|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Nombre de la práctica** | **Prácticas en C++** | | | **No.** |  |
| **Asignatura:** | **Métodos Numéricos** | **Carrera:** | **ISIC** | **Duración de la práctica (Hrs)** |  |

**Emiliano arce de Jesús Gpo.3402**

1. **Competencias especificas**

**II. Lugar de realización de la práctica (laboratorio, taller, aula u otro):**

1. **Material empleado:**

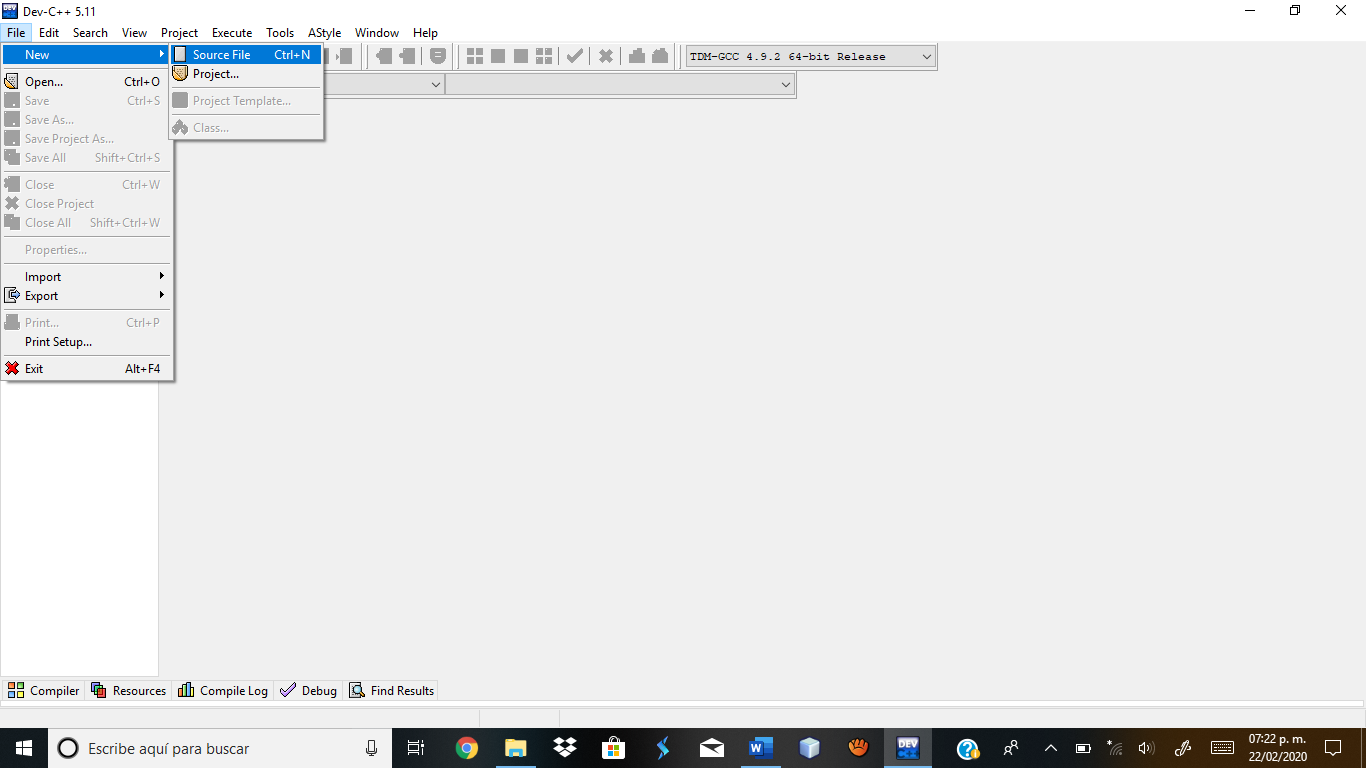
* **Lenguaje de programación estructurada Dev-C++**
* **Manuel de prácticas FO-aca**

