



## 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

*Integrante(s):* Cuevas Antunez Samantha

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

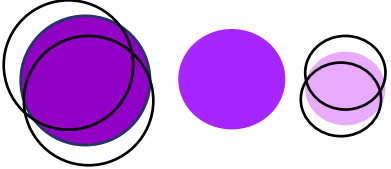
*No. de Lista o Brigada:* 12

*Semestre:* Primer semestre

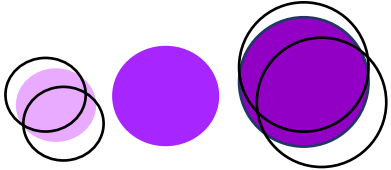
*Fecha de entrega:* 07/12/2020

*Observaciones:*

CALIFICACIÓN: \_\_\_\_\_



# Depuración de programas



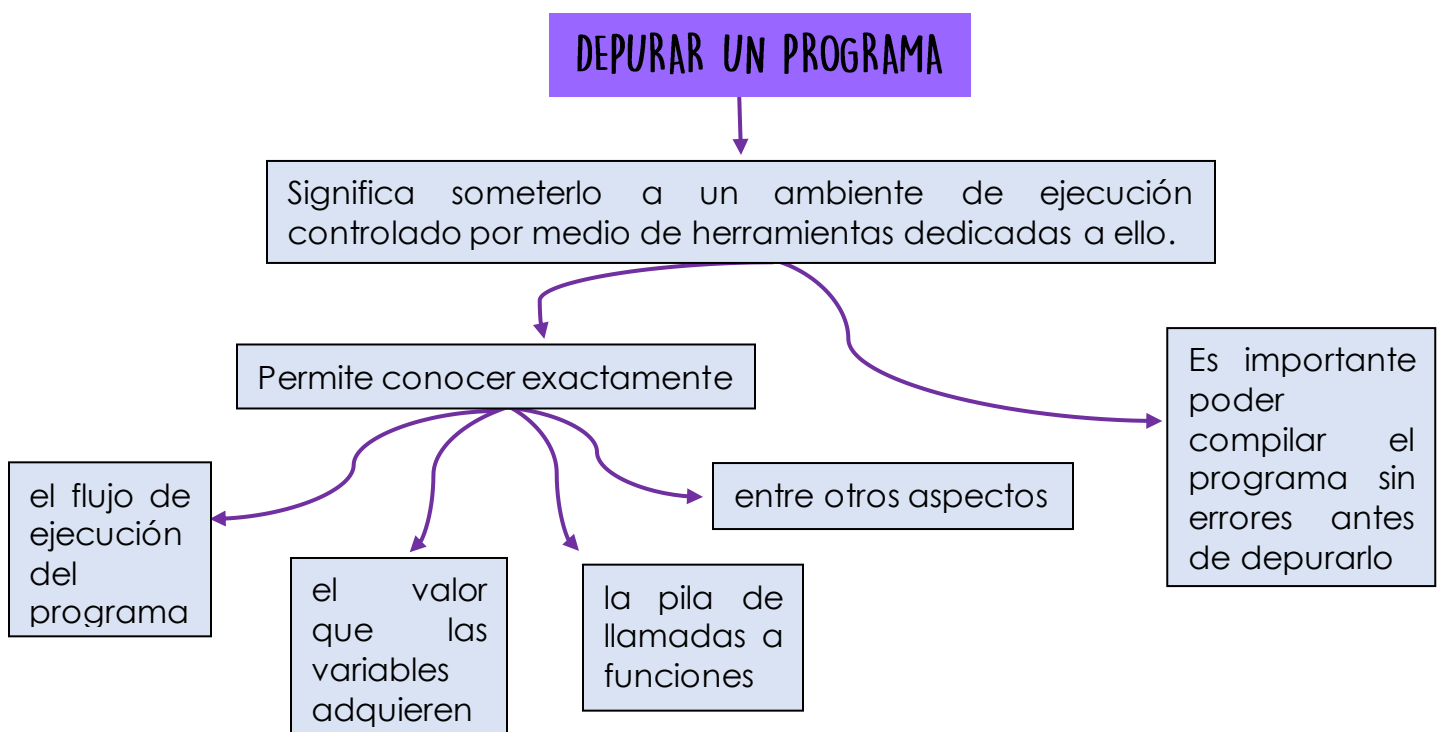
## OBJETIVOS

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



Definiciones pertinentes:

- ✓ 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. 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. 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, se indica qué parte del código fuente se está ejecutando a la hora

de correr el programa.

