|  |  |  |
| --- | --- | --- |
|  | **Carátula para entrega de prácticas** | |
| Facultad de Ingeniería | | Laboratorio de docencia |

Laboratorios de computación

salas A y B

|  |  |
| --- | --- |
| *Profesor:* | Garcia Morales Karina |
| *Asignatura:* | Fundamentos de Programación |
| *Grupo:* | 22 |
| *No de Práctica(s):* | Practica 10 |
| *Integrante(s):* | Ramos Orozco Daniel |
|  |  |
|  |  |
| *Semestre:* | 2018-2 |
| *Fecha de entrega:* | 02-05-2018 |
| *Observaciones:* |  |
|  |  |

CALIFICACIÓN: \_\_\_\_\_\_\_\_\_\_

**Objetivo:**

Elaborar 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.

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.

Definiciones:

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:

* 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.

El comando que usamos es:

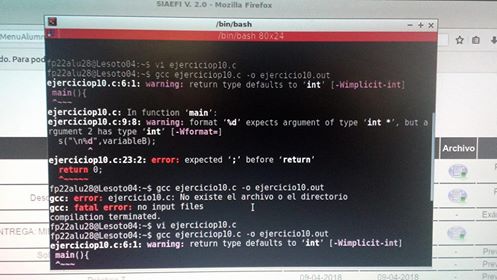
*“gcc -g -o programa programa.c”*

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

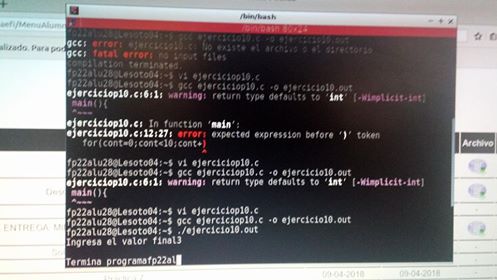
Para depurar un ejecutable debe invocarse a GDB en la terminal indicando cuál es el programa ejecutable a depurar

*“gdb ./programa”*

Para poder realizarlo utilizamos un programa dado por la maestra y lo compilamos y nos marco errores:

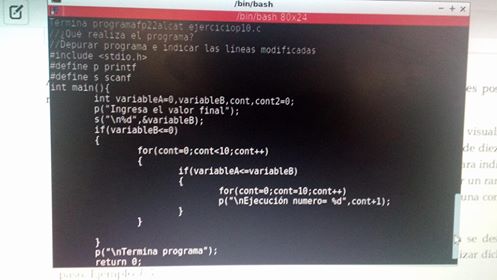


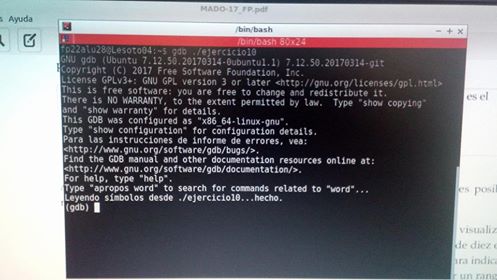
Entonces entramos al código para ver los errores de sintaxis y los corregimos entonces compilamos y ejecutamos



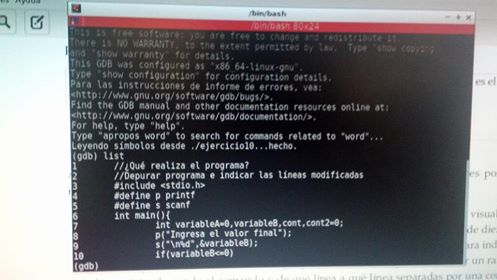
Pero nos dimos cuenta que habían otro tipo de errores.

Entonces utilizamos el comando “cat” par avisualizar el código y después lo compilamos con el gdb para poderlo depurar el programa

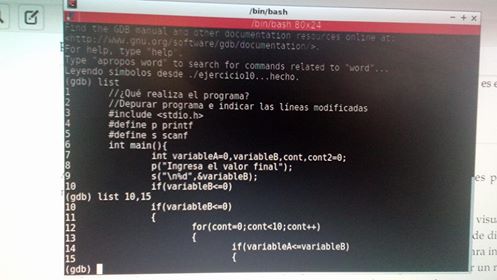




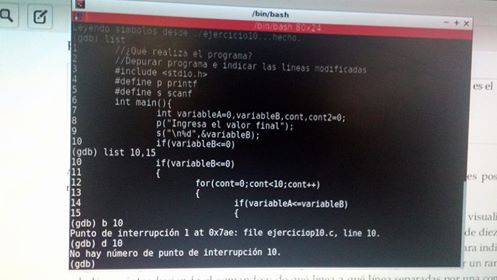
Entonces usamos el comando “list” para poder visualizar el código, solo que por sí solo te muestra 10 líneas del código:



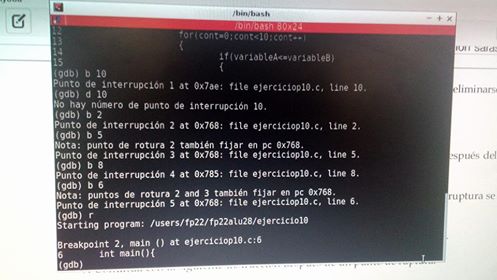
Pero podemos colocarle después del “list” de que número a que numero queremos que nos muestre



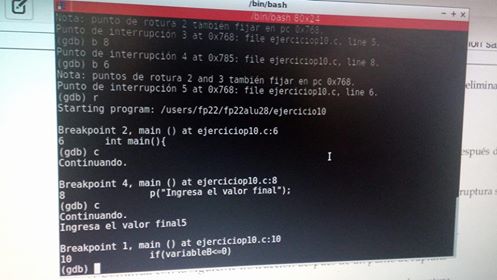
Después pudimos ver que si colocamos una “b” un espacio y un número así podíamos poner un punto de ruptura para después ver en donde se detiene el programa y poder ver el error; y después vimos que si colocamos una “d” un espacio y el número en donde hayamos puesto el punto de ruptura así lo podemos eliminar:



Posteriormente colocamos más puntos de ruptura y con la letra “r” pudimos correr nuestro programa:

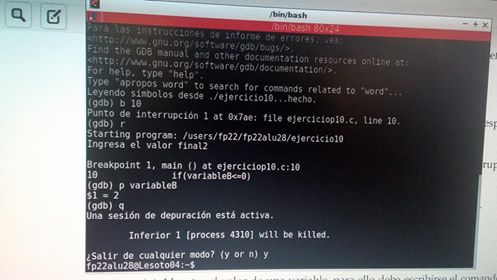


Después pudimos ver que si le colocábamos una “c” podíamos continuar el código, para ver los demás errores:

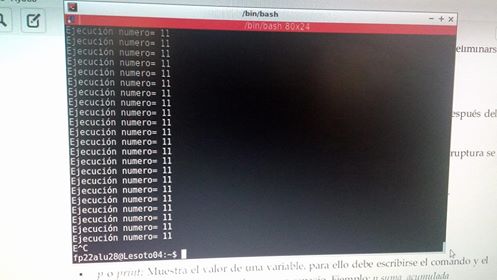


Ahora, corregimos algunos errores y volvimos a entrar al depurados para ver si todo estaba bien y lo corrimos con “r” pero antes de eso pusimos un punto de ruptur y eso hizo que nos diéramos cuanta de los demás erores que habían.