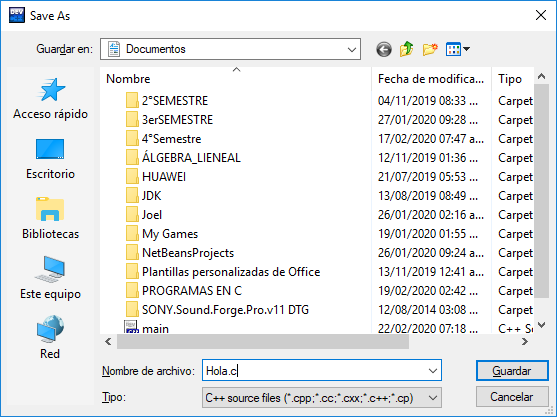
1. **Desarrollo de la práctica:**

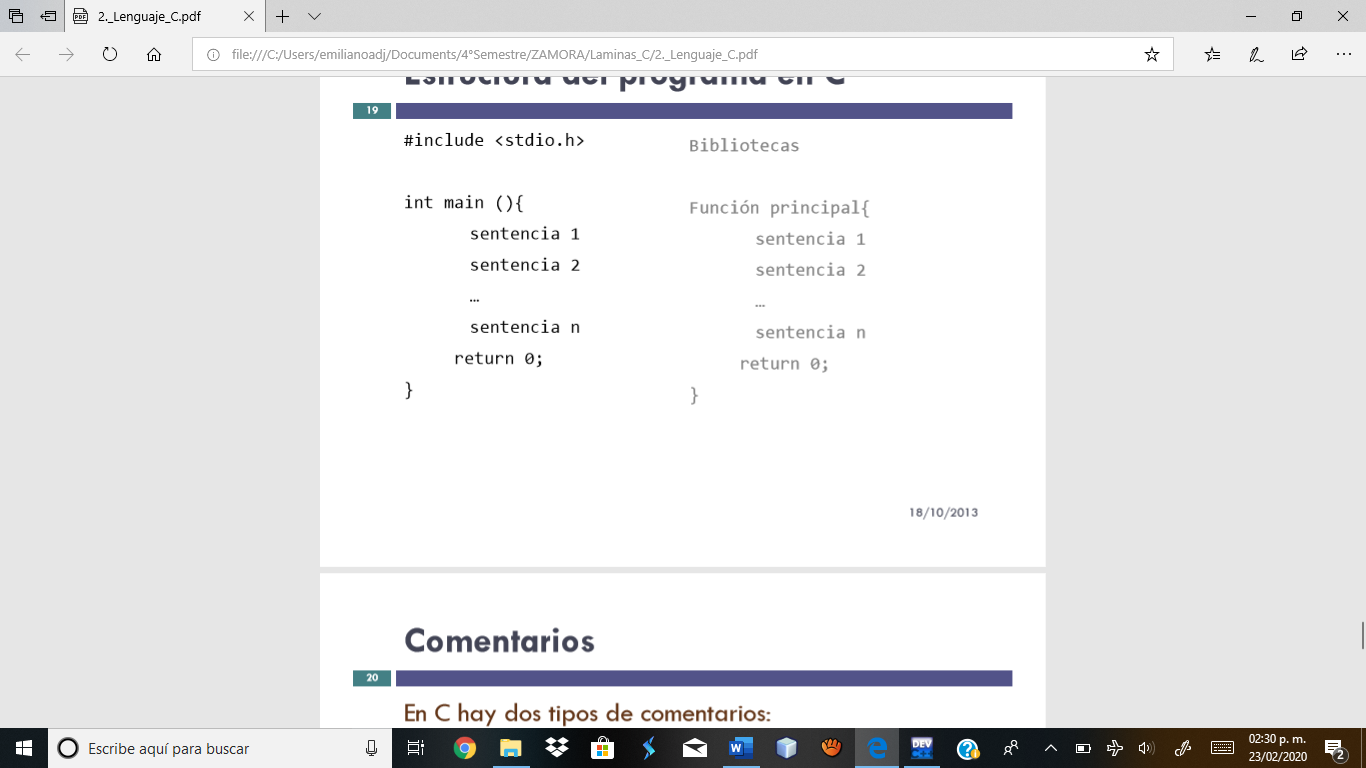
**Comenzando a programar en C++ nos encontramos con un entorno diferente pero no con un cambio total al que ya estábamos acostumbrados (Java) y podemos notar que su entendimiento y manipulación para escribir programas no esta del todo diferente a los demás lenguajes de programación.**

