Carátula para entrega de prácticas	
Facultad de Ingeniería	Laboratorio de docencia

Laboratorios de computación salas A y B

Profesor: Alejandro Esteban Pimentel Alarcon

Asignatura: Fundamentos de programación

Grupo: 3

No de Práctica(s): 10

Integrante(s): Rivera Sosa Arlethe
Yuan Xiaojing

No. de Equipo de cómputo empleado:

No. de Lista o Brigada: 317083033
317693612

Semestre: 2020-1

Fecha de entrega: 28/octubre/19

Observaciones:

CALIFICACIÓN: _____

Depuración en programa C

Durante el desarrollo de un programa pueden aparecer diferentes tipos de errores como los léxicos que son más sencillos de corregir, pues son detectados por el propio compilador. Una vez compilado el programa pueden seguir apareciendo fallos durante la ejecución debido a la presencia de errores de ejecución. Estos errores no pueden ser detectados automáticamente y para su corrección se debe repasar mentalmente, instrucción por instrucción, la traza del programa. De este modo podemos detectar el fallo, averiguando en qué momento el comportamiento real del programa se separa del comportamiento esperado.

Un depurador permite el seguimiento de la traza de un programa al facilitar:

• **La ejecución paso a paso de las instrucciones del programa:** Tras la ejecución de una instrucción el entorno devuelve el control del ordenador al programador que puede comprobar cómo se ha ejecutado la instrucción.

• **La monitorización del valor de las variables del programa:** Podemos ver cómo cambia el valor de las variables tras la ejecución de cada instrucción.

GDB, abreviatura de GNU Debugger, es el depurador más popular para los sistemas UNIX para depurar programas C y C ++. El depurador de GNU, que también se llama gdb, es el depurador más popular para los sistemas UNIX para depurar los programas C y C ++.

- GNU Debugger le ayuda a obtener información sobre lo siguiente: Si ocurrió un volcado de memoria, ¿en qué declaración o expresión se bloqueó el programa?, Si se produce un error al ejecutar una función, ¿qué línea del programa contiene la llamada a esa función, y cuáles son los parámetros?, ¿Cuáles son los valores de las variables del programa en un punto particular durante la ejecución del programa? ¿Cuál es el resultado de una expresión particular en un programa?
- GDB es una herramienta que permite entre otras cosas, correr el programa con la posibilidad de detenerlo cuando se cumple cierta condición, avanzar paso a paso, analizar que ha pasado cuando un programa se detiene o cambiar algunas cosas del programa como el valor de las variables. GDB es una herramienta muy poderosa que nos ayudará a encontrar esos errores difíciles, por ejemplo, cuando los punteros no apuntan a donde estamos pensando.

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.

```
Familia@DESKTOP-NE4QAFJ ~
$ gcc ejemplo1.c -o ejemplo1

Familia@DESKTOP-NE4QAFJ ~
$ ./ejemplo1
Primero texto solo
Luego podemos poner un entero: 10
También podemos poner un caracter: B
Un numero real: 89.80
Segmentation fault (`core' generado)

Familia@DESKTOP-NE4QAFJ ~
$ gcc -g ejemplo1.c -o ejemplo1

Familia@DESKTOP-NE4QAFJ ~
$ gdb ./ejemplo1
GNU gdb (GDB) (Cygwin 8.1.1-1) 8.1.1
Copyright (C) 2018 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 "i686-pc-cygwin".
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 ./ejemplo1...done.
(gdb)
```

Utilizamos gdb para que la terminal nos muestre en donde está el error. Primero al correrlo con gdb nos aparecerán como en que consiste lo que acabamos de escribir.

```
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./ejemplo1...done.
(gdb) run
Starting program: /home/Familia/ejemplo1
[New Thread 3032.0x23f8]
[New Thread 3032.0x13c]
[New Thread 3032.0x90c]
[New Thread 3032.0x838]
[New Thread 3032.0x3b6c]
Primero texto solo
Luego podemos poner un entero: 10
También podemos poner un caracter: B
Un numero real: 89.80

Thread 1 "ejemplo1" received signal SIGSEGV, Segmentation fault.
0x0040124c in main (argc=1, argv=0x62cc3c) at ejemplo1.c:21
21                      lista[i] = i;
(gdb) list
16              printf("También podemos poner un caracter: %c\n", caracter);
17              printf("Un numero real: %.2f\n", numeroReal);
18
19              // Podemos llenar la lista con valores
20              for(int i = numero ; i >= numero ; i++){
21                      lista[i] = i;
22              }
23
24              // Y ahora podemos hacer calculos con la lista
25              for(int i = numero ; i >= numero ; i++){
(gdb)
```