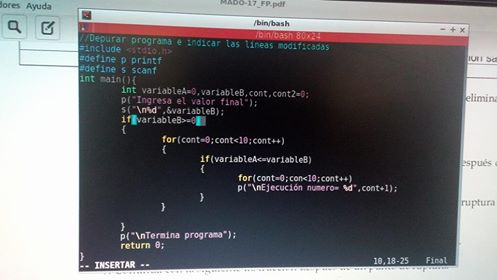
Entonces usamos la letra “p” un espacio y el nombre de una variable a visualizar para pode ver que cual es el resultado y después de ver que estaba mal le dimos “q” que era quitar (para salir del depurador) y depues colocamos una “y” de que si queríamos salir:



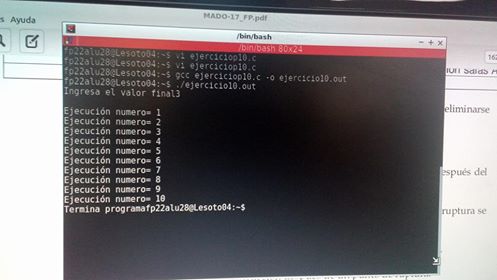
Después de corregirlo lo compilamos y ejecutamos y se fue a infinito:



Apretamos Ctrl+C par aparar el bucle y después lo corregimos:



Lo ejecutamos y compilamos para ver si el resultado si era el correcto:



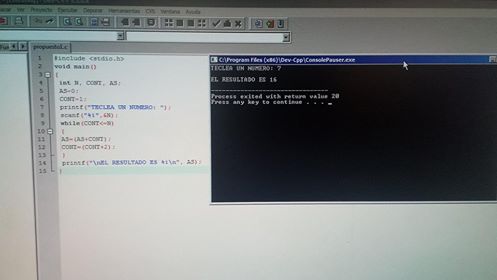
Y si ahora si nos ejecuto correctamente.

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

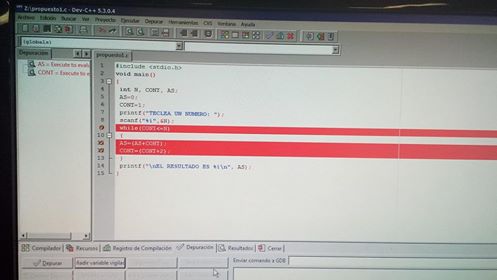
Dev-C++, es una IDE especializada para desarrollar programas escritos en C o C++. Si bien incorpora un editor de textos y un compilador integrados, también posee un depurador. Cabe destacar que por defecto Dev-C++ se basa en el compilador GCC y el depurador en GDB, aunque de manera gráfica ello es transparente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

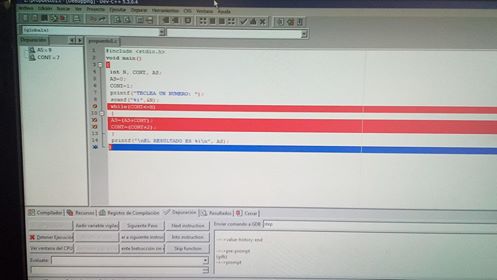
Para este ejemplo hicimos una serie de pasos para poder realizar una depuración correcta:

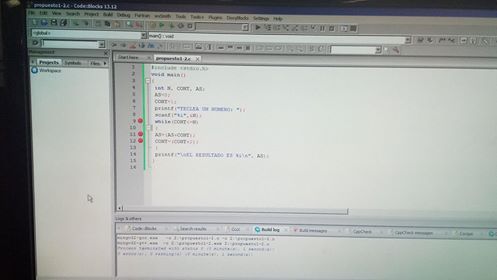
Primero lo compilamos y ejecutamos como siempre



Después pusimos los puntos de ruptura y hicimos que nos mostrara los valores conforme iba recorriendo el programa y los valores que nos iba dando para poder ver en donde estaba mal







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

Code::Blocks, es otra IDE de código abierto que puede basarse en las mismas herramientas GNU que Dev-C++. Permite cambiarse por otros motores de compilación si se desea. Proporciona un editor de texto, un compilador integrado, herramientas de depuración, etc

En este es particularmente lo mismo, únicamente lo que cambia es la forma en la que depuramos, pero fuera de eso es básicamente lo mismo.

**Ejercicios propuestos**

1) El siguiente código lo colocamos en dv.c++ para compilarlo, únicamente que este ejercicio ya lo habíamos hecho:

*#include <stdio.h>*

*void main()*

*{*

*int N, CONT, AS;*

*AS=0;*

*CONT=1;*

*printf("TECLEA UN NUMERO: ");*

*scanf("%i",&N);*

*while(CONT<=N)*

*{*

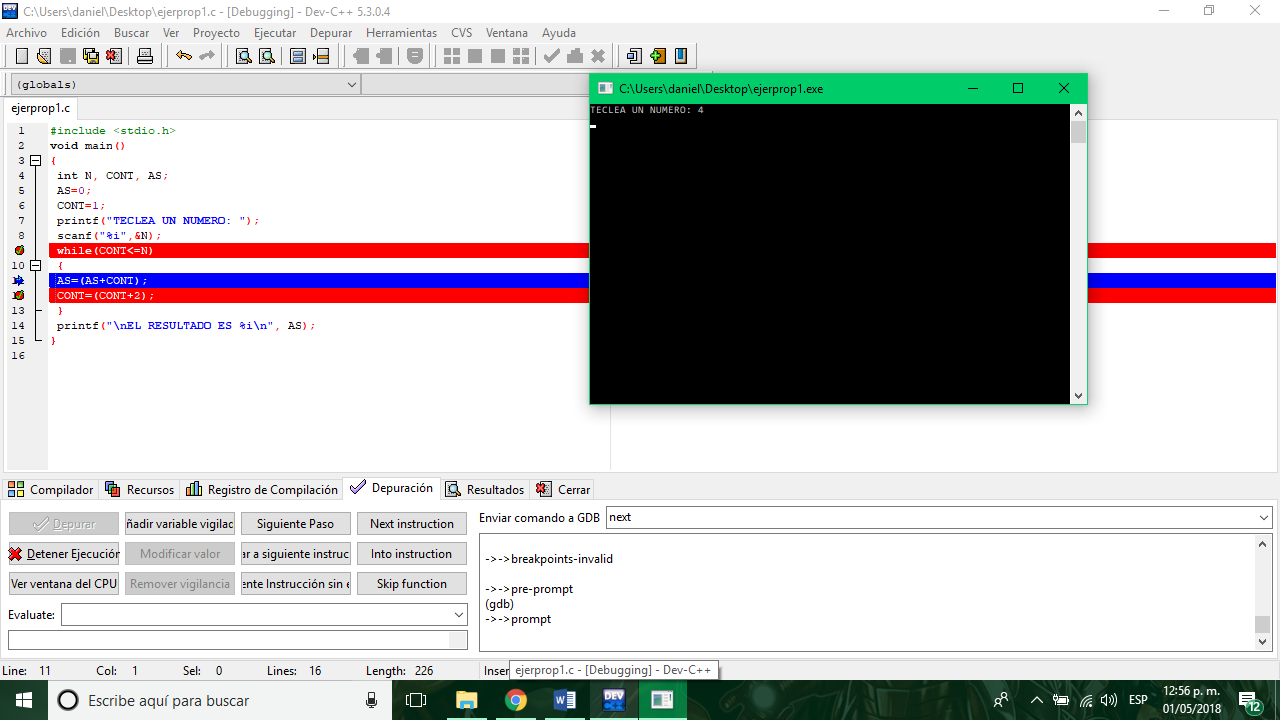
*AS=(AS+CONT);*

*CONT=(CONT+2);*

*}*

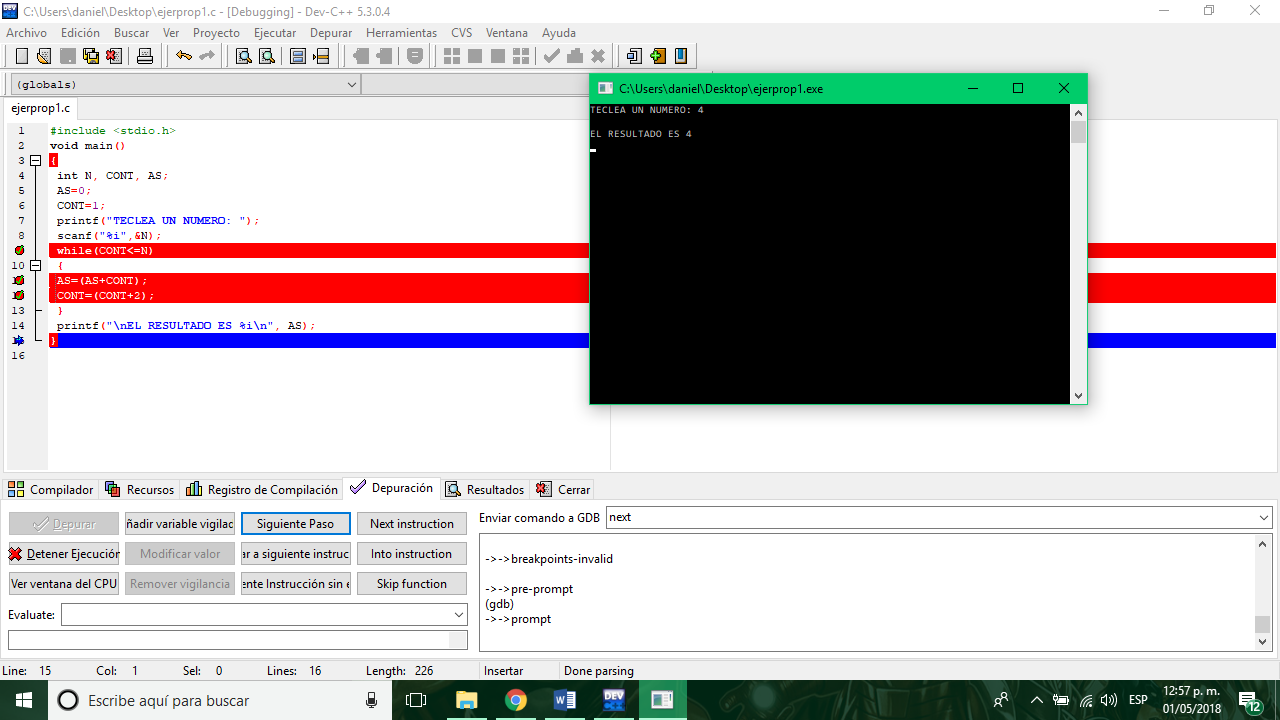
*printf("\nEL RESULTADO ES %i\n", AS);*

*}*



Como viene en los pasos me metí a herramientas y en donde decía depurador y después en esa ventana lo que hice fue darle que si generara para depurar el programa y después puse los puntos de ruptura para después lo depuramos y como ya habíamos dado cuenta utilizaba GDB como en Linux; posteriormente añadí variables de vigilancia, solo que no sé porque no se visualizaron o donde hacerle que se vean.

Al final me mostro el resultado como debía ser.



2) En el siguiente programa nos pide ver los errores con base al depurador y corregirlos