- ❖ Punto de ruptura/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.
- ❖ 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:

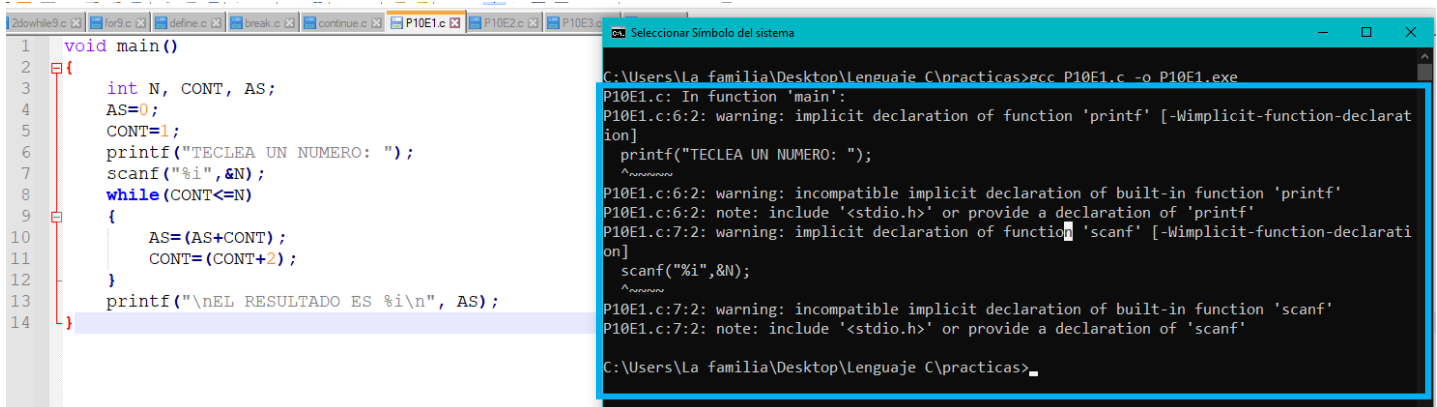
1. *list* o *l*: Permite listar diez líneas del código fuente del programa, si se desea visualizar todo el código fuente debe invocarse varias veces este comando para mostrar de diez en diez líneas. Se puede optar por colocar un número separado por un espacio para indicar a partir de qué línea desea 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`
2. *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`

3. *d* o *delete*: Elimina un punto de ruptura, indicando cuál es el que debe eliminarse usando el número de línea. Ejemplo: *d 5*
4. *clear*: Elimina todos los puntos de ruptura. Ejemplo: *clear*
5. *info line*: Permite mostrar información relativa a la línea que se indique después del comando. Ejemplo: *info line 8*
6. *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.
7. *c*: Continúa con la ejecución del programa después de un punto de ruptura.
8. *s*: Continúa con la siguiente instrucción después de un punto de ruptura.
9. *n*: Salta hasta la siguiente línea de código después de un punto de ruptura.
10. *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*
11. *ignore*: Ignora un determinado punto de ruptura indicándolo con el número de línea de código. Ejemplo: *ignore 5*
12. *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*.