**Comenzaremos por la creación de un nuevo archivo. Para ello vamos a realizar los siguientes pasos:**

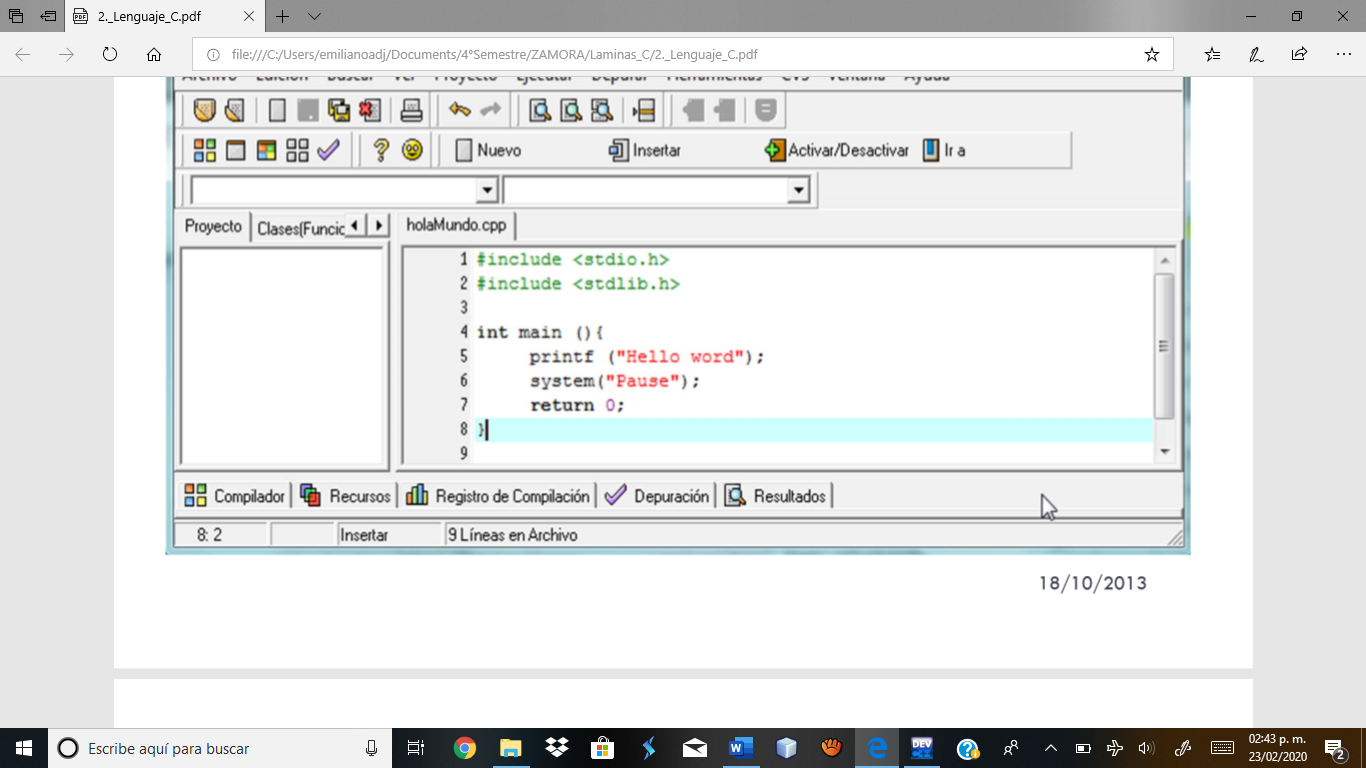
**Entrar a Dev-C++, pulsar en la pestaña File/ new/ Source File.**

****

**Después nos iremos a la pestaña File y pulsaremos en “Save all” y escribiremos el nombre de nuestro programa, en este caso se llamara “Hola.c”.**

**La estructura de un programa en C es la siguiente:**

**#include: Incluye las declaraciones de otro fichero de compilación.**

* **El siguiente código es un ejemplo claro de la estructura.**

**#include <stdio.h> : que significa (estándar de entrada y salida). Contiene macro definiciones, constantes, y las declaraciones de funciones y tipos usados para la entrada de varios estándar y operaciones de salida.**

**#include <stdlib.h> : (standard library o estándar de librería). Contiene los prototipos de funciones de C para gestión de memoria dinámica, control de procesos y otras.**