*#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++)*

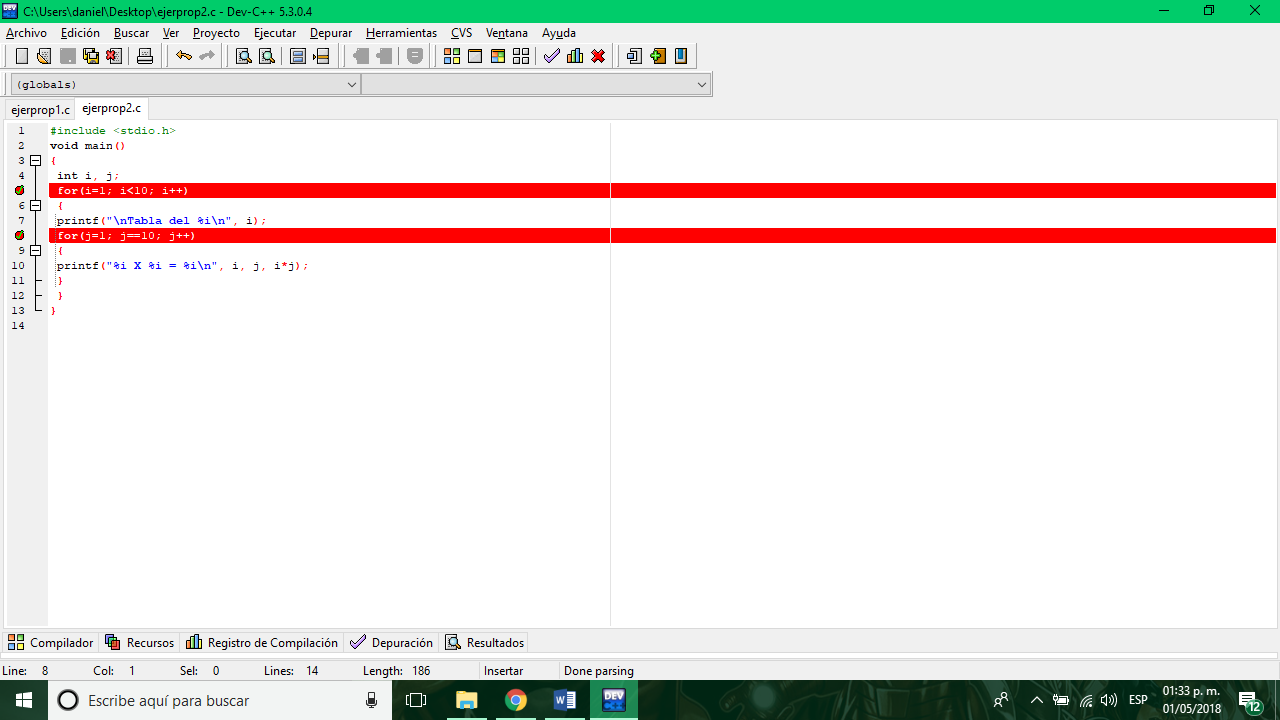
*{*

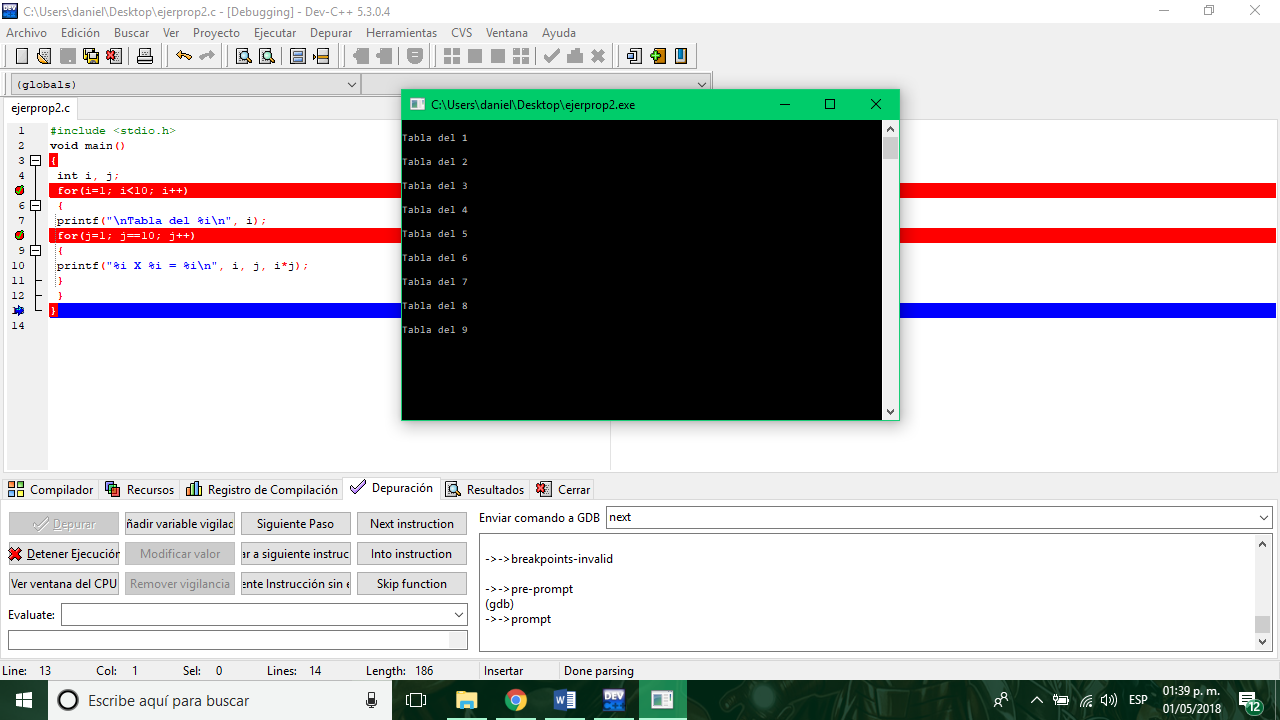
*printf("%i X %i = %i\n", i, j, i\*j);*

*}*

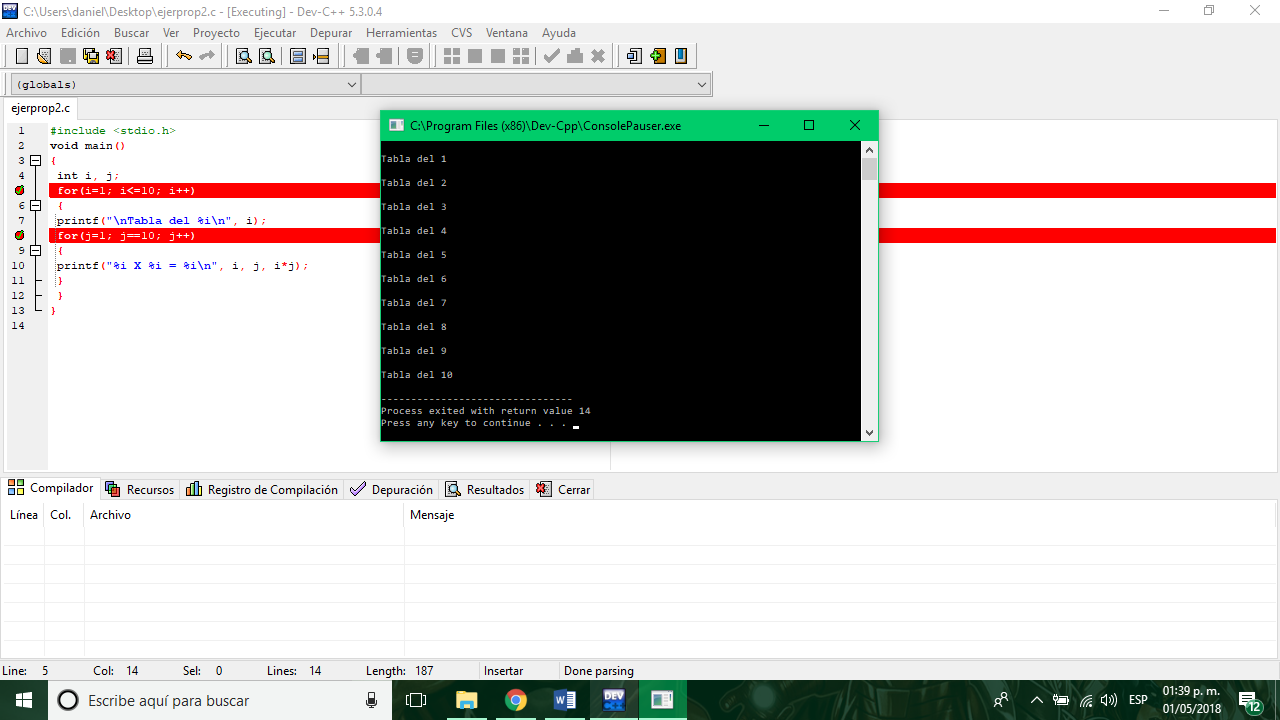
*}*

*}*

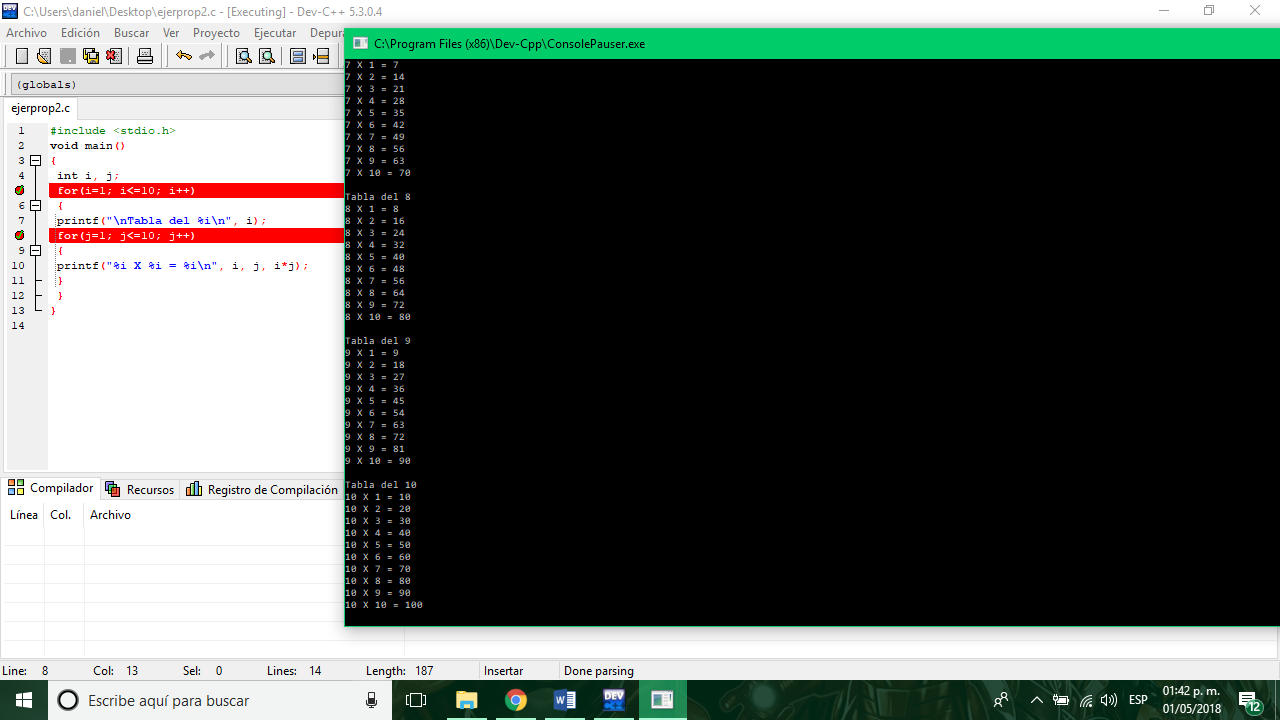




Después de depurarlo y ver el error que no nos daba la tabla del 10 lo corregí ya que la condición era <10 en vez de <=10 para qu hiciera hasta el 10 pero ahora sigue sin mostrar las tablas



Lo bueno es que como ya lo depuramos nos pudimos dar cuenta que el el segundo for no entraba ya que la condición estaba mal porque era ==10 en vez de ser igual a la anterior osea <=10 y asi pudimos ver el resultado correcto



3)