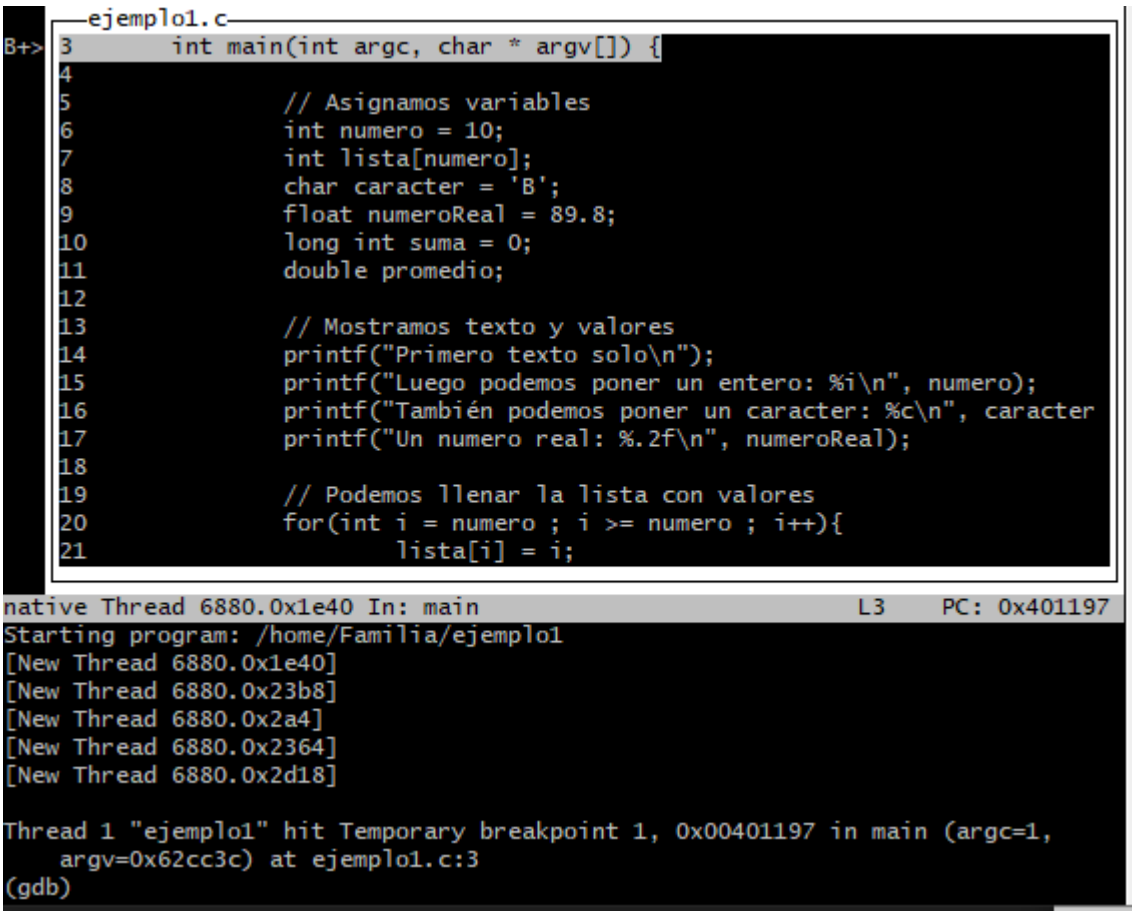
Pondremos run después de (gdb) y nos va a correr todo nuestro programa.

Después de correrlo en lugar de run, se pone list para que nos aparezca todo el programa tal cual está escrito.

