del 8
Tabla del 9

C:\Users\HP\Desktop\LenguajeC\Ejemplos>
```

Ahora procedo a depurar el programa con GCC y GDB

```
Símbolo del sistema - gdb ./Tablas
C:\Users\HP\Desktop\LenguajeC\Ejemplos>gdb ./Tablas
GNU gdb (GDB) 7.6.1
Copyright (C) 2013 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 "mingw32".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from C:\Users\HP\Desktop\LenguajeC\Ejemplos\Tablas.exe...done.
(gdb) list 1,15
1      #include <stdio.h>
2
3      void main()
4      {
5          int i, j;
6
7          for(i=1; i<10; i++)
8          {
9              printf("\nTabla del %i\n",i);
10             for(j=1; j<=10; j++)
11             {
12                 printf("%i x %i = %i \n",i,j,i*j);
13             }
14         }
15     }(gdb)

(gdb) break point 7
Function "point 7" not defined.
Make breakpoint pending on future shared library load? (y or [n]) n
(gdb) b 7
Undefined command: "b7". Try "help".
(gdb) b 7
Breakpoint 1 at 0x40141e: file Tablas.c, line 7.
(gdb) b 10
Breakpoint 2 at 0x40143c: file Tablas.c, line 10.
(gdb) b 13
Breakpoint 3 at 0x401487: file Tablas.c, line 13.
(gdb) r
Starting program: C:\Users\HP\Desktop\LenguajeC\Ejemplos\./Tablas.exe
[New Thread 8140.0x21b0]
[New Thread 8140.0x1990]

Breakpoint 1, main () at Tablas.c:7
7          for(i=1; i<10; i++)
(gdb) c
Continuing.

Tabla del 1

Breakpoint 2, main () at Tablas.c:10
10         for(j=1; j<=10; j++)
(gdb) c
Continuing.
```

```
Simbolo del sistema
$6 = 1
(gdb) p i
$7 = 2
(gdb) p i
$8 = 2
(gdb) p i
$9 = 2
(gdb) p j
$10 = 1
(gdb) p j
$11 = 1
(gdb) q
A debugging session is active.

Inferior 1 [process 8140] will be killed.

Quit anyway? (y or n) n
Not confirmed.
(gdb) q
A debugging session is active.

Inferior 1 [process 8140] will be killed.

Quit anyway? (y or n) y
error return ../../gdb-7.6.1/gdb/windows-nat.c:1275 was 5
C:\Users\HP\Desktop\LenguajeC\Ejemplos>
```

Ahora que supongo donde está el error lo corrijo en el código fuente.

```
*C:\Users\HP\Desktop\LenguajeC\Ejemplos\Tablas.c - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?

Tablas.c new 1
1 #include <stdio.h>
2
3 void main()
4 {
5     int i, j;
6
7     for(i=1; i<=10; i++)
8     {
9         printf("\nTabla del %i\n",i);
10        for(j=1; j<=10; j++)
11        {
12            printf("%i x %i = %i \n",i,j,i*j);
13        }
14    }
15 }
16
```

El error es: en la línea 7 en vez de  $i < 10$  es  $i \leq 10$   
Y en la línea 10 en vez de  $j = 10$  es  $j \leq 10$

Con estas correcciones ejecutamos nuevamente el programa

```
Simbolo del sistema
C:\Users\HP\Desktop\LenguajeC\Ejemplos>gcc Tablas.c -o Tablas.exe
C:\Users\HP\Desktop\LenguajeC\Ejemplos>Tablas.exe

Tabla del 1
1 x 1 = 1
1 x 2 = 2
1 x 3 = 3
1 x 4 = 4
1 x 5 = 5
1 x 6 = 6
1 x 7 = 7
1 x 8 = 8
1 x 9 = 9
1 x 10 = 10

Tabla del 2
2 x 1 = 2
2 x 2 = 4
2 x 3 = 6
2 x 4 = 8
2 x 5 = 10
2 x 6 = 12
2 x 7 = 14
2 x 8 = 16
2 x 9 = 18
2 x 10 = 20

Tabla del 3
3 x 1 = 3
3 x 2 = 6
3 x 3 = 9
3 x 4 = 12
3 x 5 = 15
3 x 6 = 18
3 x 7 = 21
3 x 8 = 24
3 x 9 = 27
3 x 10 = 30

Tabla del 4
4 x 1 = 4
4 x 2 = 8
4 x 3 = 12
4 x 4 = 16
4 x 5 = 20
4 x 6 = 24
4 x 7 = 28
4 x 8 = 32
4 x 9 = 36
4 x 10 = 40

Tabla del 5
5 x 1 = 5
5 x 2 = 10
5 x 3 = 15
5 x 4 = 20
5 x 5 = 25
5 x 6 = 30
5 x 7 = 35
5 x 8 = 40
5 x 9 = 45
5 x 10 = 50

