



Carátula para entrega de prácticas

Facultad de Ingeniería

Laboratorio de docencia

Laboratorios de computación salas A y B

Marco Antonio Martínez Quintana

Profesor:

Fundamentos de programación

Asignatura:

3

Grupo:

10

No de Práctica(s):

Sánchez Hernández Marco Antonio

Integrante(s):

*No. de Equipo de
cómputo empleado:*

No aplica

48

No. de Lista o Brigada:

2021-1

Semestre:

13/diciembre/2020

Fecha de entrega:

Práctica realizada en Kubuntu 20.04.1 LTS

Observaciones:

CALIFICACIÓN: _____

Depuración de programas

Introducción

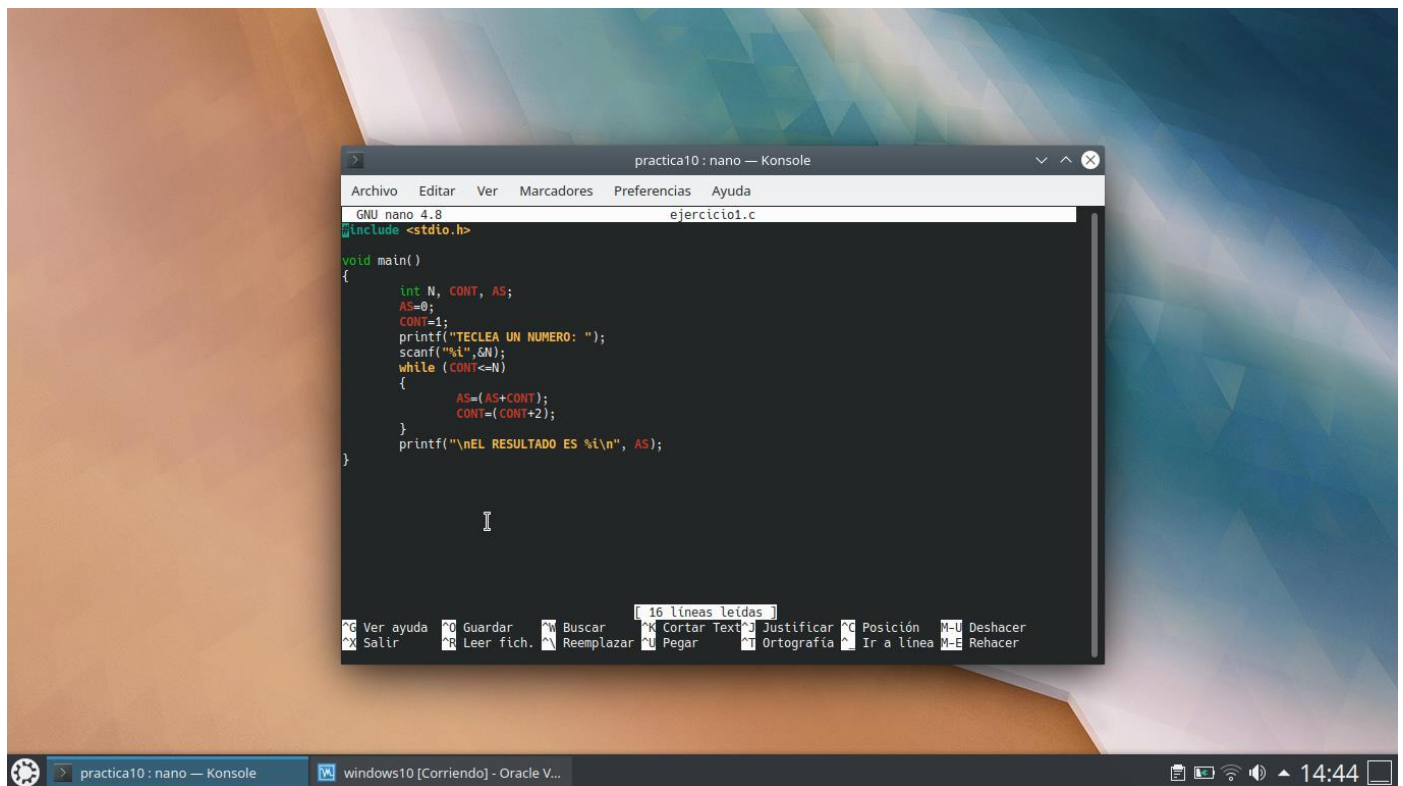
La depuración de un programa es un proceso en el cual se ejecuta el programa bajo un entorno controlado, de tal manera que se puede pausar la ejecución del programa para el análisis del flujo de datos, los valores que toman las variables involucradas en los procesos, etc., es importante que nuestro programa pueda ser compilado antes de realizar una depuración.

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.

Ejercicios

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



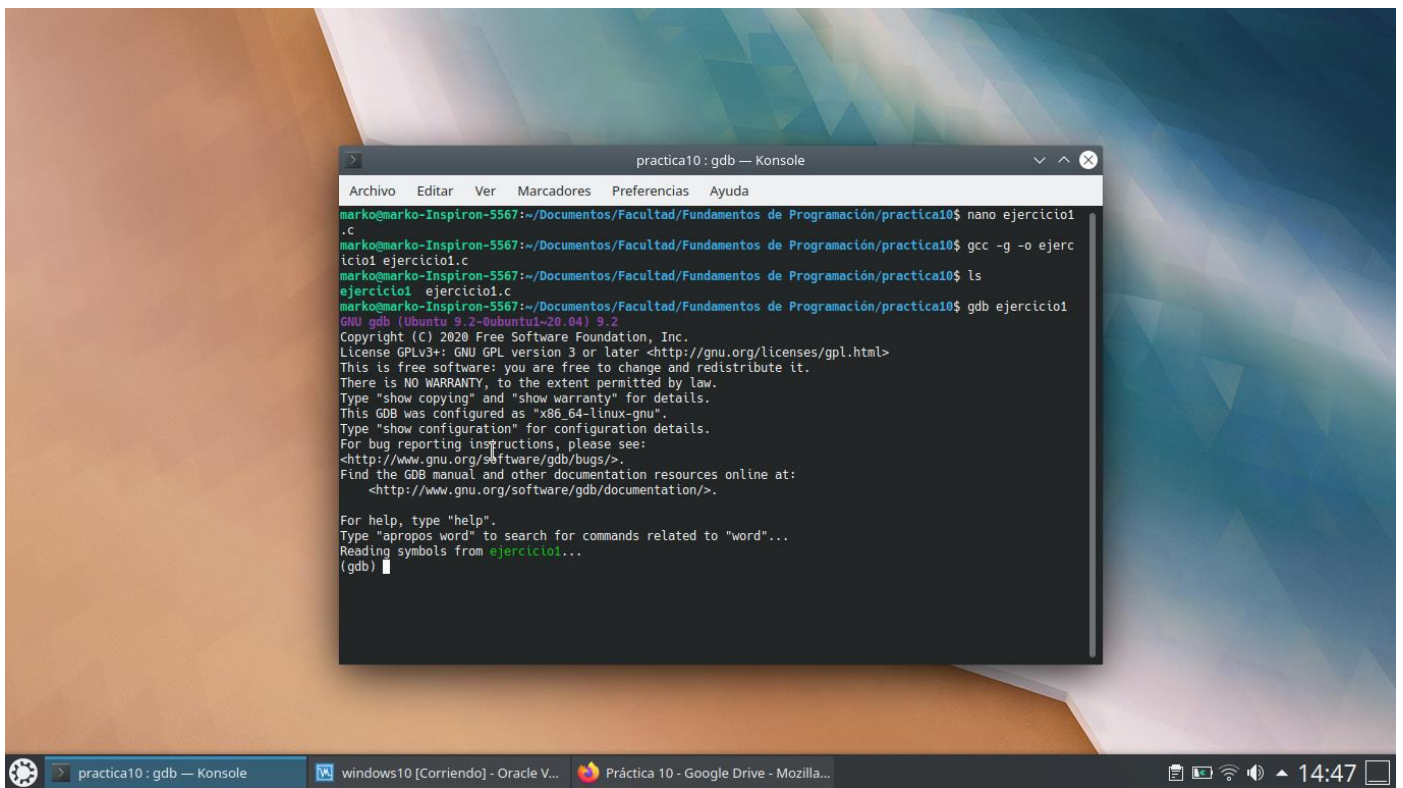
```
practica10: nano — Konsole
GNU nano 4.8 ejercicio1.c
#include <stdio.h>

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);
}
```