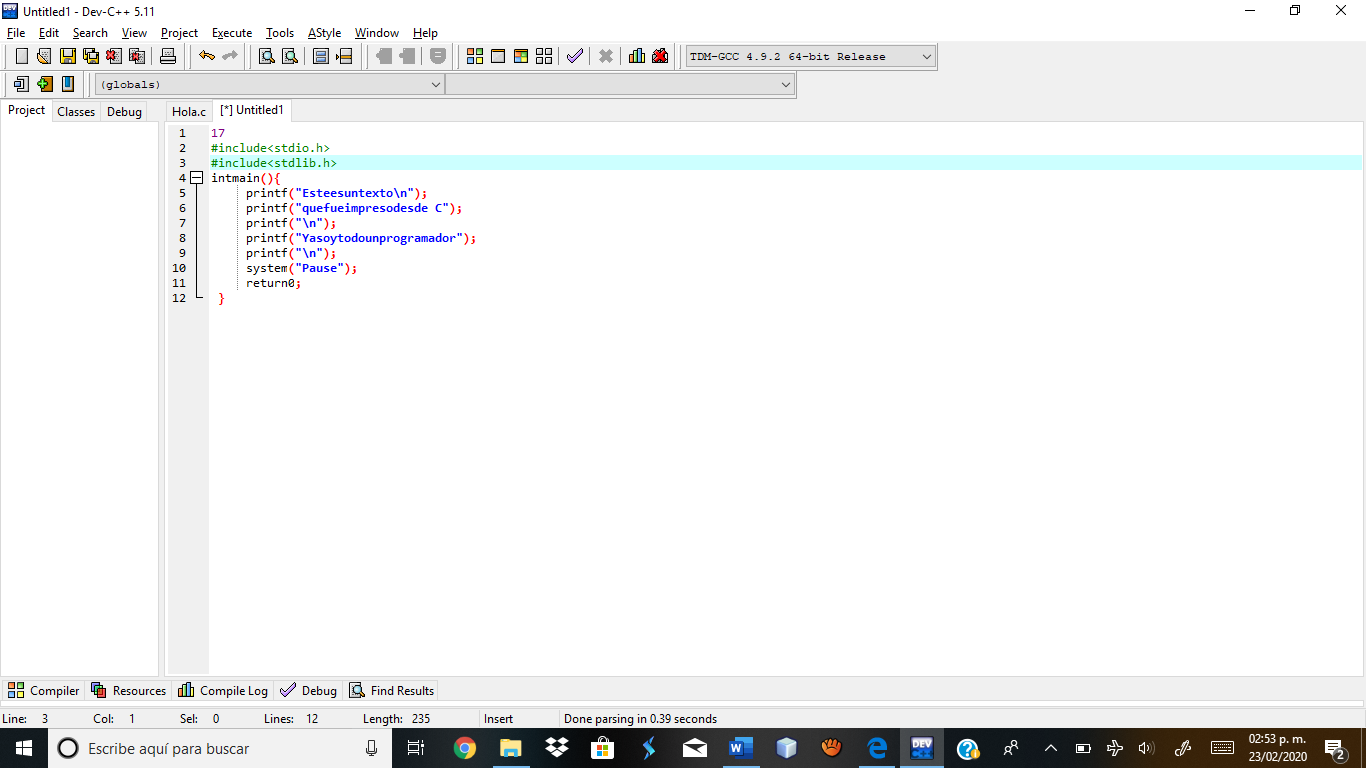
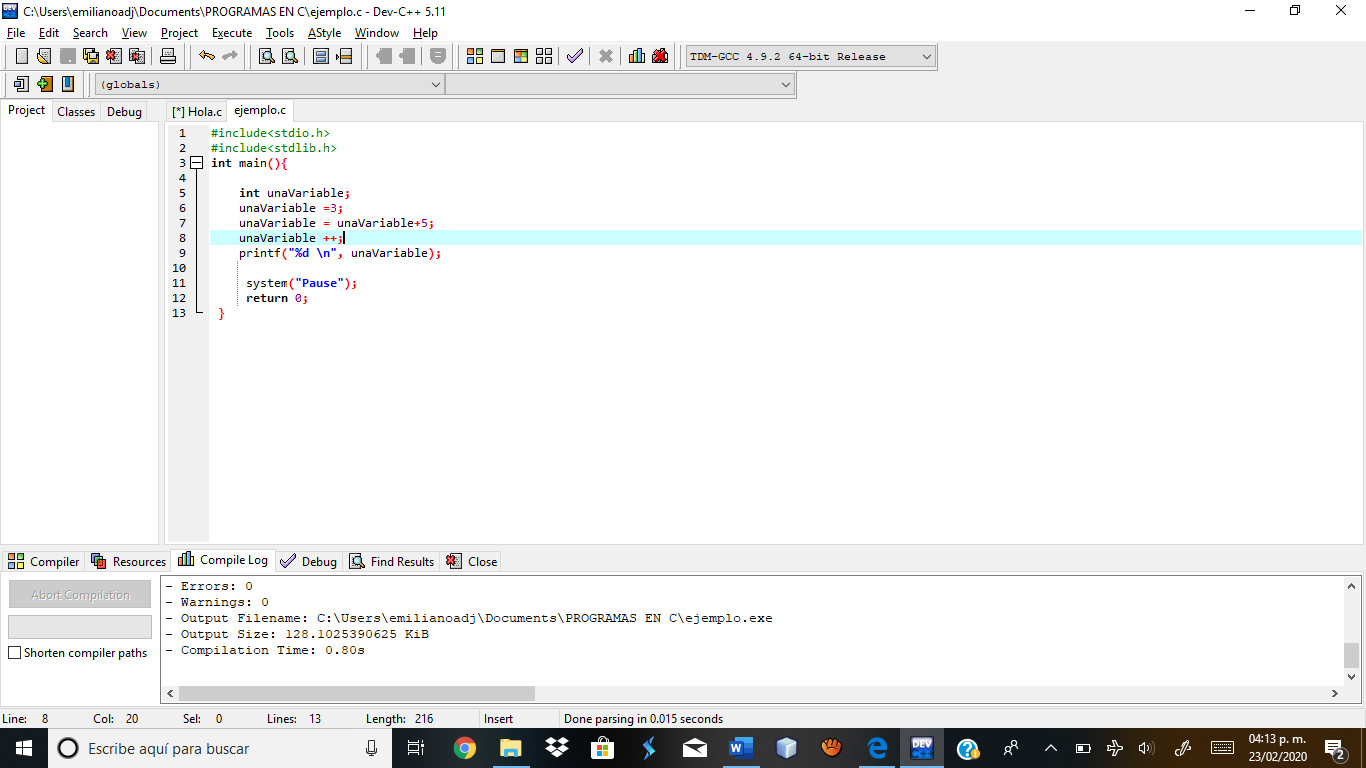
**Int main (){} : es el método principal que retornara un entero (int).**