# RESULTADOS

## Ejercicio 1.



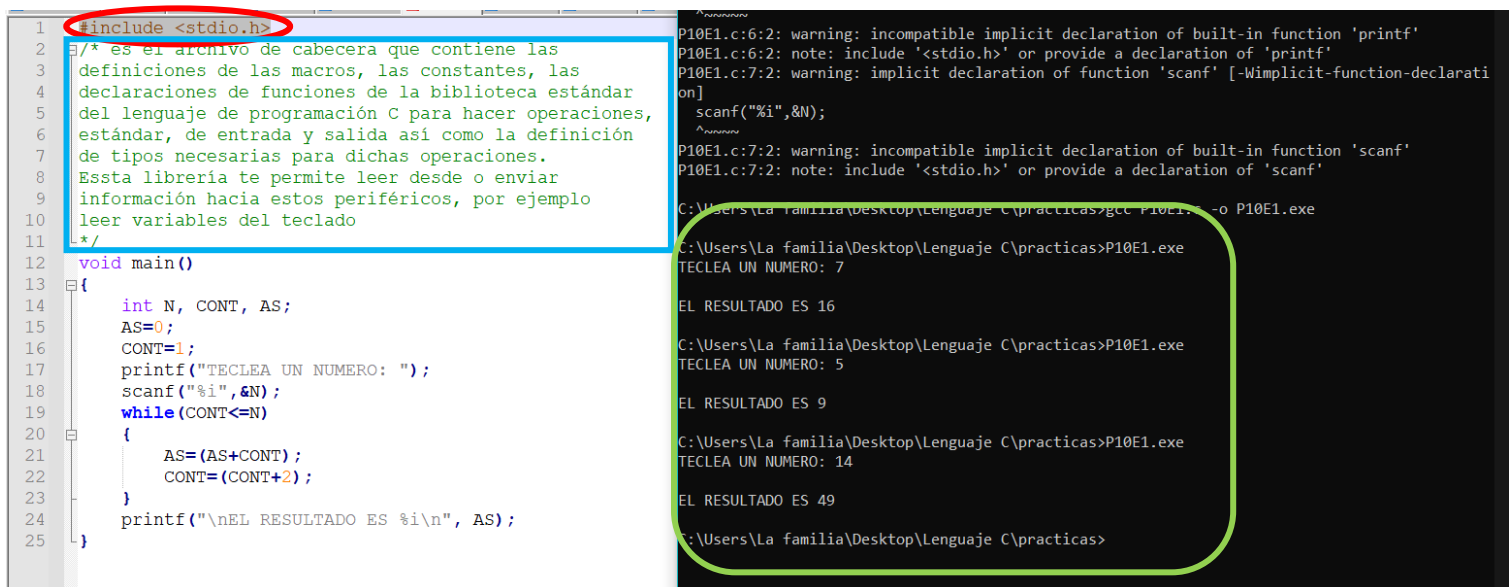
The screenshot shows a C program in a text editor and its compilation output in a terminal window. The program is as follows:

```
1 void main()
2 {
3     int N, CONT, AS;
4     AS=0;
5     CONT=1;
6     printf("TECLEA UN NUMERO: ");
7     scanf("%i",&N);
8     while (CONT<=N)
9     {
10         AS=(AS+CONT);
11         CONT=(CONT+2);
12     }
13     printf("\nEL RESULTADO ES %i\n", AS);
14 }
```

The terminal window shows the command `gcc P10E1.c -o P10E1.exe` and the following warnings:

```
P10E1.c: In function 'main':
P10E1.c:6:2: warning: implicit declaration of function 'printf' [-Wimplicit-function-declaration]
printf("TECLEA UN NUMERO: ");
P10E1.c:6:2: warning: incompatible implicit declaration of built-in function 'printf'
P10E1.c:6:2: note: include '<stdio.h>' or provide a declaration of 'printf'
P10E1.c:7:2: warning: implicit declaration of function 'scanf' [-Wimplicit-function-declaration]
scanf("%i",&N);
P10E1.c:7:2: warning: incompatible implicit declaration of built-in function 'scanf'
P10E1.c:7:2: note: include '<stdio.h>' or provide a declaration of 'scanf'
```

## Solución



The screenshot shows the corrected C program with the `#include <stdio.h>` line highlighted in red. The program is as follows:

```
1 #include <stdio.h>
2 /* es el archivo de cabecera que contiene las
3 definiciones de las macros, las constantes, las
4 declaraciones de funciones de la biblioteca estándar
5 del lenguaje de programación C para hacer operaciones,
6 estándar, de entrada y salida así como la definición
7 de tipos necesarias para dichas operaciones.
8 Esta librería te permite leer desde o enviar
9 información hacia estos periféricos, por ejemplo
10 leer variables del teclado
11 */
12 void main()
13 {
14     int N, CONT, AS;
15     AS=0;
16     CONT=1;
17     printf("TECLEA UN NUMERO: ");
18     scanf("%i",&N);
19     while (CONT<=N)
20     {
21         AS=(AS+CONT);
22         CONT=(CONT+2);
23     }
24     printf("\nEL RESULTADO ES %i\n", AS);
25 }
```