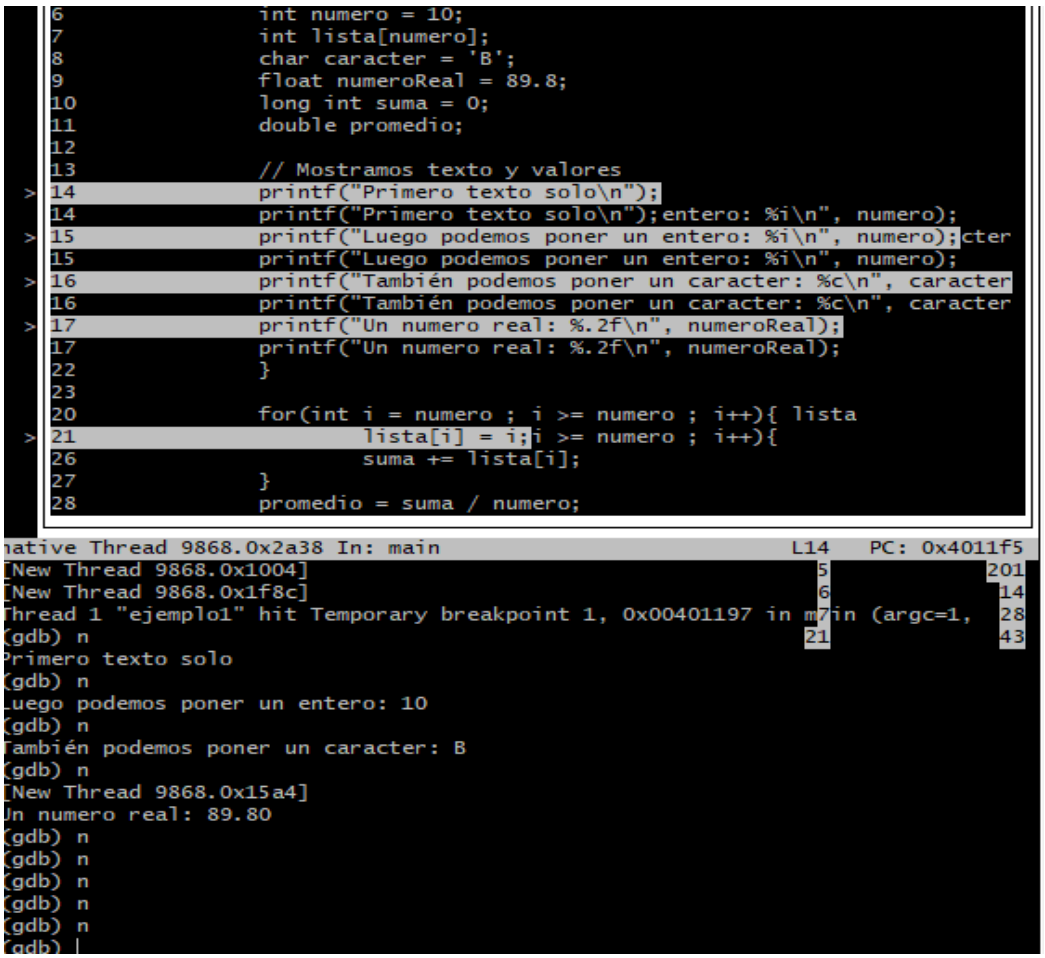
Siempre para salirnos ocupamos “q”. El programa preguntara si queremos salir o no. Si es sí=y, no=n

```
(gdb) q
A debugging session is active.

        Inferior 1 [process 3032] will be killed.

Quit anyway? (y or n) y
```