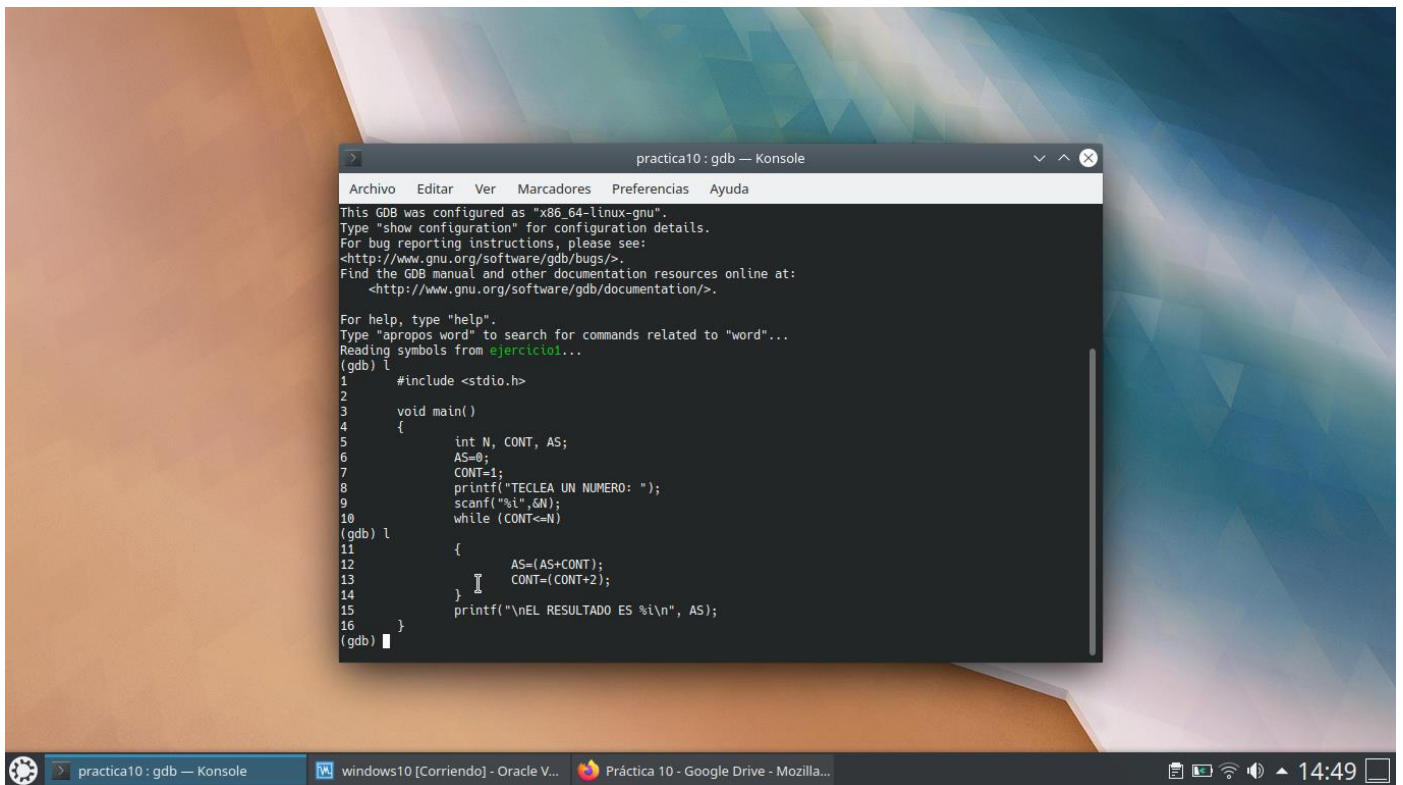
16 líneas leídas

Ver ayuda Guardar Buscar Cortar Text Justificar Posición Deshacer
Salir Leer fich. Reemplazar Pegar Ortografía Ir a línea Rehacer

practica10: nano — Konsole windows10 [Corriendo] - Oracle V... 14:44



Uso del comando *list* o *l*, el cual nos permite mostrar 10 líneas del código fuente del programa, fue utilizado dos veces para mostrar el código completo.



Con ayuda del comando *b*, establecemos un *breakpoint* en la línea número 14 del código para posteriormente analizar los valores contenidos en cada variable.

```
practica10: gdb — Konsole
Archivo  Editar  Ver  Marcadores  Preferencias  Ayuda

For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ejercicio1...
(gdb) l
1      #include <stdio.h>
2
3      void main()
4      {
5          int N, CONT, AS;
6          AS=0;
7          CONT=1;
8          printf("TECLEA UN NUMERO: ");
9          scanf("%i",&N);
10         while (CONT<=N)
(gdb) l
11         {
12             AS=(AS+CONT);
13             CONT=(CONT+2);
14         }
15         printf("\nEL RESULTADO ES %i\n", AS);
16     }
(gdb) b 14
Punto de interrupción 1 at 0x11ef: file ejercicio1.c, line 15.
(gdb)
```

Con el comando *run* o *r* ejecutamos el programa con el valor de prueba 1, para observar los valores que toma cada variable.