**printf : (print formated). Sirve para imprimir en pantalla una cadena de caracteres con formato.**

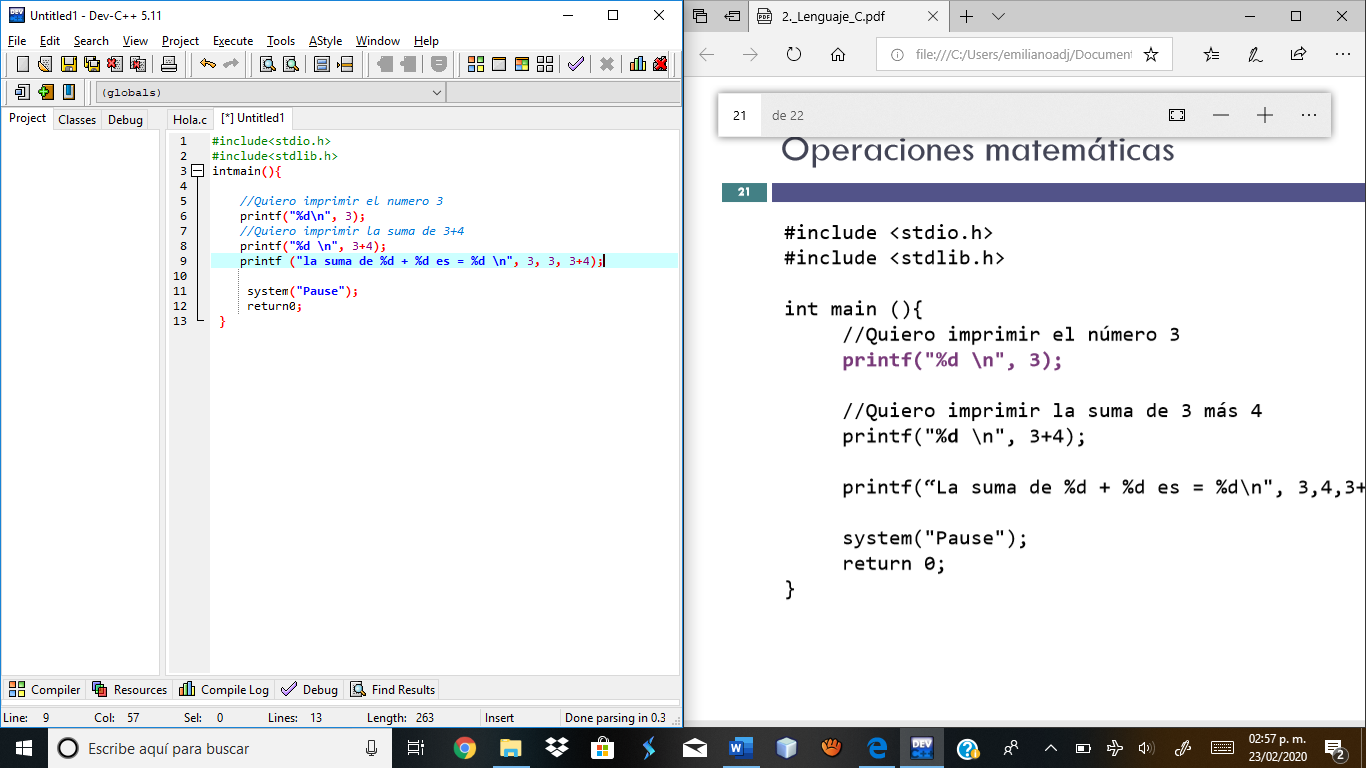
**system ("Pause"); : se esta incorporando una pausa en el programa , es decir, espera