The terminal window shows the command `gcc P10E1.c -o P10E1.exe` and the following warnings:

```
P10E1.c:6:2: warning: incompatible implicit declaration of built-in function 'printf'
P10E1.c:6:2: note: include '<stdio.h>' or provide a declaration of 'printf'
P10E1.c:7:2: warning: implicit declaration of function 'scanf' [-Wimplicit-function-declaration]
scanf("%i",&N);
P10E1.c:7:2: warning: incompatible implicit declaration of built-in function 'scanf'
P10E1.c:7:2: note: include '<stdio.h>' or provide a declaration of 'scanf'
```

The execution output shows the program running and displaying the results:

```
C:\Users\La familia\Desktop\Lenguaje C\practicasp10E1.exe
TECLEA UN NUMERO: 7

EL RESULTADO ES 16

C:\Users\La familia\Desktop\Lenguaje C\practicasp10E1.exe
TECLEA UN NUMERO: 5

EL RESULTADO ES 9

C:\Users\La familia\Desktop\Lenguaje C\practicasp10E1.exe
TECLEA UN NUMERO: 14

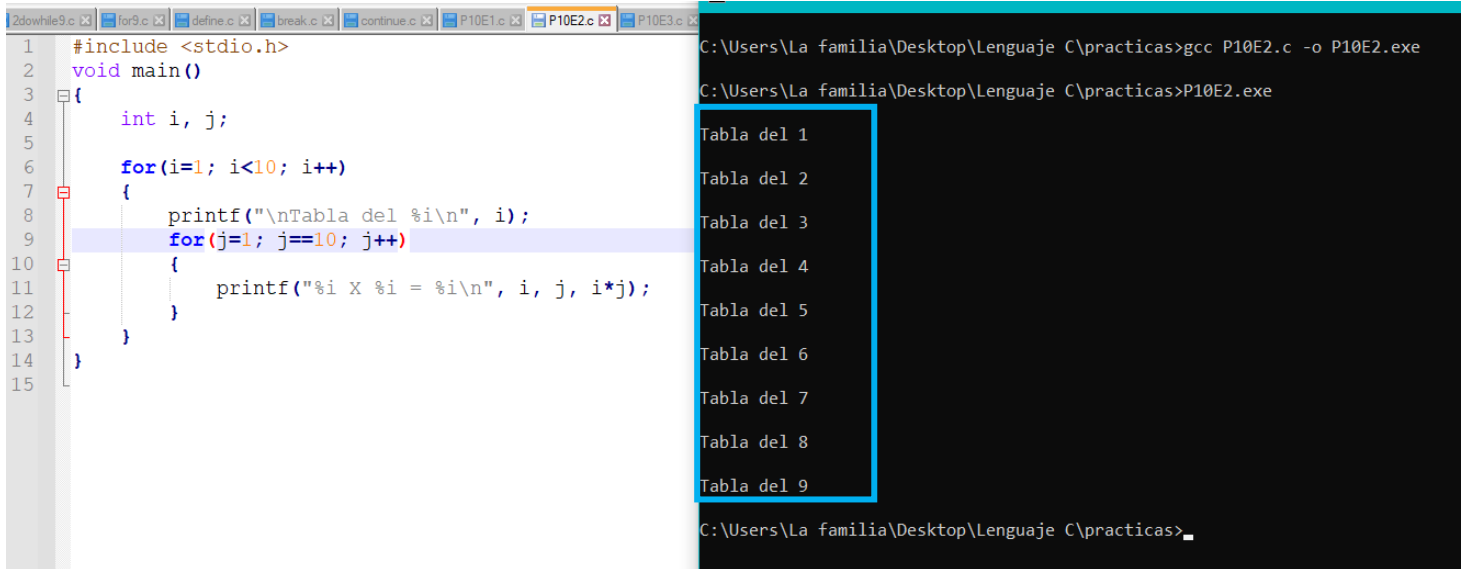
EL RESULTADO ES 49

C:\Users\La familia\Desktop\Lenguaje C\practicasp10E1.exe
```

Funcionalidad del programa.

El programa guarda un número, y le suma un contador siempre que el número sea mayor que el contador y después le suma 2 al contador.