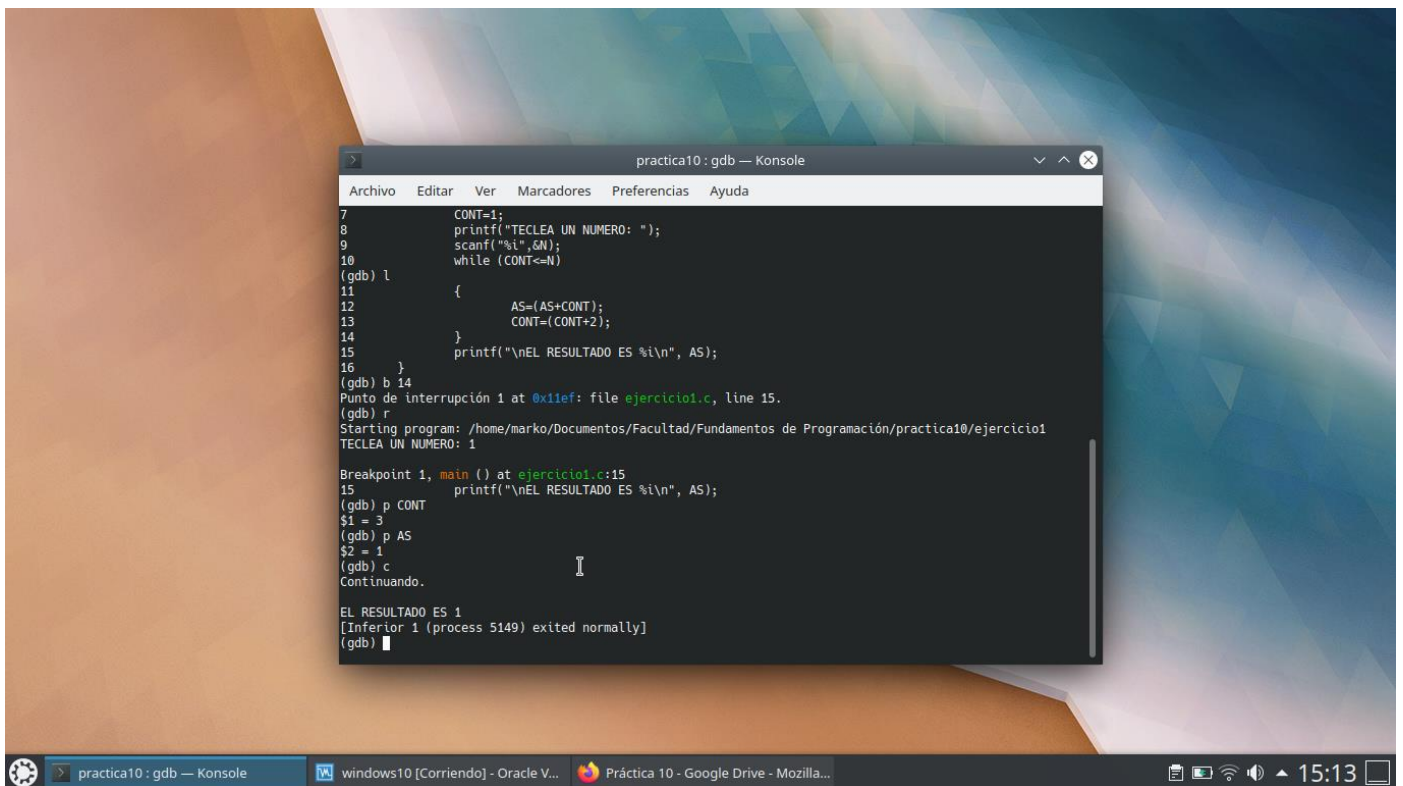
```
practica10: gdb — Konsole
Archivo  Editar  Ver  Marcadores  Preferencias  Ayuda

2
3      void main()
4      {
5          int N, CONT, AS;
6          AS=0;
7          CONT=1;
8          printf("TECLEA UN NUMERO: ");
9          scanf("%i",&N);
10         while (CONT<=N)
(gdb) l
11         {
12             AS=(AS+CONT);
13             CONT=(CONT+2);
14         }
15         printf("\nEL RESULTADO ES %i\n", AS);
16     }
(gdb) b 14
Punto de interrupción 1 at 0x11ef: file ejercicio1.c, line 15.
(gdb) r
Starting program: /home/marko/Documentos/Facultad/Fundamentos de Programación/practica10/ejercicio1
TECLEA UN NUMERO: 1

Breakpoint 1, main () at ejercicio1.c:15
15         printf("\nEL RESULTADO ES %i\n", AS);
(gdb) p CONT
$1 = 3
(gdb) p AS
$2 = 1
(gdb)
```

Con el comando *print* o *p* se muestra el valor que contiene cada variable, para la variable CONT, se tiene un valor de 3, debido a la línea número 13 del código, mientras que la variable AS tiene un valor de 1.