a que el usuario presione una tecla para continuar con la ejecución del mismo.**

**return 0; : devuelve un valor entero al método.**

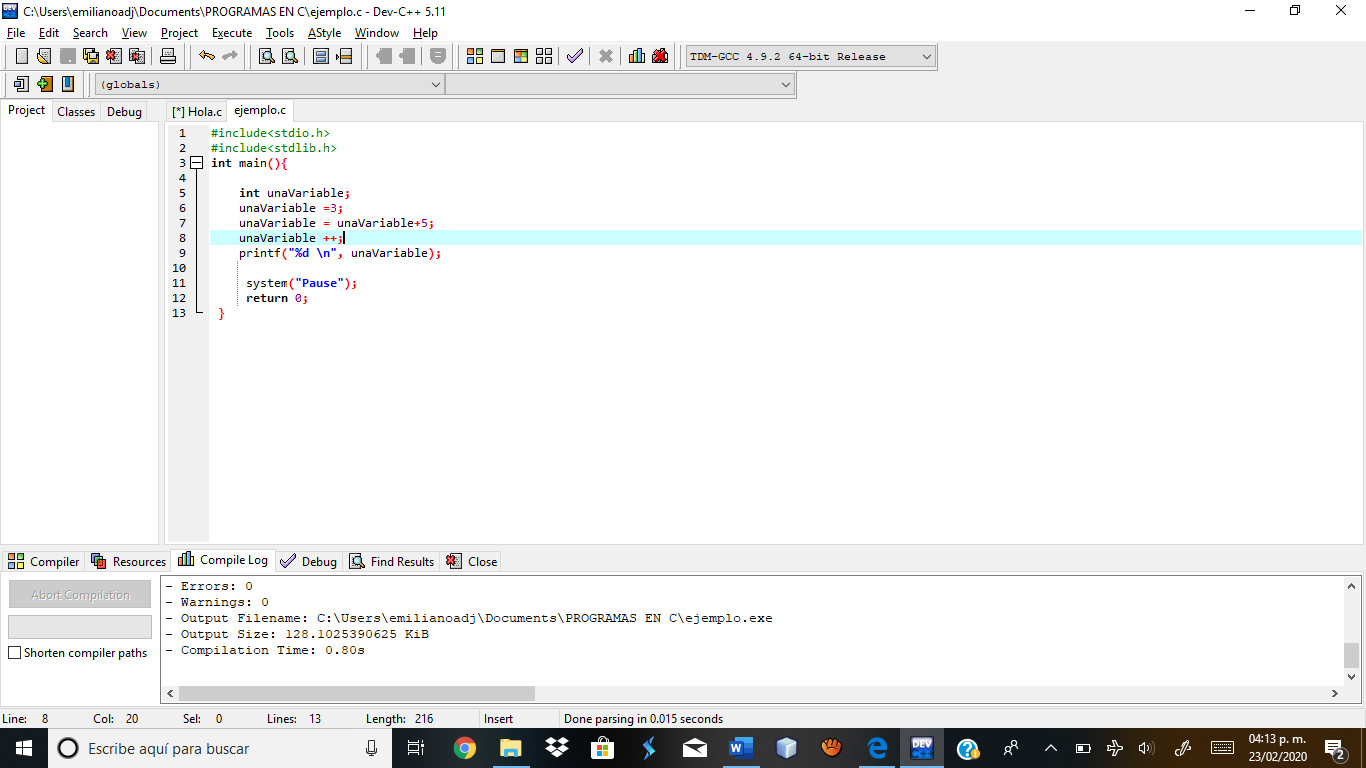
* **Ahora veremos un ejemplo más amplio.**