Tabla del 6
6 x 1 = 6
6 x 2 = 12
6 x 3 = 18
6 x 4 = 24
6 x 5 = 30
6 x 6 = 36
6 x 7 = 42
6 x 8 = 48
6 x 9 = 54
6 x 10 = 60

Tabla del 7
7 x 1 = 7
7 x 2 = 14
7 x 3 = 21
7 x 4 = 28
7 x 5 = 35
7 x 6 = 42
7 x 7 = 49
7 x 8 = 56
7 x 9 = 63
7 x 10 = 70

Tabla del 8
8 x 1 = 8
8 x 2 = 16
8 x 3 = 24
8 x 4 = 32
8 x 5 = 40
8 x 6 = 48
8 x 7 = 56
8 x 8 = 64
8 x 9 = 72
8 x 10 = 80

Tabla del 9
9 x 1 = 9
9 x 2 = 18
9 x 3 = 27
9 x 4 = 36
9 x 5 = 45
9 x 6 = 54
9 x 7 = 63
9 x 8 = 72
9 x 9 = 81
9 x 10 = 90

Tabla del 10
10 x 1 = 10
10 x 2 = 20
10 x 3 = 30
10 x 4 = 40
10 x 5 = 50
10 x 6 = 60
10 x 7 = 70
10 x 8 = 80
10 x 9 = 90
10 x 10 = 100

C:\Users\HP\Desktop\LenguajeC\Ejemplos>
```

El siguiente programa muestra una *violación de segmento* durante su ejecución y se interrumpe; usar un depurador para detectar y corregir la falla.

```
#include <stdio.h>
#include <math.h>
void main()
{
    int K, X, AP, N;
    float AS;
    printf("EL TERMINO GENERICO DE LA SERIE ES: X^K/K!");
    printf("\nN=");
    scanf("%d",N);
    printf("X=");
    scanf("%d",X);
    K=0;
    AP=1;
    AS=0;
    while(K<=N)
    {
        AS=AS+pow(X,K)/AP;
        K=K+1;
        AP=AP*K;
    }
    printf("SUM=%le",AS);
}
```

Primero se copia el código en un editor de texto

Se ejecuta y se ven los errores. El programa no nos muestra errores y se ejecuta correctamente, sin embargo, no se muestran los resultados esperados, el programa solo pide una variable y después de detiene.



```
Símbolo del sistema

C:\Users\HP\Desktop\LenguajeC\Ejemplos>gcc violacionDeSegmento.c -o violacionDeSegmento.exe

C:\Users\HP\Desktop\LenguajeC\Ejemplos>violacionDeSegmento.exe
EL TERMINO GENERICO DE LA SERIE ES: X^K/K!
N=5

C:\Users\HP\Desktop\LenguajeC\Ejemplos>
```

Ahora procedo a depurar el programa con GCC y GDB para encontrar el origen de la posible falla.

```
Símbolo del sistema - gdb ./violacionDeSegmento

C:\Users\HP\Desktop\LenguajeC\Ejemplos>gcc violacionDeSegmento.c -o violacionDeSegmento.exe

C:\Users\HP\Desktop\LenguajeC\Ejemplos>violacionDeSegmento.exe
EL TERMINO GENERICO DE LA SERIE ES: X^K/K!
N=5

C:\Users\HP\Desktop\LenguajeC\Ejemplos>gcc -g -o violacionDeSegmento violacionDeSegmento.c

C:\Users\HP\Desktop\LenguajeC\Ejemplos>gdb ./violacionDeSegmento
GNU gdb (GDB) 7.6.1
Copyright (C) 2013 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 "mingw32".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from C:\Users\HP\Desktop\LenguajeC\Ejemplos\violacionDeSegmento.exe...done.
(gdb)

(gdb) list 1,22
1      #include <stdio.h>
2      #include <math.h>
3      void main()
4      {
5          int K, X, AP, N;
6          float AS;
7          printf("EL TERMINO GENERICO DE LA SERIE ES: X^K/K!");
8          printf("\nN=");
9          scanf("%d",N);
10         printf("X=");
11         scanf("%d",X);
12         K=0;
13         AP=1;
14         AS=0;
15         while(K<=N)
16         {
17             AS=AS+pow(X,K)/AP;
18             K=K+1;
19             AP=AP*K;
20         }
21         printf("USM=%le",AS);
22     }(gdb) run
Starting program: C:\Users\HP\Desktop\LenguajeC\Ejemplos\./violacionDeSegmento.exe
[New Thread 10144.0x1490]
[New Thread 10144.0x2678]
EL TERMINO GENERICO DE LA SERIE ES: X^K/K!
N=5