Con el comando *continue* o *c* se continuará con el flujo del programa.



```
practica10: gdb — Konsole
Archivo  Editar  Ver  Marcadores  Preferencias  Ayuda

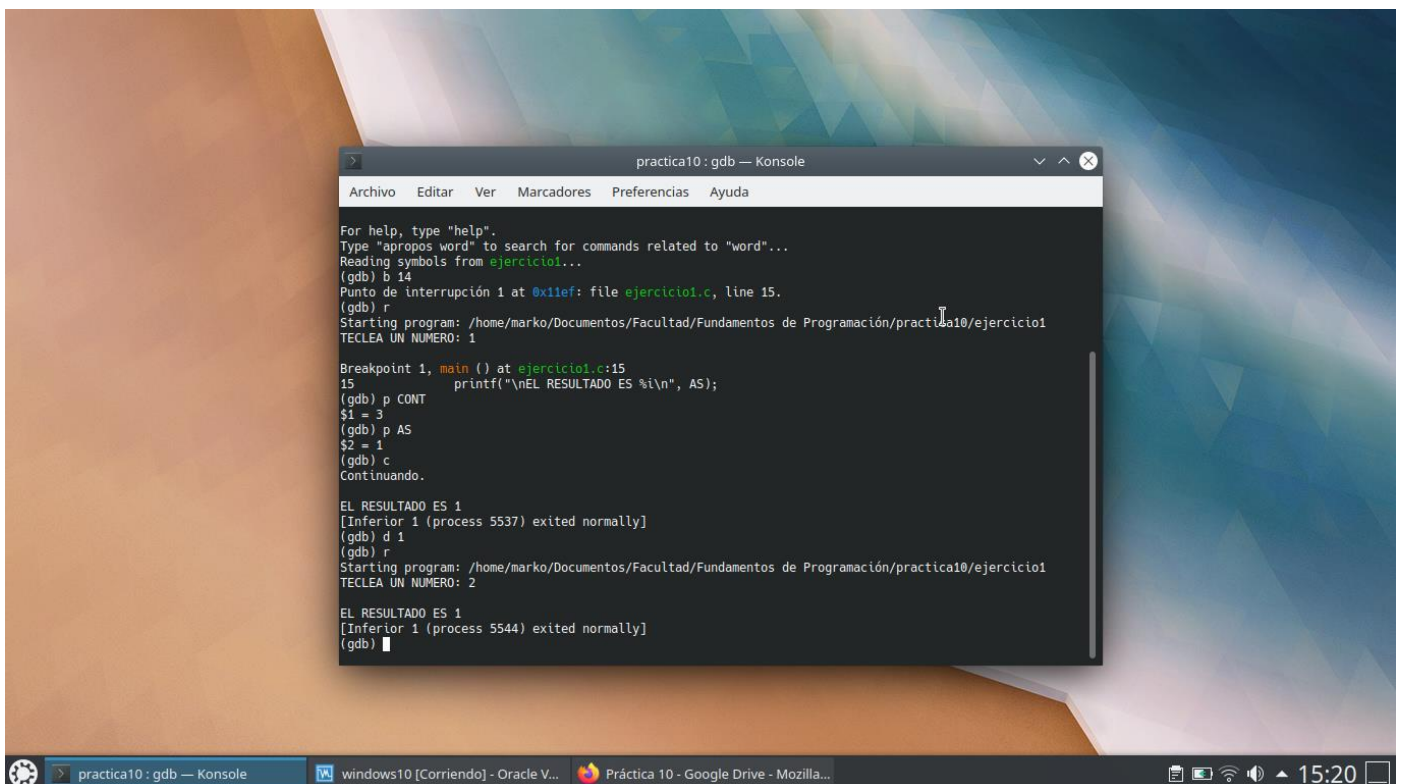
7      CONT=1;
8      printf("TECLEA UN NUMERO: ");
9      scanf("%i",&N);
10     while (CONT<=N)
(gdb) l
11     {
12         AS=(AS+CONT);
13         CONT=(CONT+2);
14     }
15     printf("\nEL RESULTADO ES %i\n", AS);
16 }
(gdb) b 14
Punto de interrupción 1 at 0x11ef: file ejercicio1.c, line 15.
(gdb) r
Starting program: /home/marko/Documentos/Facultad/Fundamentos de Programación/practica10/ejercicio1
TECLEA UN NUMERO: 1

Breakpoint 1, main () at ejercicio1.c:15
15     printf("\nEL RESULTADO ES %i\n", AS);
(gdb) p CONT
$1 = 3
(gdb) p AS
$2 = 1
(gdb) c
Continuando.

EL RESULTADO ES 1
[Inferior 1 (process 5149) exited normally]
(gdb)
```

Debido a que la variable CONT es mayor que N ($3 > 1$) el programa terminará, mostrando en pantalla el valor acumulado de la variable AS.

Eliminamos el *breakpoint* que se había establecido anteriormente con el comando *delete* o *d* y ejecutamos nuevamente el código con el valor de 2.



```
practica10: gdb — Konsole
Archivo  Editar  Ver  Marcadores  Preferencias  Ayuda

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ejercicio1...
(gdb) b 14
Punto de interrupción 1 at 0x11ef: file ejercicio1.c, line 15.
(gdb) r
Starting program: /home/marko/Documentos/Facultad/Fundamentos de Programación/practica10/ejercicio1
TECLEA UN NUMERO: 1

Breakpoint 1, main () at ejercicio1.c:15
15     printf("\nEL RESULTADO ES %i\n", AS);
(gdb) p CONT
$1 = 3
(gdb) p AS
$2 = 1
(gdb) c
Continuando.

EL RESULTADO ES 1
[Inferior 1 (process 5537) exited normally]
(gdb) d 1
(gdb) r
Starting program: /home/marko/Documentos/Facultad/Fundamentos de Programación/practica10/ejercicio1
TECLEA UN NUMERO: 2

EL RESULTADO ES 1
[Inferior 1 (process 5544) exited normally]
(gdb)
```

En esta prueba, ocurre lo mismo que en el caso anterior, puesto que $CONT > N$, entonces el proceso no se ejecuta una segunda vez, mostrando como resultado 1 nuevamente.

A partir de estas dos pruebas, podemos construir una tabla que muestre los valores obtenidos (AS) en función de los valores ingresados (N), para primeros 10 números.

N	1	2	3	4	5	6	7	8	9	10
AS	1	1	4	4	9	9	16	16	25	25

A partir de esto, se observa que la serie muestra el cuadrado de un número entero, por lo que, la funcionalidad de este programa sería calcular el cuadrado de un número entero, sin embargo, al comparar los valores entregados por el programa con los valores reales de los cuadrados de los números enteros del intervalo [1, 10], no coinciden. A falta de comentarios en el programa, y guiándonos por los resultados obtenidos, podemos concluir que el diseño de este programa no es el correcto para un programa que calcule el cuadrado de cierto número entero N.

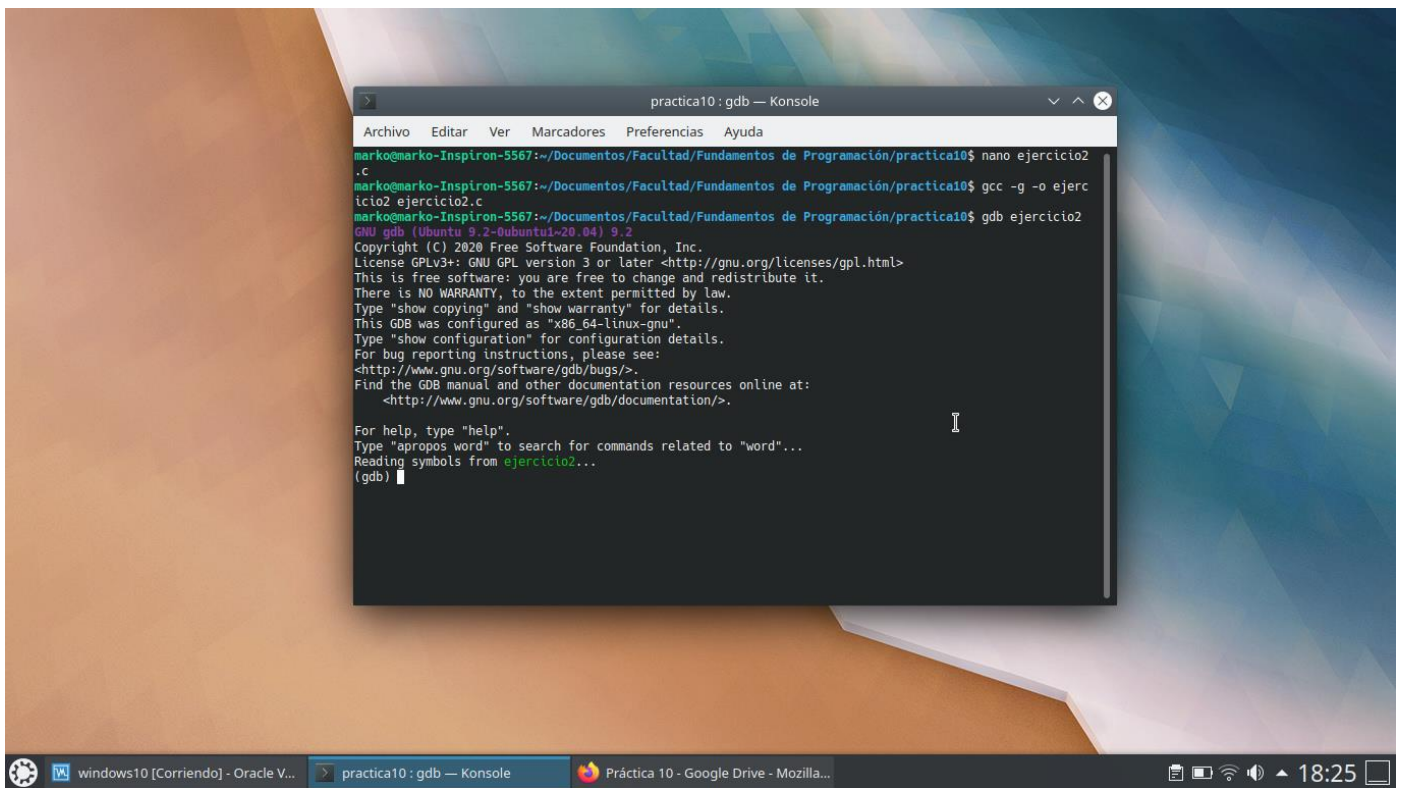
2. El siguiente programa debe mostrar las tablas de multiplicar desde la del 1 hasta la del 10. En un principio no se mostraba la tabla del 10, luego después de intentar corregirse sin un depurador dejaron de mostrarse el resto de las tablas. Usar un depurador de C para averiguar el funcionamiento del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