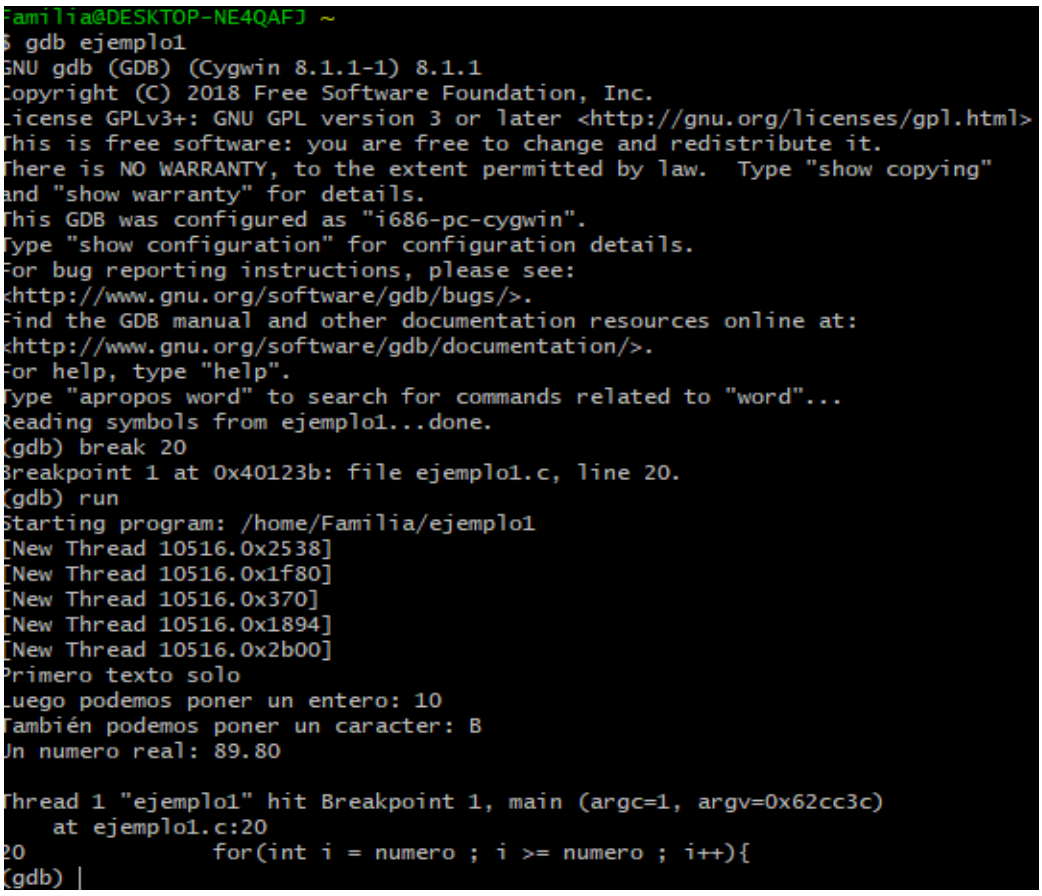
Apretamos Control X+A, nos aparecerá una pantalla como la de abajo solo que vacía y si queremos que nos agregue nuestro programa, hay que poner start y dar enter. Una vez hecho esto, nos aparece nuestro programa y como esta cada línea.



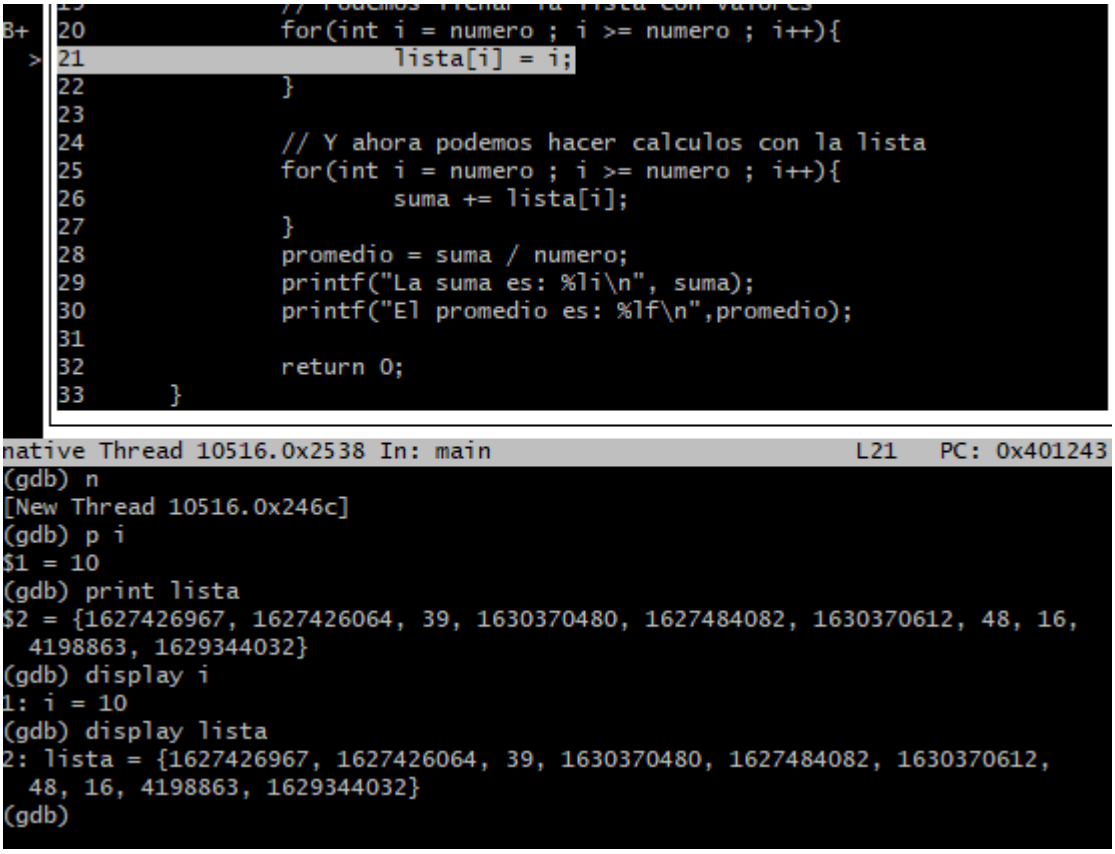
Ocupamos n o la palabra next para movernos hacia abajo, pero hay que tener cuidado porque solo podemos movernos hacia abajo.



