Carátula para entrega de prácticas	
Facultad de Ingeniería	Laboratorio de docencia

Laboratorios de computación salas A y B

Profesor:	Alejando Pimentel
Asignatura:	Fundamentos de programación
Grupo:	3
No de Práctica(s):	10
Integrante(s):	Uno Karin Natalia
No. de Equipo de cómputo empleado:	26
No. de Lista o Brigada:	8723 #50
Semestre:	1
Fecha de entrega:	28 de octubre del 2019
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.

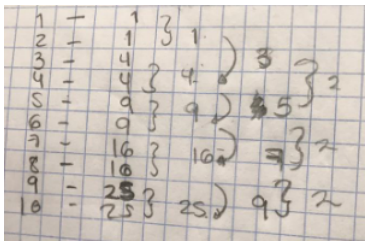
INTRODUCCIÓN
Aquellos que desarrollan en C, conocen de las dificultades a las que se enfrenta cuando trata de depurar un programa, que por ejemplo, por qué no se agrega un nodo a una lista o por qué no se copia determinado string. GDB (Gnu Project Debugger) es una herramienta que permite entre otras 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. Si bien este tutorial está pensado para el lenguaje C, probablemente también sirva para depurar programas en Fortran o C++ con los mismos comandos o similares.

DESARROLLO/RESULTADOS
Actividad 1.

```c
#include <stdio.h>

void main()
{
    int N, CONT, AS;
    AS=0;
    CONT=1;
    printf("Ingresa un número: ");
    scanf("%i",&N);
    while(CONT<=N)
    {
        AS=(AS+CONT);
        CONT=(CONT+2);
    }
    printf("\nEl resultado es: %i\n", AS);
}
```



En este programa se ejecuta lo siguiente:
El programa va incrementando desde cierto numero entero donde ese numero muestra otro numero que se va sumando con cada numero impar y el numero mostrado por el el resultado de la suma del par y conforme incrementa se repite dos veces y se

“muestra” una sola vez, se le resta al numero siguiente repetido, así sucesivamente hasta que todos lleguen a dos .

```
[Karins-MacBook-Pro:~ karinnataliauno$ ./ac
 Ingresa un número: 2

 El resultado es: 1
 Karins-MacBook-Pro:~ karinnataliauno$
```

Así corre el programa.

ACTIVIDAD 2. Corregir el programa.

```
8               printf("Ingrese cuántos términos calcular de
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
[New Thread 9772.0x299c]                                          10
[New Thread 9772.0x1db8]
Thread 1 "actividad2" hit Temporary breakpoint 1, main () at act
(gdb) n
(gdb) n
Ingrese cuántos términos calcular de la serie: X^K/K!
(gdb) n
```

Aquí se muestran los errores que presentó este programa.

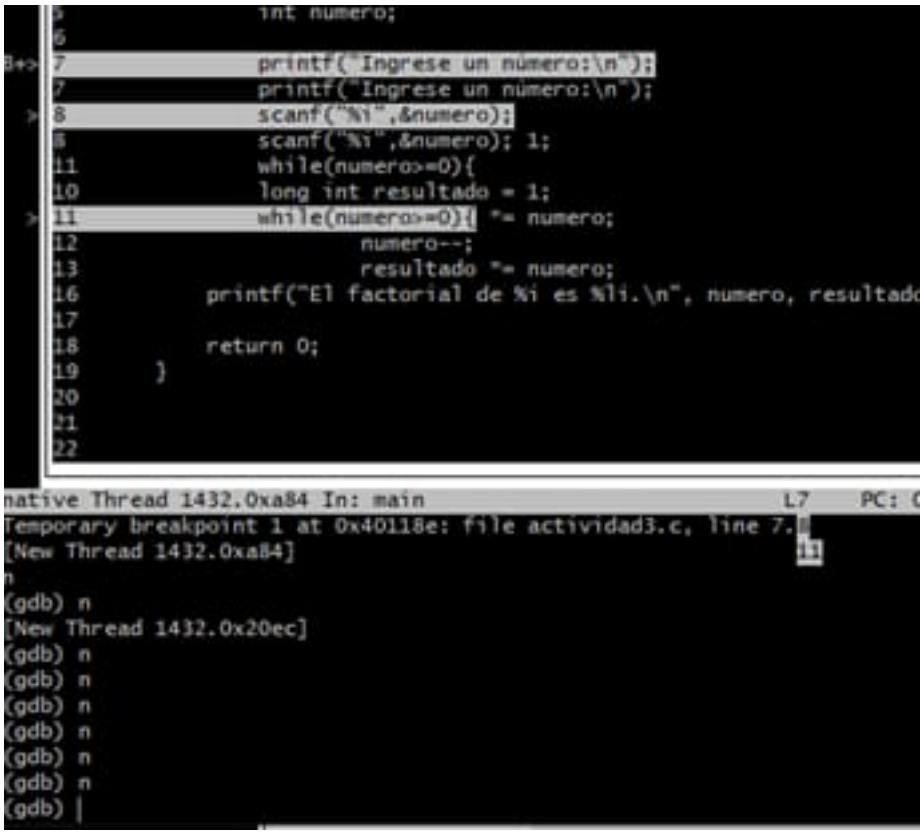
```
1   #include <stdio.h>
2   #include <math.h>
3
4   void main()
5   {
6       int K, AP, N;
7       double X, AS;
8       printf("Ingrese cuántos términos calcular de la serie: X^K/K!");
9       printf("\n N=");
10      scanf("%i",&N);
11      printf("X=");
12      scanf("%lf",&X);
13      K=0;
14      AP=1;
15      AS=0;
16      while(K<=N)
17      {
18          AS=AS+pow(X,K)/AP;
19          K=K+1;
20          AP=AP*K;
21      }
22      printf("Resultado= %le",AS);
23  }
```

Se muestra el programa sin errores de sintaxis.

```
Ingrese cuántos términos calcular de la serie: X^K/K!
 N=2
X=3
Resultado= 8.500000e+00Karins-MacBook-Pro:~ karinnataliauno$
```

Se puede notar que el programa corre correctamente.

ACTIVIDAD 3.Corregir el programa.



```c
#include <stdio.h>

int main()
{
    int numero;
    inti=1;

    printf("Ingrese un número:\n");
    scanf("%i",&numero);

    long int resultado = 1;
    while(numero>=0){

        resultado = numero *1;
        i++
    }

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

    return 0;
}

```

El programa corregido sintácticamente.

El programa ya corregido.

```
Karins-MacBook-Pro:~ karinnataliauno$ ./ac
Ingrese un número:
5
El factorial de -1 es 0.
Karins-MacBook-Pro:~ karinnataliauno$
```

Aquí corre correctamente, hay errores de lógica.

CONCLUSIÓN

En conclusión, el uso del gdb puede ser una forma de corregir de forma más clara y precisa, que el uso de la terminal; ya que este te señala si el programa que se está compilando está correctamente programado mostrando los errores en pantalla.