The screenshot shows a Windows 10 desktop with a taskbar at the bottom. The taskbar includes icons for Windows, Oracle VM VirtualBox, a terminal window titled 'practica10: nano — Konsole', and LibreOffice Draw. The terminal window is open to a file named 'ejercicio2.c' and displays the following C code:

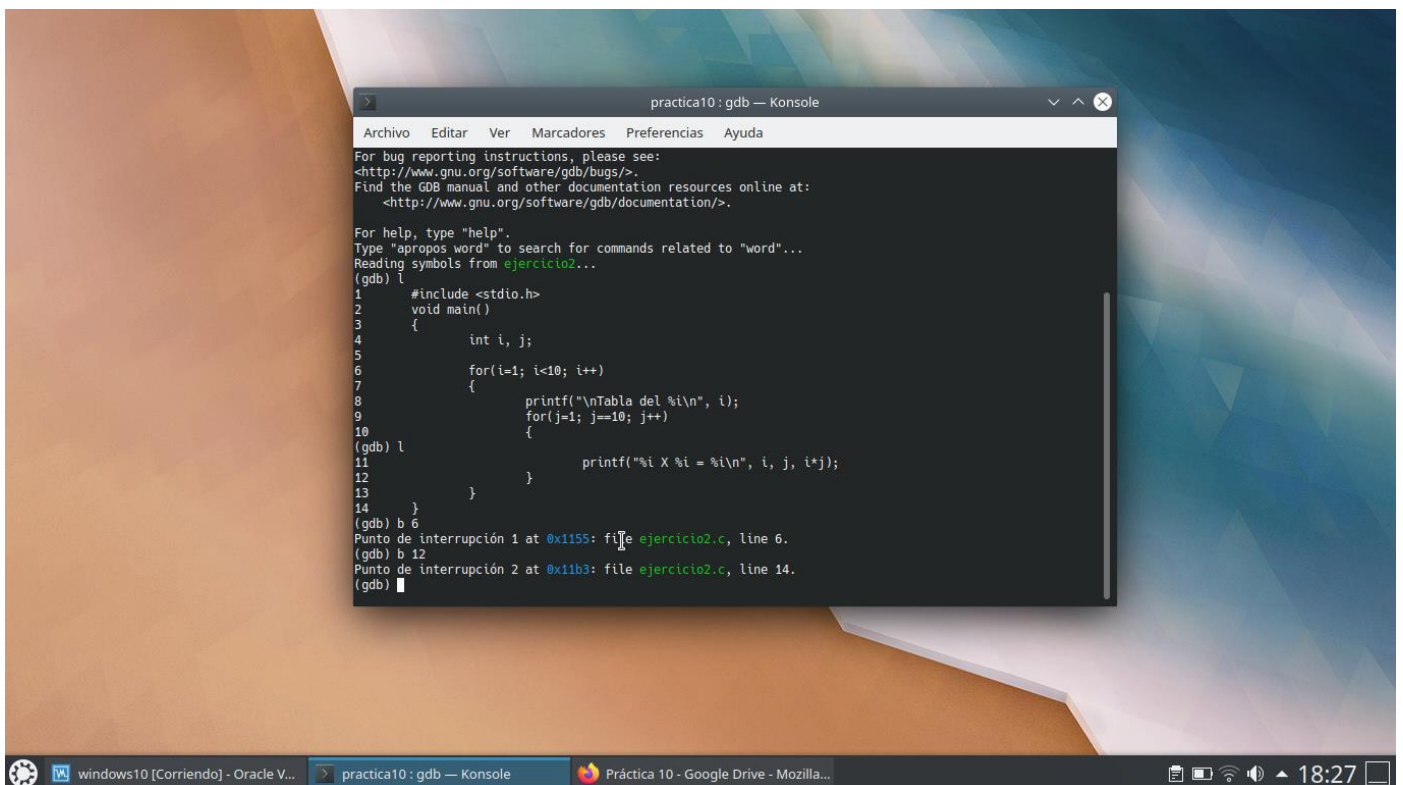
```

GNU nano 4.8
#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);
        }
    }
}
  
```

The terminal window has a menu bar with options: Archivo, Editar, Ver, Marcadores, Preferencias, Ayuda. The status bar at the bottom of the terminal shows various keyboard shortcuts for file operations like 'Ver ayuda', 'Guardar', 'Buscar', etc. The system clock in the bottom right corner of the desktop shows the time as 17:50.



Listamos todas las líneas del código para conocer el número de cada una de ellas y asignamos dos *breakpoints* donde podremos consultar el valor de cada una de las variables involucradas en el programa.



The screenshot shows a GDB console window titled "practica10 : gdb — Konsole". The menu bar includes "Archivo", "Editar", "Ver", "Marcadores", "Preferencias", and "Ayuda". The console output shows the following sequence of commands and results:

```
6 for(i=1; i<10; i++)
(gdb) p i
$1 = 0
(gdb) p j
$2 = 0
(gdb) c
Continuando.

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

Breakpoint 2, main () at ejercicio2.c:14
14 }
(gdb)
```

The taskbar at the bottom shows "windows10 [Corriendo] - Oracle V...", "practica10 : gdb — Konsole", and "Práctica 10 - Google Drive - Mozilla...". The system clock on the right indicates 18:28.