## Ejercicio 2



The screenshot shows a C program in a text editor and its execution output in a terminal. The program is designed to print multiplication tables for numbers 1 through 9. The code uses nested loops: an outer loop for the multiplier (i) and an inner loop for the multiplicand (j). The output in the terminal shows the first nine tables, with the text 'Tabla del 1' through 'Tabla del 9' highlighted by a blue box.

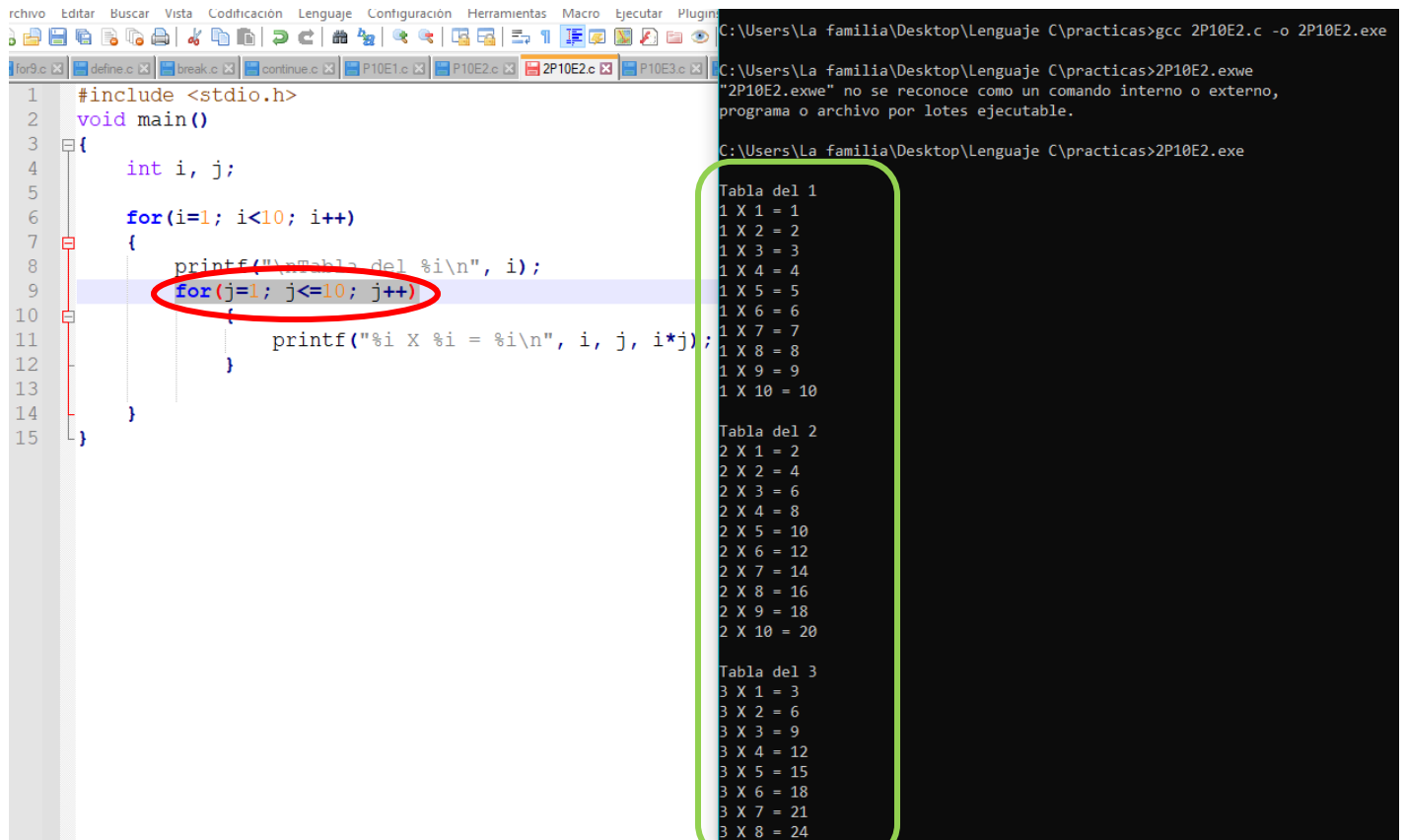
```
1 #include <stdio.h>
2 void main()
3 {
4     int i, j;
5
6     for(i=1; i<10; i++)
7     {
8         printf("\nTabla del %i\n", i);
9         for(j=1; j<=10; j++)
10        {
11            printf("%i X %i = %i\n", i, j, i*j);
12        }
13    }
14 }
15
```

```
C:\Users\La familia\Desktop\Lenguaje C\practicas>gcc P10E2.c -o P10E2.exe
C:\Users\La familia\Desktop\Lenguaje C\practicas>P10E2.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\La familia\Desktop\Lenguaje C\practicas>
```