****

**“\n” Salto de línea.**

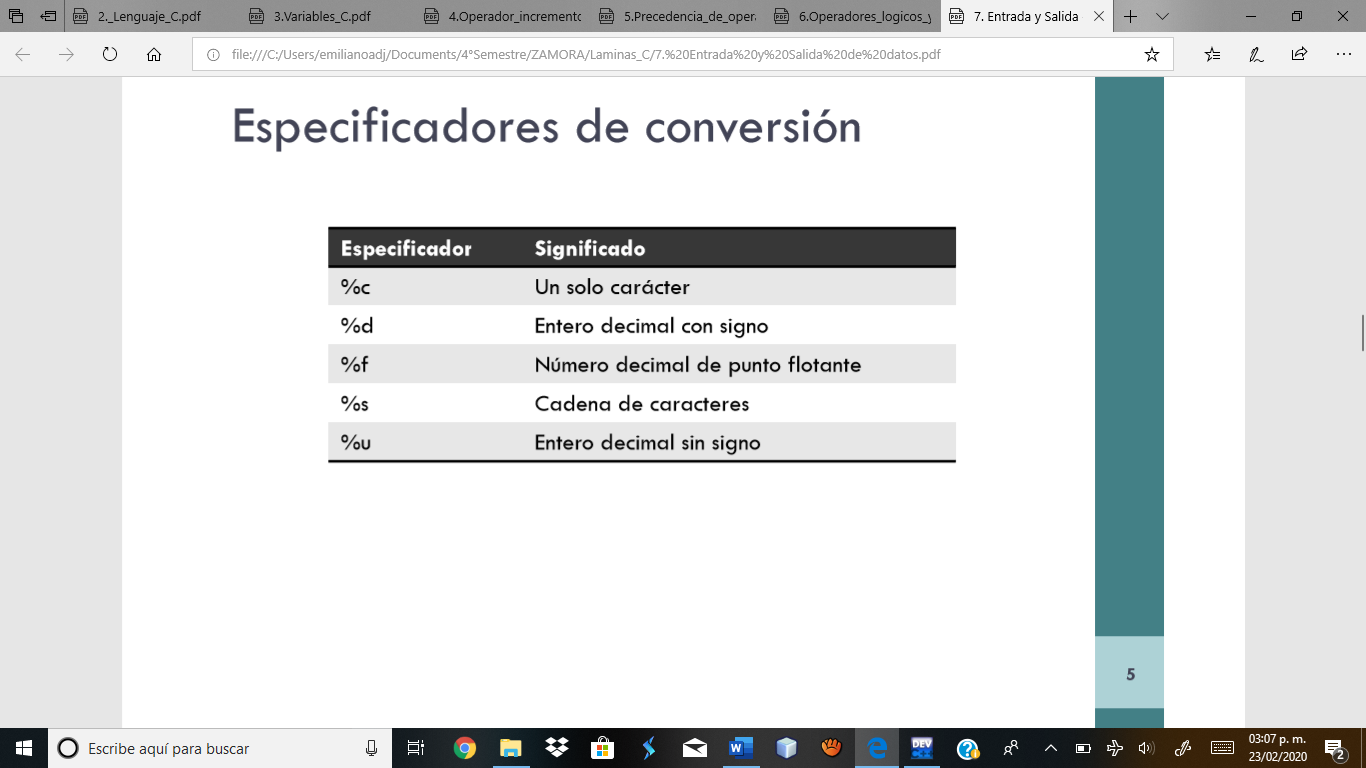
* **Operaciones matemáticas.**

\*\*\*Un especificador de conversión o de formato