Para el primer *breakpoint* tenemos que la variable *i* tiene un valor de 0 al igual que la variable *j*.

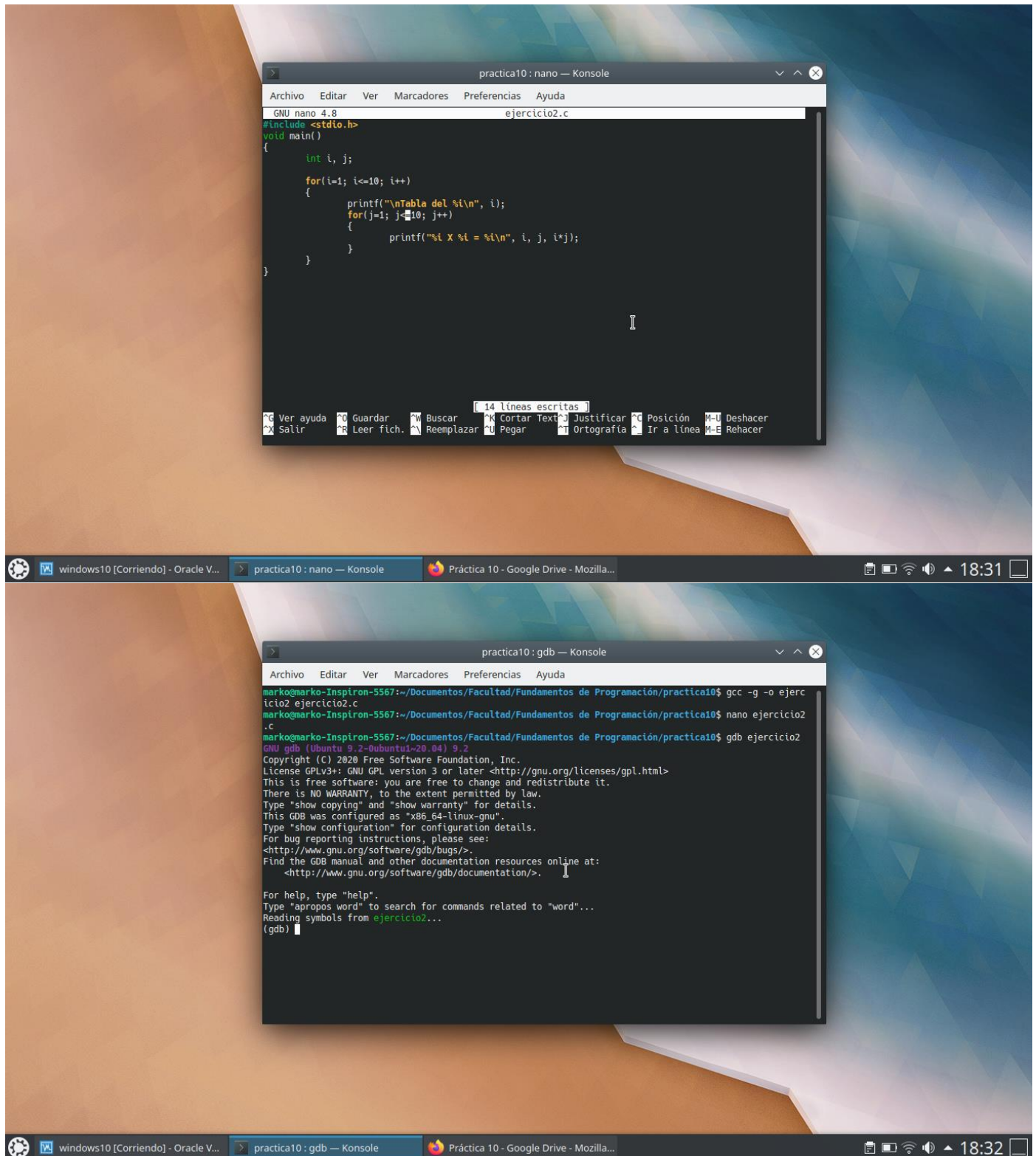
The screenshot shows the same GDB console window at a later point in time. The output now includes the second breakpoint:

```
Breakpoint 2, main () at ejercicio2.c:14
14 }
(gdb) p i
$3 = 10
(gdb) p j
$4 = 1
(gdb)
```

The taskbar and system clock (18:29) are also visible.

Para el segundo *breakpoint* la variable *i* tiene un valor de 10, mientras que la variable *j* tiene un valor de 1. A pesar de que la variable *i* tiene un valor de 10, en la ejecución del programa no mostraba "Tabla del 10", esto se debe a que en la línea número 6 la condición dentro de `for` indica `i<10`, es decir, que el bloque de instrucciones no será repetido si la variable *i* toma el valor de 10, para corregir este error solo se añade un signo de igual (`i<=10`), lo cual indicará al programa que debe ejecutar el bloque de instrucciones hasta que *i* sea mayor a 10. Por otra parte, la *j* se mantiene con el valor de 1 en ambos *breakpoints* debido a que en la línea 9 indica que el bloque de instrucciones