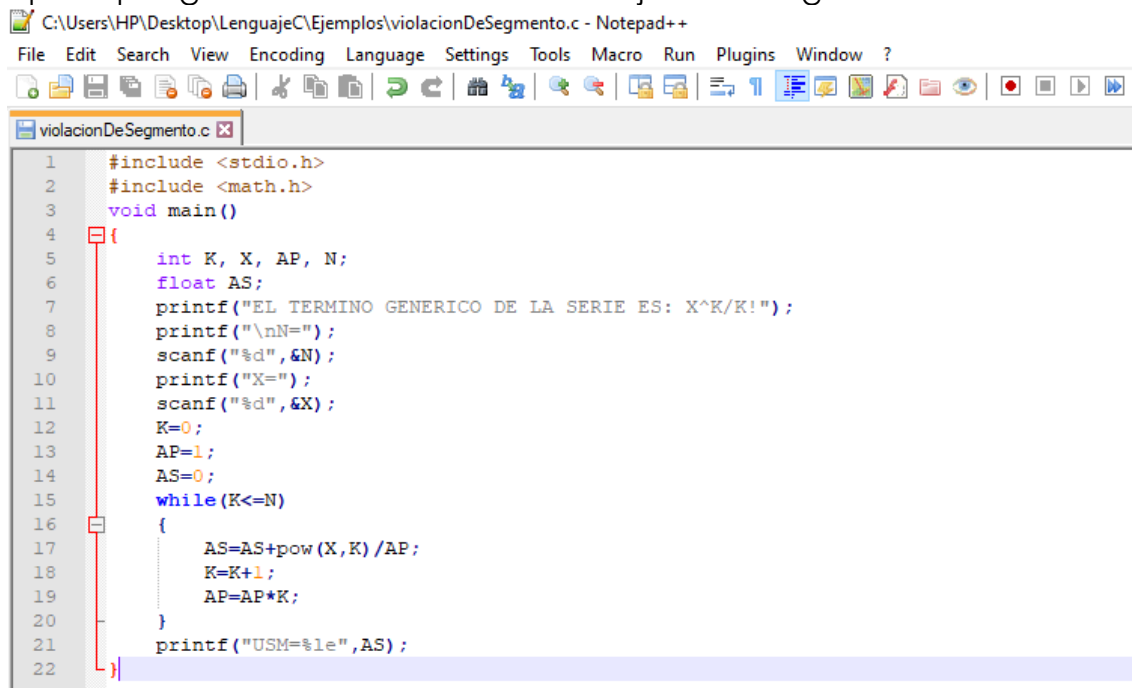
Program received signal SIGSEGV, Segmentation fault.
0x7720040c in ungetwc () from C:\Windows\SysWOW64\msvcrt.dll
```

```
Símbolo del sistema
10 printf("X=");
11 scanf("%d",X);
12 K=0;
13 AP=1;
14 AS=0;
15 while(K<=N)
16 {
17     AS=AS+pow(X,K)/AP;
18     K=K+1;
19     AP=AP*K;
20 }
21 printf("USM=%le",AS);
22 }(gdb) run
Starting program: C:\Users\HP\Desktop\LenguajeC\Ejemplos\./violacionDeSegmento.exe
[New Thread 10144.0xd490]
[New Thread 10144.0x2678]
EL TERMINO GENERICO DE LA SERIE ES: X^K/K!
N=5
Program received signal SIGSEGV, Segmentation fault.
0x7720040c in ungetwc () from C:\Windows\System32\user32.dll
(gdb) b 9
Breakpoint 1 at 0x401436: file violacionDeSegmento.c, line 9.
(gdb) q
A debugging session is active.

    Inferior 1 [process 10144] will be killed.

Quit anyway? (y or n) y
error return ../../gdb-7.6.1/gdb/windows-nat.c:1275 was 5
```

Ahora que supongo donde está el error lo corrijo en el código fuente.