Vamos a ir bajado poco a poco en nuestro programa y si cuando bajamos nos empieza a subrayar ciertos renglones y a escribir el texto debajo de donde pusimos n, significa que algo está mal. Igual para salirnos ocupamos la letra q.



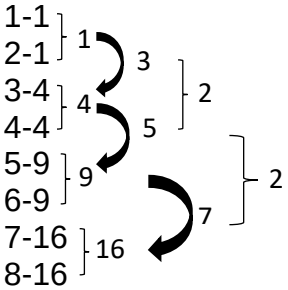
Una vez que nos salimos pondremos: gdb espacio, el nombre de nuestro archivo. Nos aparecera como en el incio. pondremos break 20 (ahí es donde tenemos el problema) y después run para correrlo nuevamente y nos van a aparecer los errores.



Volvemos a poner Control X+A como anteriormente y también start. Nos bajamos a la línea hasta en donde nos aparecían nuestros problemas y escribimos: p i, para que muestre el valor que tiene en ese momento la variable. Ahora pondremos: print lista para que nos imprima la lista de números. Y ahora cambiamos el print por display para que nos diga una lista de valores que puede tener la variable.

Actividad1: Utilizar GDB para encontrar la utilidad del programa y describir su funcionalidad.

Este programa funciona de la siguiente manera:



Coloqué como prueba los primeros ocho números y me di cuenta de que van de par en par, colocando 1 y 2 el número de salida es 1. Después 3 y 4 y su número de salida es el mismo y así sucesivamente. Del primer número de salida vamos sumando número hasta llegar al tercer número de salida (hasta que van cambiando), y nos damos cuenta de que esos números se van incrementando de dos en dos y es así como funciona el programa.

```
Familia@DESKTOP-NE4QAFJ ~
$ ./act
Ingresa un número: 1

El resultado es: 1

Familia@DESKTOP-NE4QAFJ ~
$ ./act
Ingresa un número: 2

El resultado es: 1

Familia@DESKTOP-NE4QAFJ ~
$ ./act
Ingresa un número: 3

El resultado es: 4

Familia@DESKTOP-NE4QAFJ ~
$ ./act 4
Ingresa un número: 4

El resultado es: 4
```

Actividad2: Utilizar GDB para corregir el programa.