*#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;*

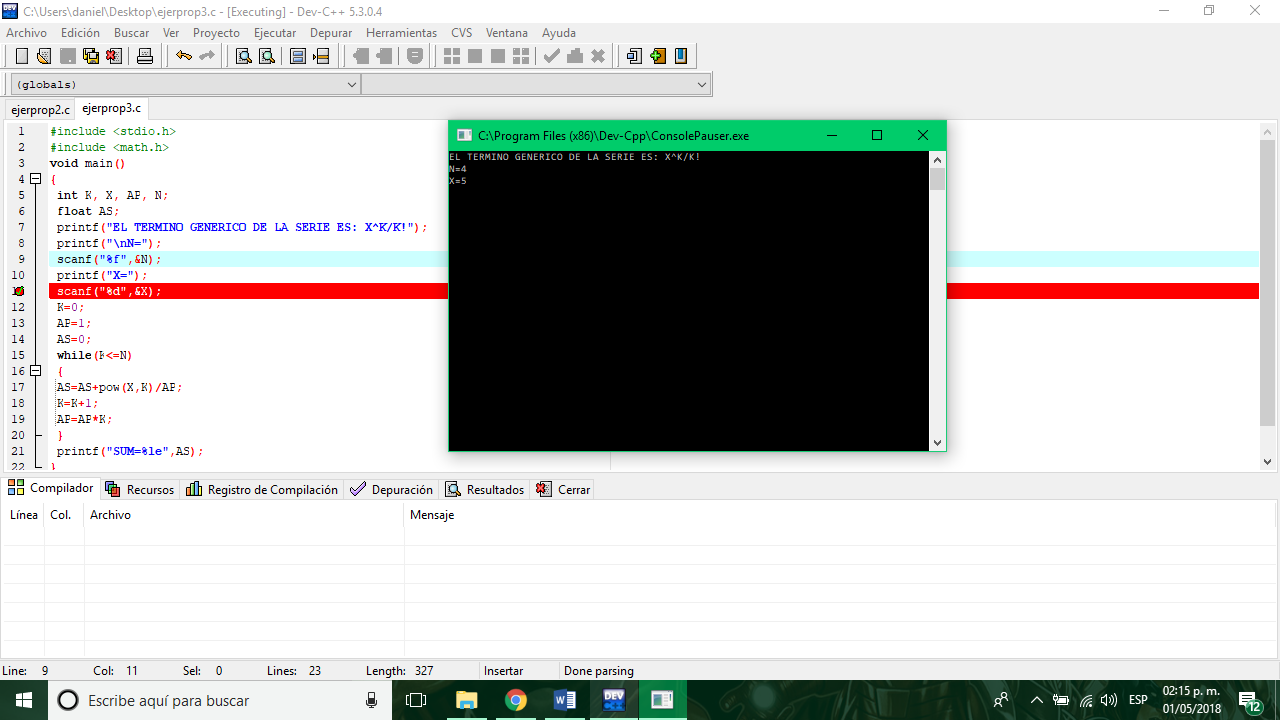
*K=K+1;*

*AP=AP\*K;*

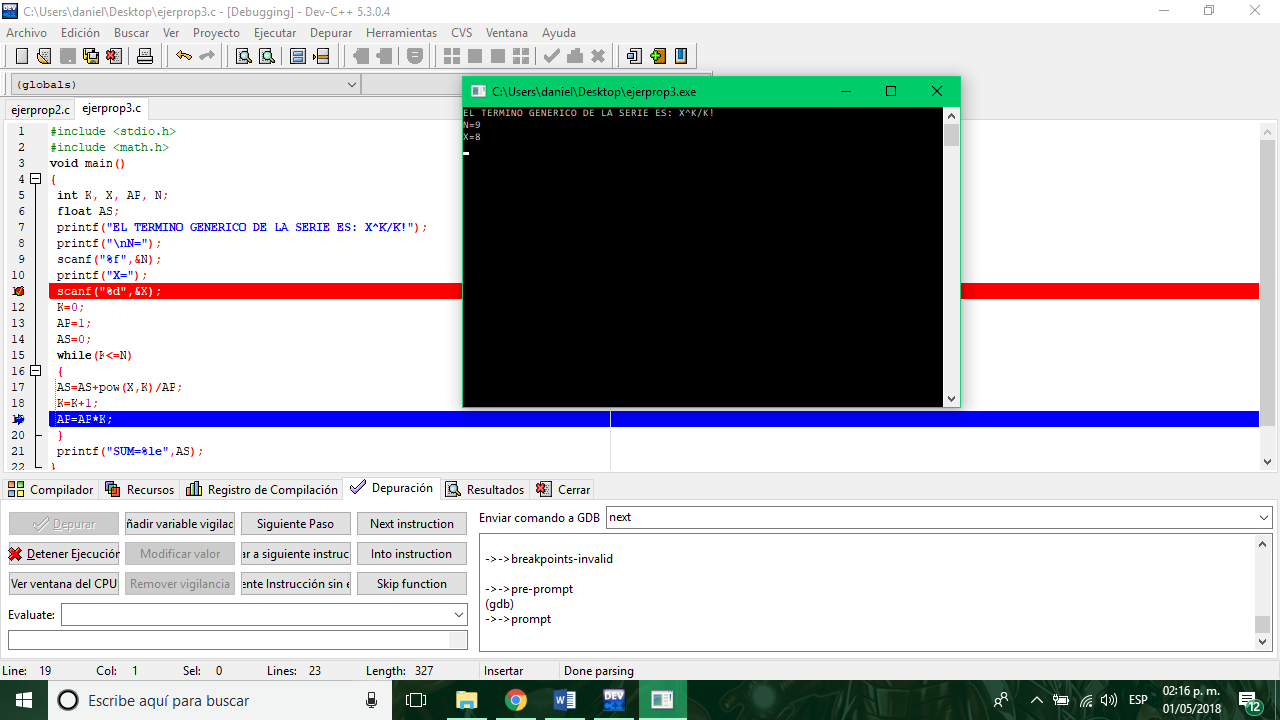
*}*

*printf("SUM=%le",AS);*

*}*

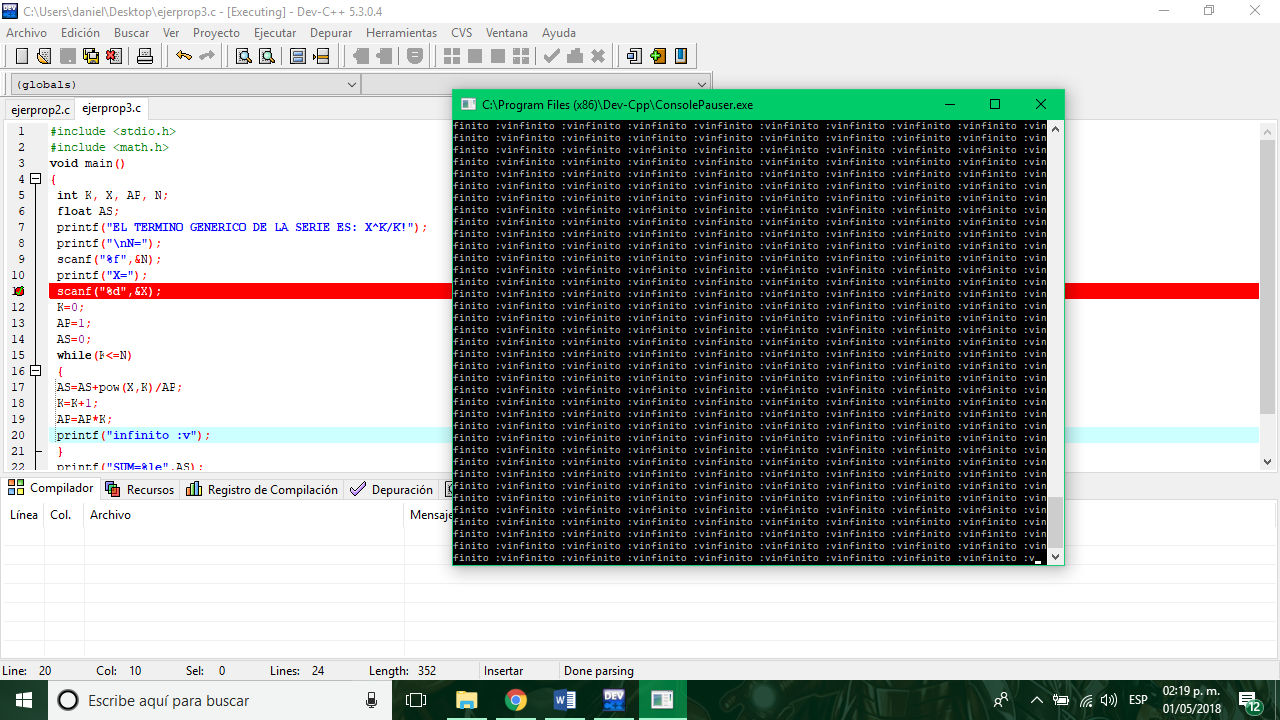


Primero lo ejecute y compile y me marca que algo estaba mal, después lo depure y asi me di cuenta que el primer scanf esta haciendo referencia a un entero en vez de un real y que en los dos scanf del principio les faltaba el “&”, entonces lo corregí pero algo sigue mal…