```
C:\Users\HP\Desktop\LenguajeC\Ejemplos\violacionDeSegmento.c - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
violacionDeSegmento.c
1 #include <stdio.h>
2 #include <math.h>
3 void main()
4 {
5     int K, X, AP, N;
6     float AS;
7     printf("EL TERMINO GENERICO DE LA SERIE ES: X^K/K!");
8     printf("\nN=");
9     scanf("%d",N);
10    printf("X=");
11    scanf("%d",X);
12    K=0;
13    AP=1;
14    AS=0;
15    while(K<=N)
16    {
17        AS=AS+pow(X,K)/AP;
18        K=K+1;
19        AP=AP*K;
20    }
21    printf("USM=%le",AS);
22 }
```

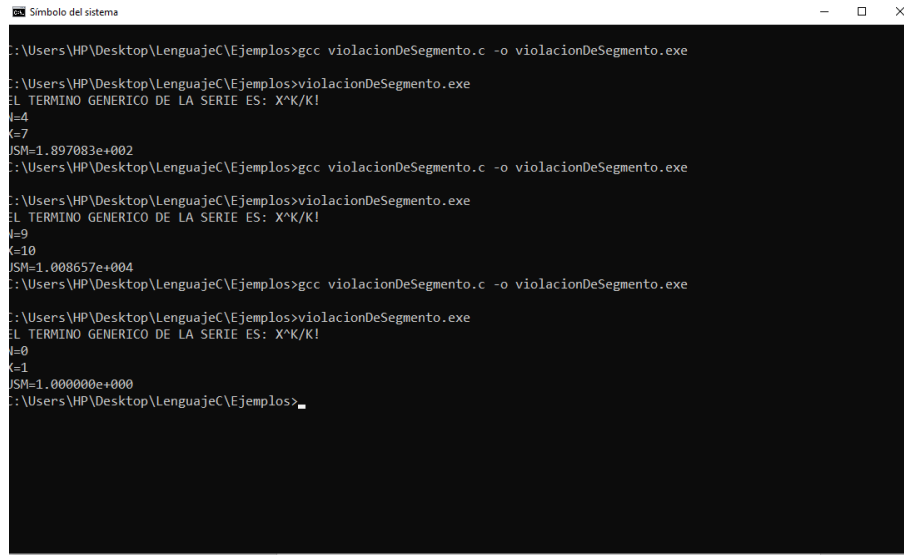
El error es que en la línea 9 y 11 cuando ingresamos el primer valor en este caso de N no lo almacena como variable ya que falta el símbolo & , es decir que en vez de:

scanf("%d",N); → scanf("%d",&N);

Lo mismo ocurre con la variable X:

scanf("%d",X); → scanf("%d",&X);

Con estas correcciones ejecutamos nuevamente el programa



```
Símbolo del sistema

C:\Users\HP\Desktop\LenguajeC\Ejemplos>gcc violacionDeSegmento.c -o violacionDeSegmento.exe

C:\Users\HP\Desktop\LenguajeC\Ejemplos>violacionDeSegmento.exe
EL TERMINO GENERICO DE LA SERIE ES: X*K/K!
N=4
K=7
JSM=1.897083e+002

C:\Users\HP\Desktop\LenguajeC\Ejemplos>gcc violacionDeSegmento.c -o violacionDeSegmento.exe

C:\Users\HP\Desktop\LenguajeC\Ejemplos>violacionDeSegmento.exe
EL TERMINO GENERICO DE LA SERIE ES: X*K/K!
N=9
K=10
JSM=1.008657e+004

C:\Users\HP\Desktop\LenguajeC\Ejemplos>gcc violacionDeSegmento.c -o violacionDeSegmento.exe

C:\Users\HP\Desktop\LenguajeC\Ejemplos>violacionDeSegmento.exe
EL TERMINO GENERICO DE LA SERIE ES: X*K/K!
N=0
K=1
JSM=1.000000e+000

C:\Users\HP\Desktop\LenguajeC\Ejemplos>
```

## Conclusiones

Cuando no hay errores en un programa este puede ejecutarse, sin embargo, si el programa no nos muestra los resultados esperados o se interrumpen, quiere decir que aún hay fallas que deben ser corregidas. Ahora bien, el depurar un programa no solo nos ayuda a encontrar estas fallas más fácilmente colocando puntos de ruptura o listando las partes del programa. Pero no nos señala donde estamos mal, solo nos ayuda a investigar en donde posiblemente lo estemos, ayudándonos a analizar el programa de una manera más sencilla.

## Bibliografía

- Gutiérrez Rodríguez, Javier Jesús. *Primeros pasos con GDB*. Consulta: octubre de 2016. Disponible en: [http://www.lsi.us.es/~javierj/ssoo\\_ficheros/GuiaGDB.htm](http://www.lsi.us.es/~javierj/ssoo_ficheros/GuiaGDB.htm)
- Ferreira, Amelia. *Depurador gdb*. Consulta: octubre de 2016. Disponible en: <http://learnassembler.com/gdbesp.html>
- Ferreira, Amelia. *Depurador gdb - uso de la opción -g de gcc*. Consulta: octubre de 2016. Disponible en: <http://learnassembler.com/opc.html>
- Gutiérrez, Erik Marín. *Depuración de programas Dev C++*. Consulta: octubre de 2016. Disponible en: <http://programacionymetodos.blogspot.mx/2012/05/depuracion-de-programas-dev-c.html>
- González Cárdenas, Miguel Eduardo; Marín Lara, Claudia Lorena; Noguerón Pérez, Pedro. *Apuntes De Computadoras Y Programación*. Universidad Nacional Autónoma de México.
- Pozo Coronado, Salvador. *Primeros pasos con GDB*. Consulta: octubre de 2016. Disponible en: <http://www.c.conclase.net/devcpp/?cap=depurar>