## Solución



The screenshot shows a C program in a text editor and its execution output in a terminal. The program is designed to print multiplication tables for numbers 1 through 3. The code uses nested loops: an outer loop for the multiplier (i) and an inner loop for the multiplicand (j). The output in the terminal shows the first three tables, with the text 'Tabla del 1', 'Tabla del 2', and 'Tabla del 3' highlighted by a green box. The inner loop condition in the code is circled in red, indicating a correction from the previous version.

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

```
C:\Users\La familia\Desktop\Lenguaje C\practicas>gcc 2P10E2.c -o 2P10E2.exe
C:\Users\La familia\Desktop\Lenguaje C\practicas>2P10E2.exe
"2P10E2.exe" no se reconoce como un comando interno o externo,
programa o archivo por lotes ejecutable.

C:\Users\La familia\Desktop\Lenguaje C\practicas>2P10E2.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 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
```

```
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
```

```
C:\Users\La familia\Desktop\Lenguaje C\practicas>
```

Funcionalidad del programa.

El programa muestra las tablas de multiplicar del 1-10, esto se pudo arreglar de diversas formas, el más fácil fue el arreglar el segundo for poniendo un `<=` en la expresión lógica.

### Ejercicio 3

```
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("SUM=%le", AS);
22 }
```

```
C:\Users\La familia\Desktop\Lenguaje C\practicas>gcc P10E3.c -o P10E3.exe
C:\Users\La familia\Desktop\Lenguaje C\practicas>P10E3.exe
EL TERMINO GENERICO DE LA SERIE ES: X^K/K!
N=8
C:\Users\La familia\Desktop\Lenguaje C\practicas>
```

### Solución

El error que se nota es el hecho de que no se están guardando los valores de las variables, como sabemos para que se guarde este valor basta con agregar un &