- NOTA: para compilar el código de la actividad, ejecutar:
  $ gcc -w actividad2.c -o actividad2 -lm

```
Familia@DESKTOP-NE4QAFJ ~
$ gcc -w actividad2.c -o actividad2 -lm

Familia@DESKTOP-NE4QAFJ ~
$ ./actividad2
Ingrese cuántos términos calcular de la serie: X^K/K!
N= 1 2 3
Segmentation fault (`core' generado)

Familia@DESKTOP-NE4QAFJ ~
$
```

```
8               printf("Ingrese cuántos términos calcular de la serie: X^
9               printf("\nN=");
9               printf("\nN=");
10              scanf("%i",N);
10              scanf("%i",N);;
11              printf("X=");
12              scanf("%lf",X);
15              AS=0;
16              while(K<=N)
17              {
native Thread 9772.0x3b5c In: main                                L9    PC: 0x40119a
[New Thread 9772.0x299c]                                          10              a6
[New Thread 9772.0x1db8]                                          2               c6
Thread 1 "actividad2" hit Temporary breakpoint 1, main () at actividad2.c:8
(gdb) n
(gdb) n
Ingrese cuántos términos calcular de la serie: X^K/K!
(gdb) n
N=n
```

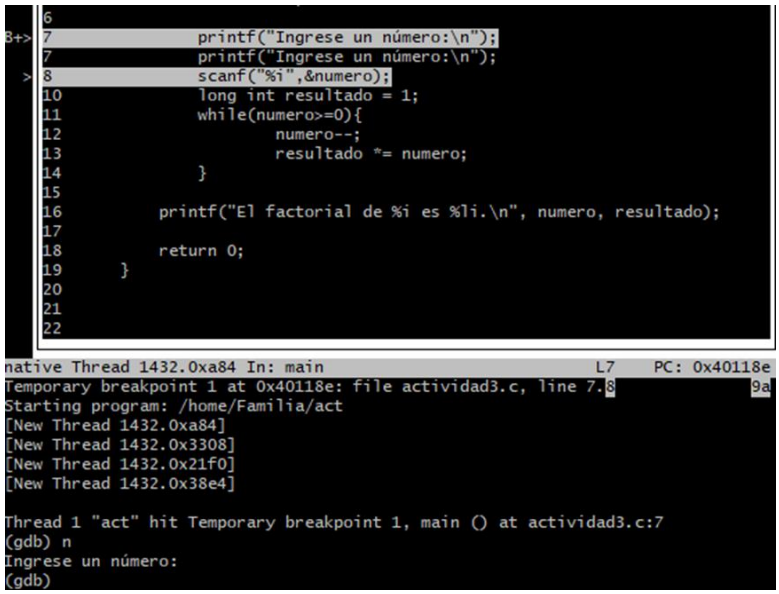
```c
#include <stdio.h>
#include <math.h>

int main()
{
    int K, AP, N;
    double X, AS;
    printf("Ingrese cuantos terminos calcular de la serie: X^K/K!");
    printf("\n N=");
    scanf("%i",&N);
    printf("X=");
    scanf("%lf",&X);
    K=0;
    AP=1;
    AS=0;
    while(K<=N)
    {
        AS=AS+pow(X,K)/AP;
        K=K+1;
        AP=AP*K;
    }
    printf("Resultado=%le",AS);
}
```

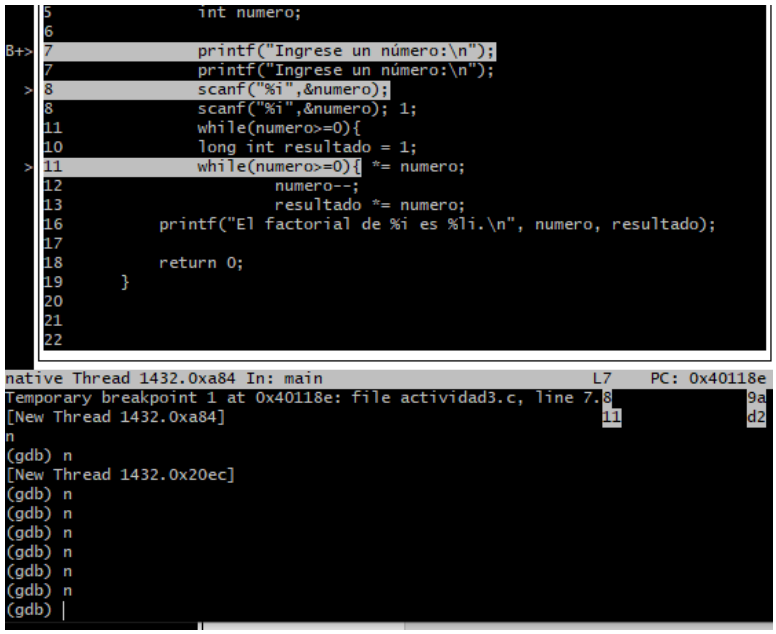
```
Familia@DESKTOP-NE4QAFJ ~
$ ./act
Ingrese cuántos términos calcular de la serie: X^K/K!
 N= 1 2
X=Resultado=3.000000e+00
Familia@DESKTOP-NE4QAFJ ~
$ gcc actividad2.c -o act

Familia@DESKTOP-NE4QAFJ ~
$ ./act
Ingrese cuantos terminos calcular de la serie: X^K/K!
 N= 1 3
X=Resultado=4.000000e+00
```