se ejecutará solamente cuando j sea igual a 10 ($j==10$), como la variable se inicia en 1, y 1 es diferente de 10, entonces el bloque de instrucciones no se ejecuta y el programa sigue con el flujo sin entrar ni una sola vez al bloque, de igual forma, para corregir este error basta solo con reemplazar un signo igual por un signo menor que ($j<10$).



```
practica10: gdb — Konsole
Archivo  Editar  Ver  Marcadores  Preferencias  Ayuda
Type "apropos word" to search for commands related to "word"...
Reading symbols from ejercicio2...
(gdb) r
Starting program: /home/marko/Documentos/Facultad/Fundamentos de Programación/practica10/ejercicio2

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

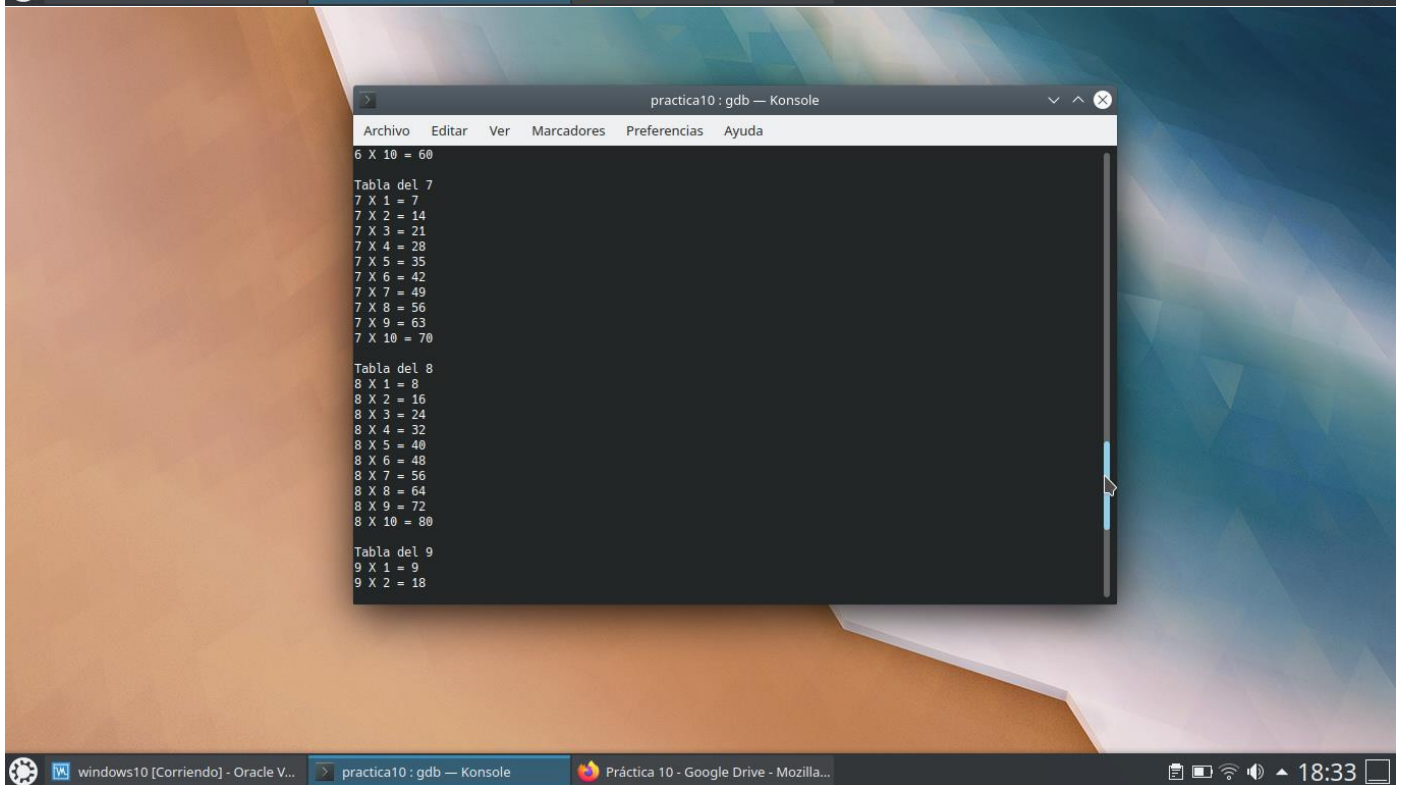
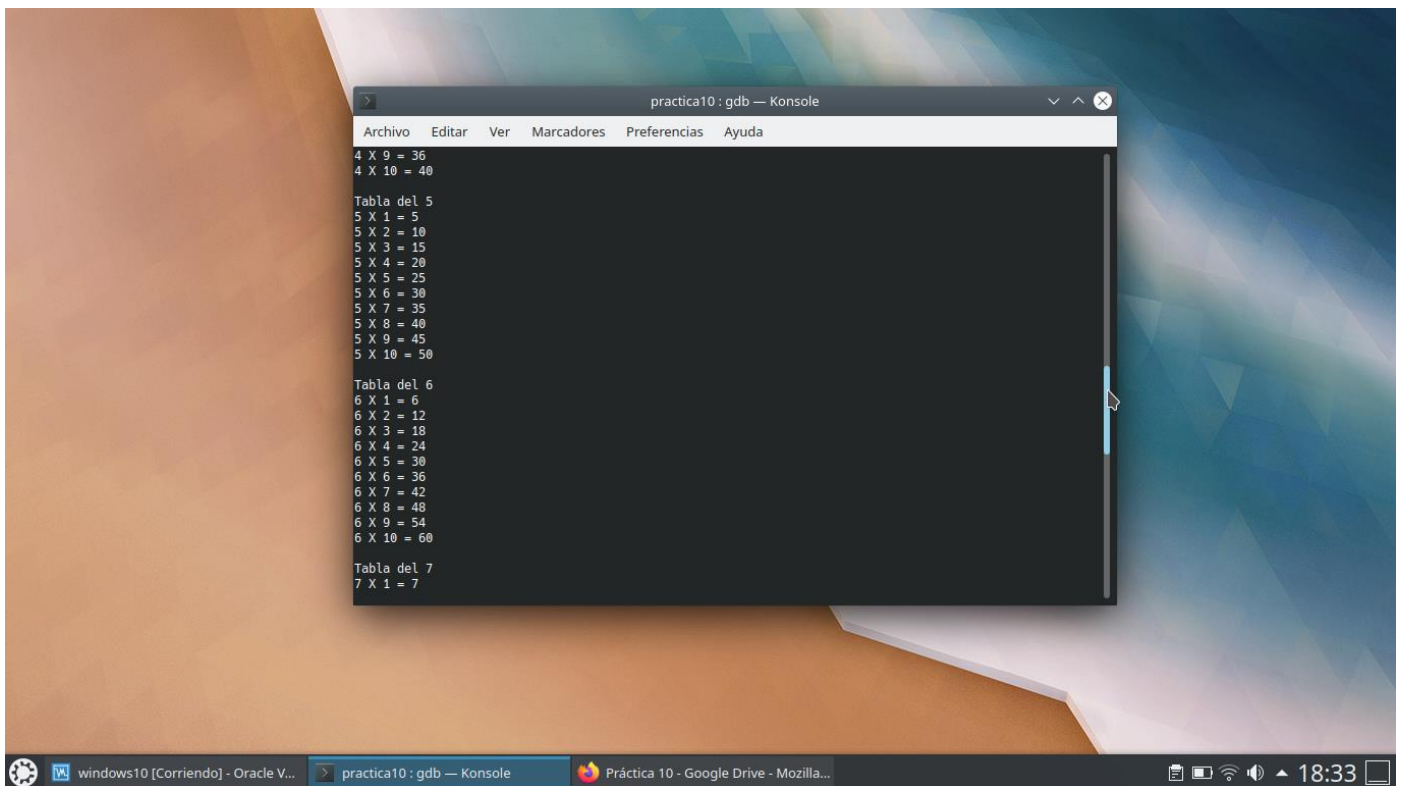
Tabla del 2
2 X 1 = 2
2 X 2 = 4
2 X 3 = 6
2 X 4 = 8
2 X 5 = 10
2 X 6 = 12
2 X 7 = 14
2 X 8 = 16
2 X 9 = 18
2 X 10 = 20
```

```
practica10: gdb — Konsole
Archivo  Editar  Ver  Marcadores  Preferencias  Ayuda

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
```




```
practica10: gdb - Konsole
Archivo  Editar  Ver  Marcadores  Preferencias  Ayuda

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
[Inferior 1 (process 3582) exited with code 016]
(gdb)
```