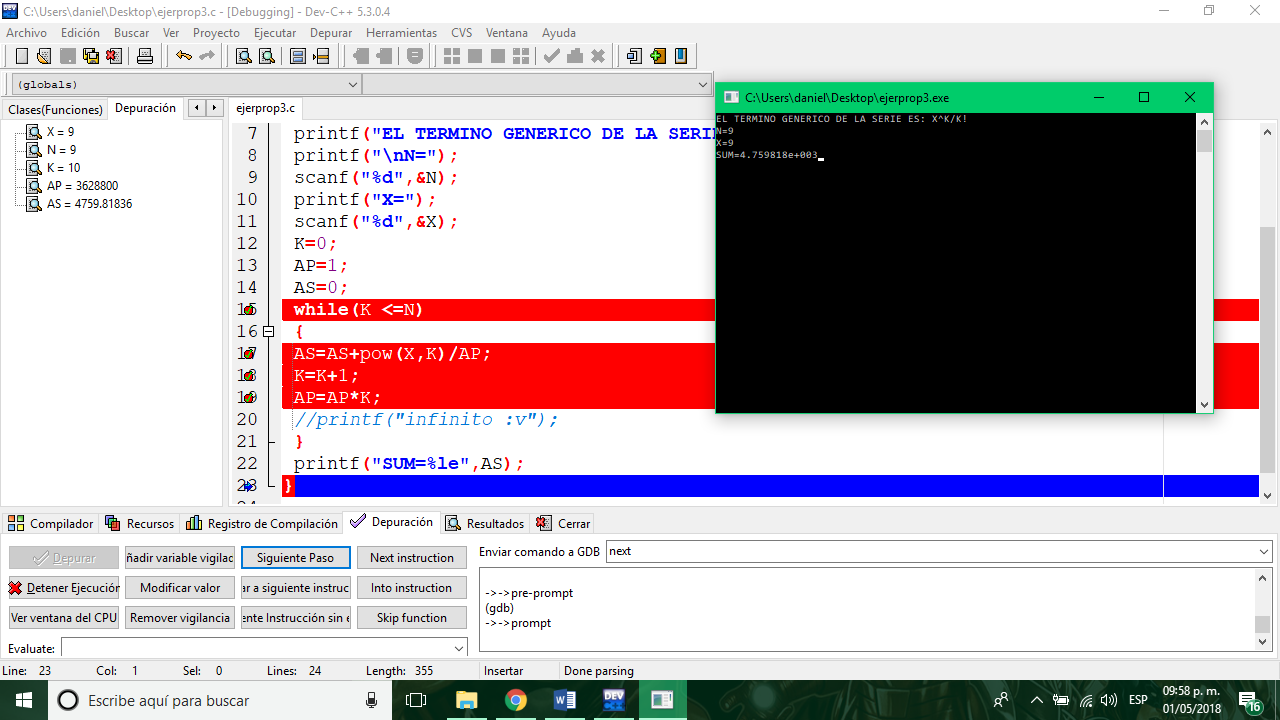


El introducir el dato sigue la secuencia del código, solo que cuando llega al while hace un siclo infinito, entonces no muestra nada porque se va a infinito.

Y para mostrarlo, coloque un printf para verificar que si se valla a infinito y asi fue:



Lo gracioso es que era un error mio en vez de ponerle %d le puse %f y eso me lanzaba un numero muy muy grande y por ello se iba a infinito y me di cuenta porque ver la variables por fin:



Propuesto:

¿Que realiza el programa?