Después simplemente lo corregimos en las líneas que nos había marcado el programa, lo corremos y vemos que hace una suma.

Actividad3: Utilizar GDB para corregir el programa.

```
Familia@DESKTOP-NE4QAFJ ~
$ gcc actividad3.c -o act

Familia@DESKTOP-NE4QAFJ ~
$ ./act
Ingrese un número:
8
El factorial de -1 es 0.
```


6
B+> 7 printf("Ingrese un número:\n");
7 printf("Ingrese un número:\n");
> 8 scanf("%i",&numero);
10 long int resultado = 1;
11 while(numero>=0){
12 numero--;
13 resultado *= numero;
14 }
15
16 printf("El factorial de %i es %li.\n", numero, resultado);
17
18 return 0;
19 }
20
21
22
native Thread 1432.0xa84 In: main L7 PC: 0x40118e
Temporary breakpoint 1 at 0x40118e: file actividad3.c, line 7.8 9a
Starting program: /home/Familia/act
[New Thread 1432.0xa84]
[New Thread 1432.0x3308]
[New Thread 1432.0x21f0]
[New Thread 1432.0x38e4]
Thread 1 "act" hit Temporary breakpoint 1, main () at actividad3.c:7
(gdb) n
Ingrese un número:
(gdb)


5 int numero;
6
B+> 7 printf("Ingrese un número:\n");
7 printf("Ingrese un número:\n");
> 8 scanf("%i",&numero);
8 scanf("%i",&numero); 1;
11 while(numero>=0){
10 long int resultado = 1;
> 11 while(numero>=0){ *= numero;
12 numero--;
13 resultado *= numero;
16 printf("El factorial de %i es %li.\n", numero, resultado);
17
18 return 0;
19 }
20
21
22
native Thread 1432.0xa84 In: main L7 PC: 0x40118e
Temporary breakpoint 1 at 0x40118e: file actividad3.c, line 7.8 9a
[New Thread 1432.0xa84] 11 d2
n
(gdb) n
[New Thread 1432.0x20ec]
(gdb) n
(gdb) n
(gdb) n
(gdb) n
(gdb) n
(gdb) n
(gdb)

```c
#include <stdio.h>

int main()
{
    int numero;
    int i=1;
    printf("Ingrese un número:\n");
    scanf("%i",&numero);

    long int resultado = 1;
    while(i<=numero){
        resultado=resultado *i;
        i++;
    }

    printf("El factorial de %i es %li.\n", numero, resultado);

    return 0;
}
```

```
Familia@DESKTOP-NE4QAFJ ~
$ gcc actividad3.c -o act

Familia@DESKTOP-NE4QAFJ ~
$ ./act
Ingrese un número:
2
El factorial de 2 es 2.

Familia@DESKTOP-NE4QAFJ ~
$ ./act
Ingrese un número:
3
El factorial de 3 es 6.
```

Aplicamos el mismo procedimiento, utilizamos gdb para ver en donde está el problema y lo corregimos. Para que así nos pueda dar la factorial de ese número.

Conclusión: Gracias a gdb podemos darnos cuenta de cuál es nuestro problema y en qué línea, ya que al compilarlo directamente solo podemos saber ciertos detalles, como puntuación o algunos signos, variables que no declaramos, simplemente errores que no son tan complejos. Mientras que con gdb logramos saber en dónde están nuestros errores para que cuando ejecutemos nuestro programa, este bien hecho y cumpla con la función que queremos.