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

```
practica10: nano - Konsole
GNU nano 4.8                                ejercicio3.c
#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);
}
```

En el caso de este problema, la solución a este podía incluso al compilarlo, ya que muestra en pantalla una advertencia en las líneas 9 y 11.

```
practica10: bash — Konsole
Archivo  Editar  Ver  Marcadores  Preferencias  Ayuda
marko@marko-Inspiron-5567:~/Documentos/Facultad/Fundamentos de Programación/practica10$ nano ejercicio3.c
marko@marko-Inspiron-5567:~/Documentos/Facultad/Fundamentos de Programación/practica10$ gcc -g -o ejercicio3.o ejercicio3.c -lm
ejercicio3.c: In function 'main':
ejercicio3.c:9:10: warning: format '%d' expects argument of type 'int *', but argument 2 has type 'int' [-Wformat=]
     9 |     scanf("%d", N);
       |           ~^   ~
       |           |   int
       |           |   *
ejercicio3.c:11:10: warning: format '%d' expects argument of type 'int *', but argument 2 has type 'int' [-Wformat=]
    11 |     scanf("%d", X);
       |           ~^   ~
       |           |   int
       |           |   *
marko@marko-Inspiron-5567:~/Documentos/Facultad/Fundamentos de Programación/practica10$
```

Iniciamos el depurador y listamos las líneas de código del programa para poder trabajar con el.

```
practica10: gdb — Konsole
Archivo  Editar  Ver  Marcadores  Preferencias  Ayuda
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./ejercicio3...
(gdb) l
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=");
(gdb) l
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         }
(gdb)
```

Al momento de ejecutar el programa y después de ingresar el primer valor solicitado, aparece un error que interrumpe el programa y lo finaliza por error, esto nos indica que puede pasar algo al momento de ingresar los datos.


```
practica10: gdb - Konsole
Archivo  Editar  Ver  Marcadores  Preferencias  Ayuda

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=");
(gdb) l
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    }
(gdb) r
Starting program: /home/marko/Documents/Facultad/Fundamentos de Programación/practica10/ejercicio3
EL TERMINO GENERICO DE LA SERIE ES: X^K/K!
N=1

Program received signal SIGSEGV, Segmentation fault.
0x00007ffff7cd94b5 in __vfscanf_internal (s=<optimized out>, format=<optimized out>,
argptr=argptr@entry=0x7fffffffd930, mode_flags=mode_flags@entry=2) at vfscanf-internal.c:1895
1895 vfscanf-internal.c: No existe el archivo o el directorio.
(gdb)
```

Para comprobar lo anteriormente planteado, procedemos a establecer un *breakpoint* en la línea 9, después de esto movemos el *breakpoint* a la siguiente línea, retirando el anterior.

```
practica10: gdb - Konsole
Archivo  Editar  Ver  Marcadores  Preferencias  Ayuda

Program received signal SIGSEGV, Segmentation fault.
0x00007ffff7cd94b5 in __vfscanf_internal (s=<optimized out>, format=<optimized out>,
argptr=argptr@entry=0x7fffffffd930, mode_flags=mode_flags@entry=2) at vfscanf-internal.c:1895
1895 vfscanf-internal.c: No existe el archivo o el directorio.
(gdb) b 9
Punto de interrupción 1 at 0x7ffff7cd4b3a: /build/glibc-ZN95T4/glibc-2.31/stdio-common/vfscanf-intern
al.c:9. (50 locations)
(gdb) r
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/marko/Documents/Facultad/Fundamentos de Programación/practica10/ejercicio3
EL TERMINO GENERICO DE LA SERIE ES: X^K/K!

Breakpoint 1, char_buffer_rewind (buffer=0x7fffffffd9b0) at vfscanf-internal.c:484
484 vfscanf-internal.c: No existe el archivo o el directorio.
(gdb) d 1
(gdb) b 10
Punto de interrupción 2 at 0x7ffff7cd4b3a: /build/glibc-ZN95T4/glibc-2.31/stdio-common/vfscanf-intern
al.c:10. (50 locations)
(gdb) r
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/marko/Documents/Facultad/Fundamentos de Programación/practica10/ejercicio3
EL TERMINO GENERICO DE LA SERIE ES: X^K/K!

Breakpoint 2, char_buffer_rewind (buffer=0x7fffffffd9b0) at vfscanf-internal.c:484
484 vfscanf-internal.c: No existe el archivo o el directorio.
(gdb)
```

Los resultados nos muestran que el error proviene de la lectura de datos, ya que hace falta tanto en la línea 9 como en la 11, el símbolo "&" el cual almacena el valor ingresado por el usuario en la variable indicada. La corrección del código es simple, basta únicamente con añadir el símbolo (&) en ambas líneas y ejecutará con normalidad.


```
practica10: nano — Konsole
GNU nano 4.8
ejercicio3.c
#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);
}
```

```
practica10: gdb — Konsole
marko@marko-Inspiron-5567:~/Documentos/Facultad/Fundamentos de Programación/practica10$ nano ejercicio3.c
marko@marko-Inspiron-5567:~/Documentos/Facultad/Fundamentos de Programación/practica10$ gcc -g -o ejercicio3 ejercicio3.c -lm
marko@marko-Inspiron-5567:~/Documentos/Facultad/Fundamentos de Programación/practica10$ gdb ./ejercicio3
GNU gdb (Ubuntu 9.2-0ubuntu1~20.04) 9.2
Copyright (C) 2020 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 "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./ejercicio3...
(gdb) r
Starting program: /home/marko/Documentos/Facultad/Fundamentos de Programación/practica10/ejercicio3
EL TERMINO GENERICO DE LA SERIE ES: X*K/K!
N=1
X=2
SUM=3.000000e+00[Inferior 1 (process 4683) exited normally]
(gdb)
```

Conclusión

Las herramientas de depuración, como GDB, permiten visualizar detenidamente los procesos internos de nuestro programa, así como buscar errores de manera más eficiente, ya que, al poder colocar *breakpoints* se puede dar con un error sin la necesidad de modificar el código, para posteriormente modificarlo. Por otra parte, en programas tan pequeños como los presentados en la práctica, las herramientas de depuración pueden hacer más tardado el proceso de encontrar un error, incluso, en el último programa, era innecesario su uso en primera instancia, ya que automáticamente GCC mostraba una advertencia al momento de compilarlo, esto resalta la

importancia de no pasar por alto los mensajes emitidos por los compiladores o IDE's cuando se compila un programa, esto puede ahorrarte tiempo. Siempre debe evitarse ese tipo de omisiones antes de compilar un programa para su depuración.

Referencias

Laboratorio de Computación Salas A y B. (2020). Facultad de Ingeniería. *Manual de prácticas de Fundamentos de Programación MADO-17 EP*. Recuperado de <http://lcp02.fi-b.unam.mx/>