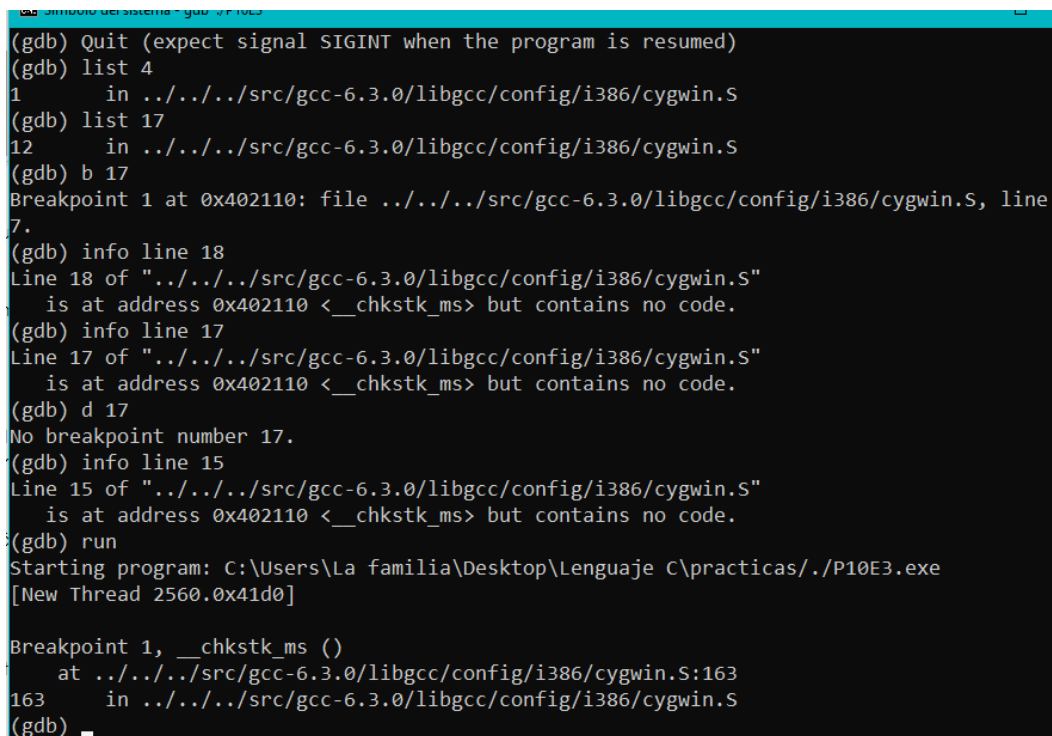
```
1 #include <stdio.h>
2 #include <math.h>
3 void main()
4 {
5     int K, X, AP, N;
6     float AS;
7     printf("EL TERMINO GENERICO DE LA SERIE ES: X^K/K!");
8     printf("\nN=");
9     scanf("%d", &N);
10    printf("X=");
11    scanf("%d", &X);
12    K=0;
13    AP=1;
14    AS=0;
15    while (K<=N)
16    {
17        AS=AS+pow(X,K)/AP;
18        K=K+1;
19        AP=AP*K;
20    }
21    printf("SUM=%ld", AS);
22 }
```

```
C:\Users\La familia\Desktop\Lenguaje C\practicas>P10E3.exe
EL TERMINO GENERICO DE LA SERIE ES: X^K/K!
N=7
X=9
SUM=2.624564e+003
C:\Users\La familia\Desktop\Lenguaje C\practicas>P10E3.exe
EL TERMINO GENERICO DE LA SERIE ES: X^K/K!
N=5
X=8
SUM=5.700667e+002
C:\Users\La familia\Desktop\Lenguaje C\practicas>
```

## CONCLUSIONES

Depurar programas no ayuda a mejorarlo de distintas maneras, desde hacer el código más eficaz o inclusive arreglar los problemas que este presenta. Una característica destacable de la depuración es que da las herramientas necesarias para revisar un programa línea por línea.

El gcc -o ya lo habíamos usado en otros programas anteriores, a pesar de usar y deducir que servía para señalar los errores en la sintaxis de dichos programas no conocía el depurador GDB y a pesar de que lo utilicé e intenté usar algunos de los comandos mencionados no entendí muy bien lo que me indicaban pero pude ver el funcionamiento de cada uno de ellos.



```
(gdb) Quit (expect signal SIGINT when the program is resumed)
(gdb) list 4
1      in ../../src/gcc-6.3.0/libgcc/config/i386/cygwin.S
(gdb) list 17
12     in ../../src/gcc-6.3.0/libgcc/config/i386/cygwin.S
(gdb) b 17
Breakpoint 1 at 0x402110: file ../../src/gcc-6.3.0/libgcc/config/i386/cygwin.S, line
7.
(gdb) info line 18
Line 18 of "../../src/gcc-6.3.0/libgcc/config/i386/cygwin.S"
      is at address 0x402110 <__chkstk_ms> but contains no code.
(gdb) info line 17
Line 17 of "../../src/gcc-6.3.0/libgcc/config/i386/cygwin.S"
      is at address 0x402110 <__chkstk_ms> but contains no code.
(gdb) d 17
No breakpoint number 17.
(gdb) info line 15
Line 15 of "../../src/gcc-6.3.0/libgcc/config/i386/cygwin.S"
      is at address 0x402110 <__chkstk_ms> but contains no code.
(gdb) run
Starting program: C:\Users\La familia\Desktop\Lenguaje C\practicas/./P10E3.exe
[New Thread 2560.0x41d0]

Breakpoint 1, __chkstk_ms ()
      at ../../src/gcc-6.3.0/libgcc/config/i386/cygwin.S:163
163     in ../../src/gcc-6.3.0/libgcc/config/i386/cygwin.S
(gdb) _
```

## REFERENCIAS

Facultad de Ingeniería. (2018, 6 abril). Manual de prácticas de fundamentos de programación MADO-17\_FP. Laboratorio de Computación Salas A y B. <http://lcp02.fi-b.unam.mx/>

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)