//Depurar programa e indicar las lineas modificadas

*#include <stdio.h>*

*#define p printf*

*#define s scanf*

*main(){*

*int variableA=0,variableB,cont,cont2=0;*

*p("Ingresa el valor final");*

*s("\n%d",variableB);*

*if(variableB<=0)*

*{*

*for(cont=0;cont<10;cont++);*

*{*

*if(variableA<=variableB)*

*{*

*for(cont=0;cont=10;cont++)*

*p("\nEjecuciÃ³n numero= %d",cont+1);*

*}*

*}*

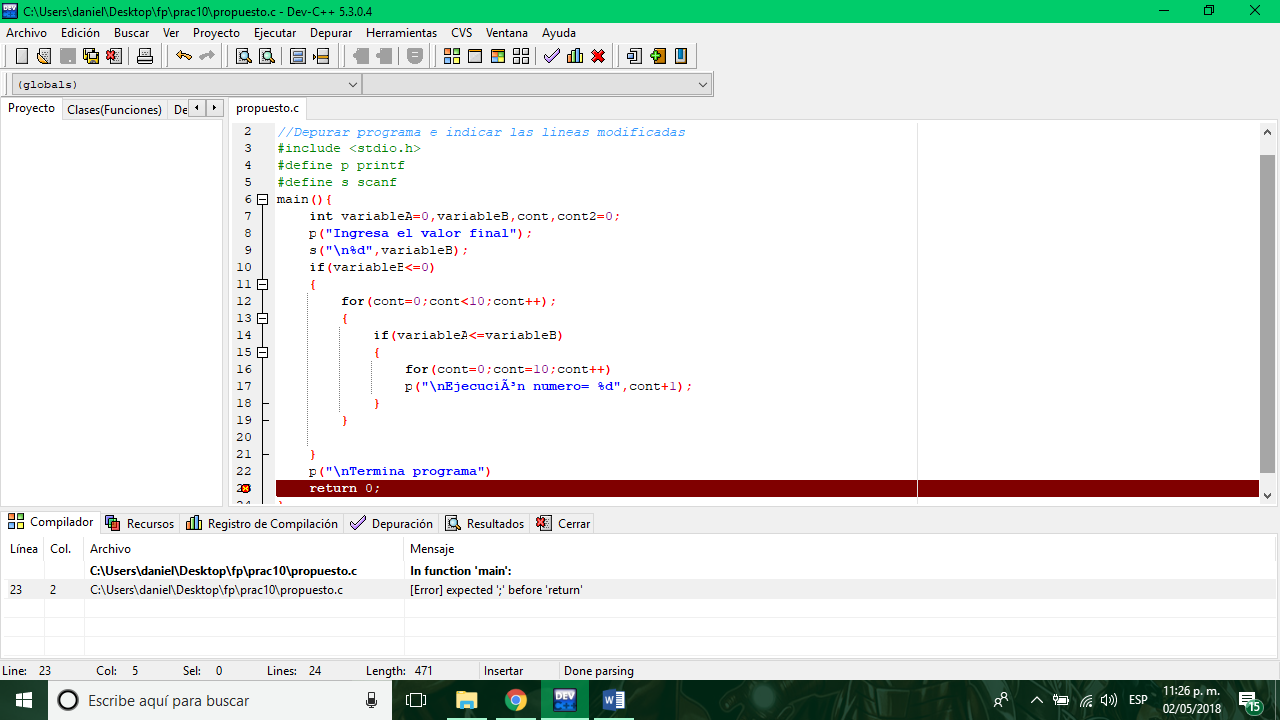
*}*

*p("\nTermina programa")*

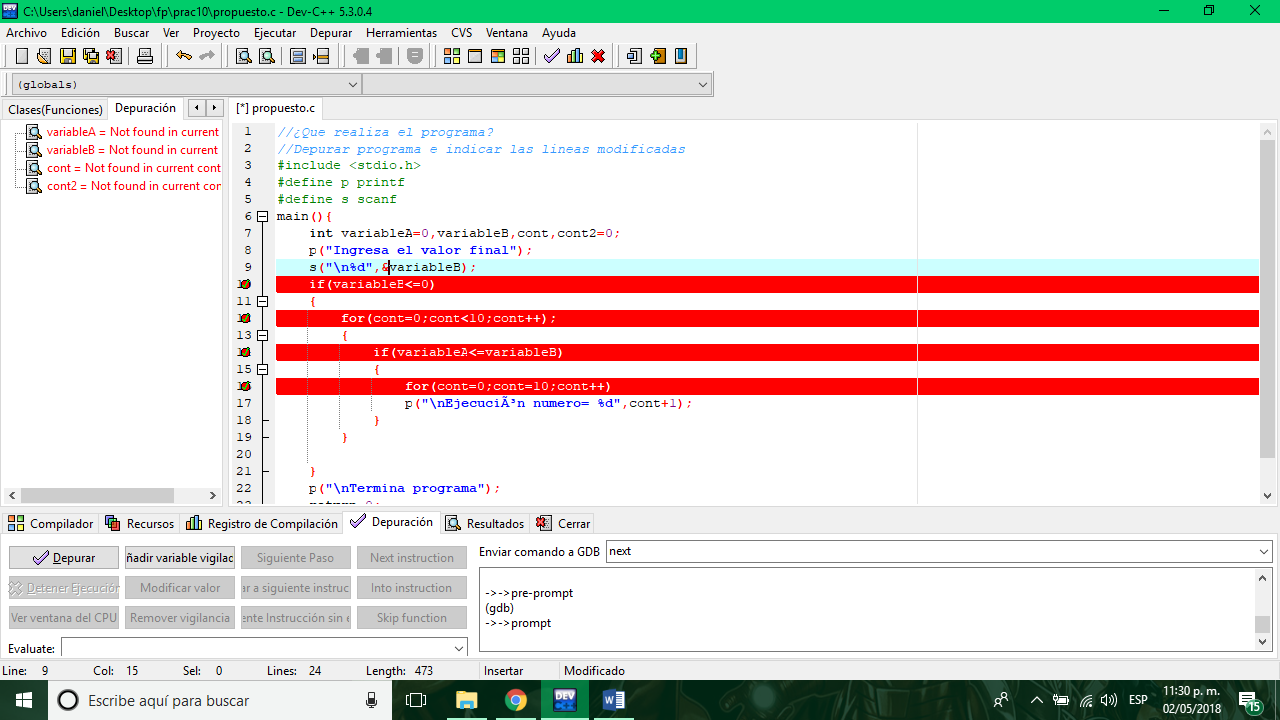
*return 0;*

*}*

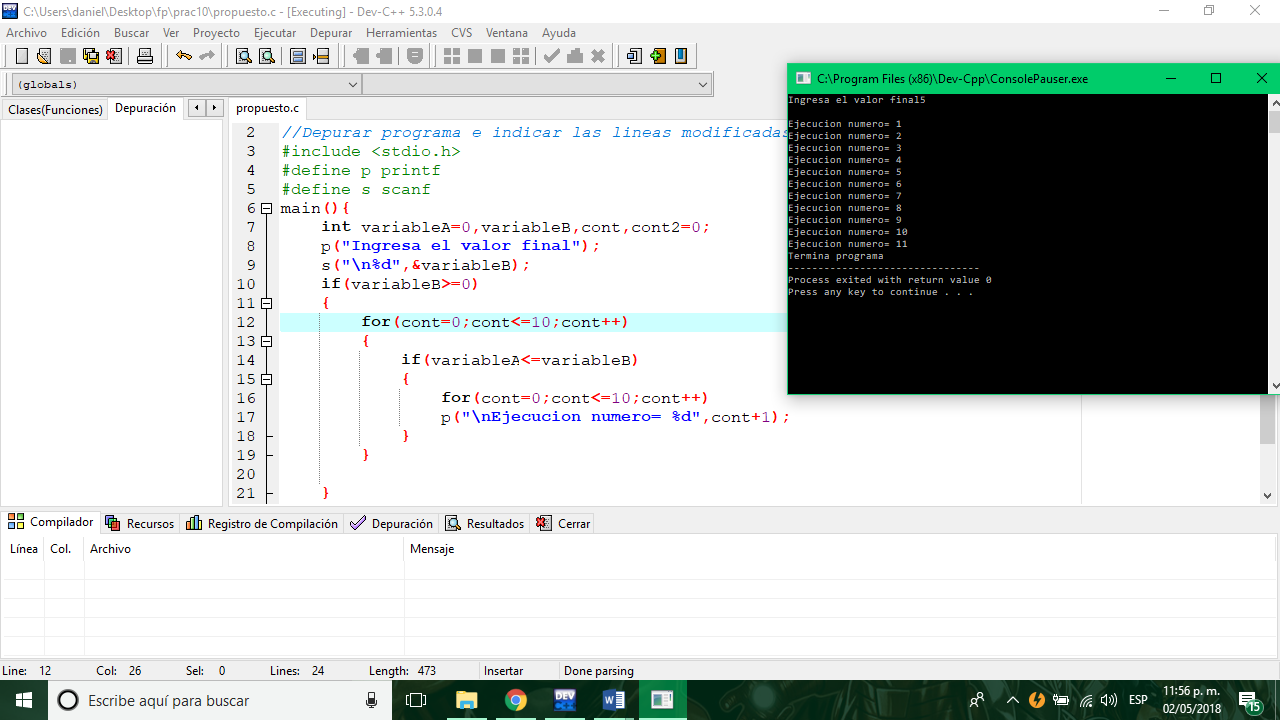
Primero lo ejecute y lo compile y me marco un error ya que antes del return le faltaba un ;



Ahora al depurarlo nos damos cuenta que en el primer scanf le faltaba un & junto la variableB



Después el segundo for tiene un ; que no va y en el 2 y en el ultimo for en ves de solo llevar solo < o = le colocamos ambas para que funcionara correctamente



Según lo que hace es que si un valor es menor mayor o igual a 0 hace un conteo pero creo que sigue mal

Conclusión: Podemos concluir que gracias a la herramienta depuración podemos ver otro tipi de errores como lógicos u otros, que no se llegaran a ver a simple vista y así podemos obtener un programa más eficiente y que cumpla con lo que necesitamos.

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>
* 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/depuracionde-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>
* <http://lcp02.fi-b.unam.mx/>