es un carácter o caracteres especiales que indican como se tratará una variable por su tipo de dato. \*\*\*

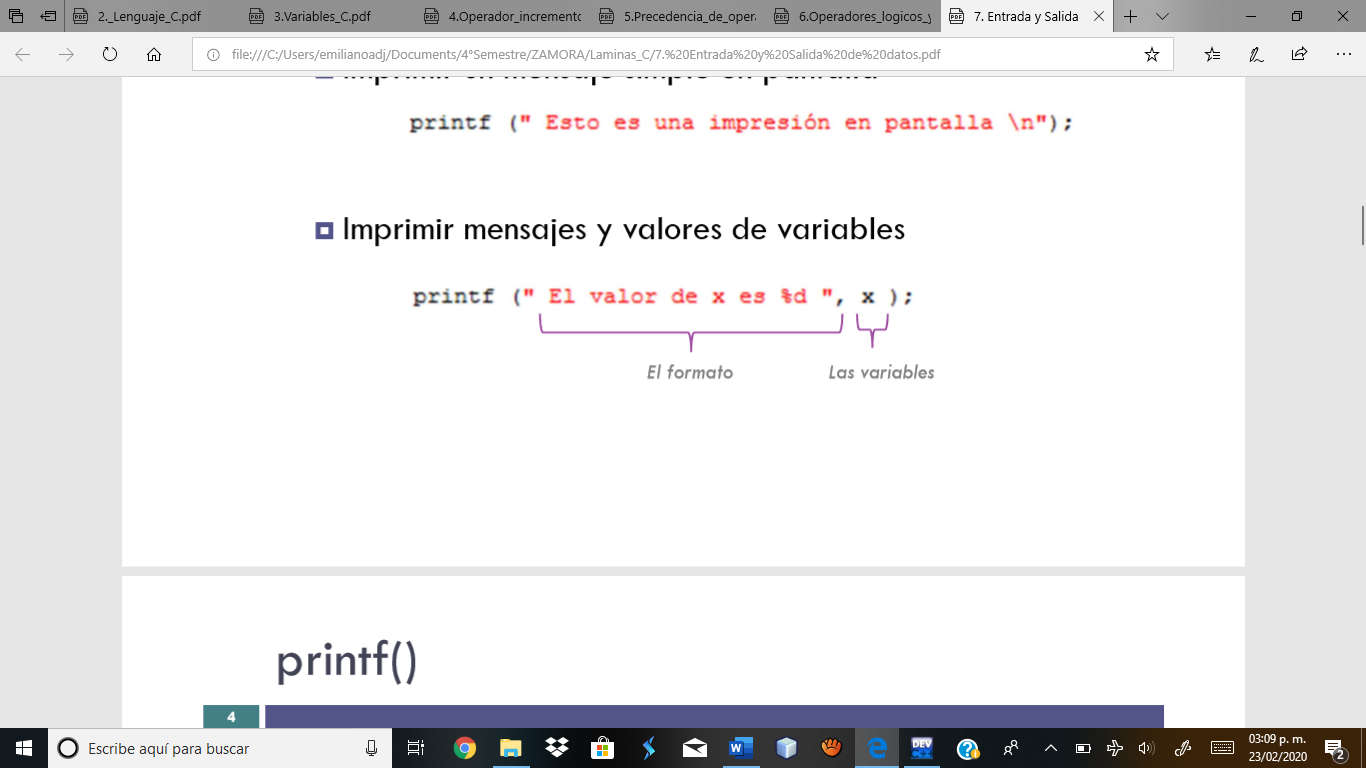
****

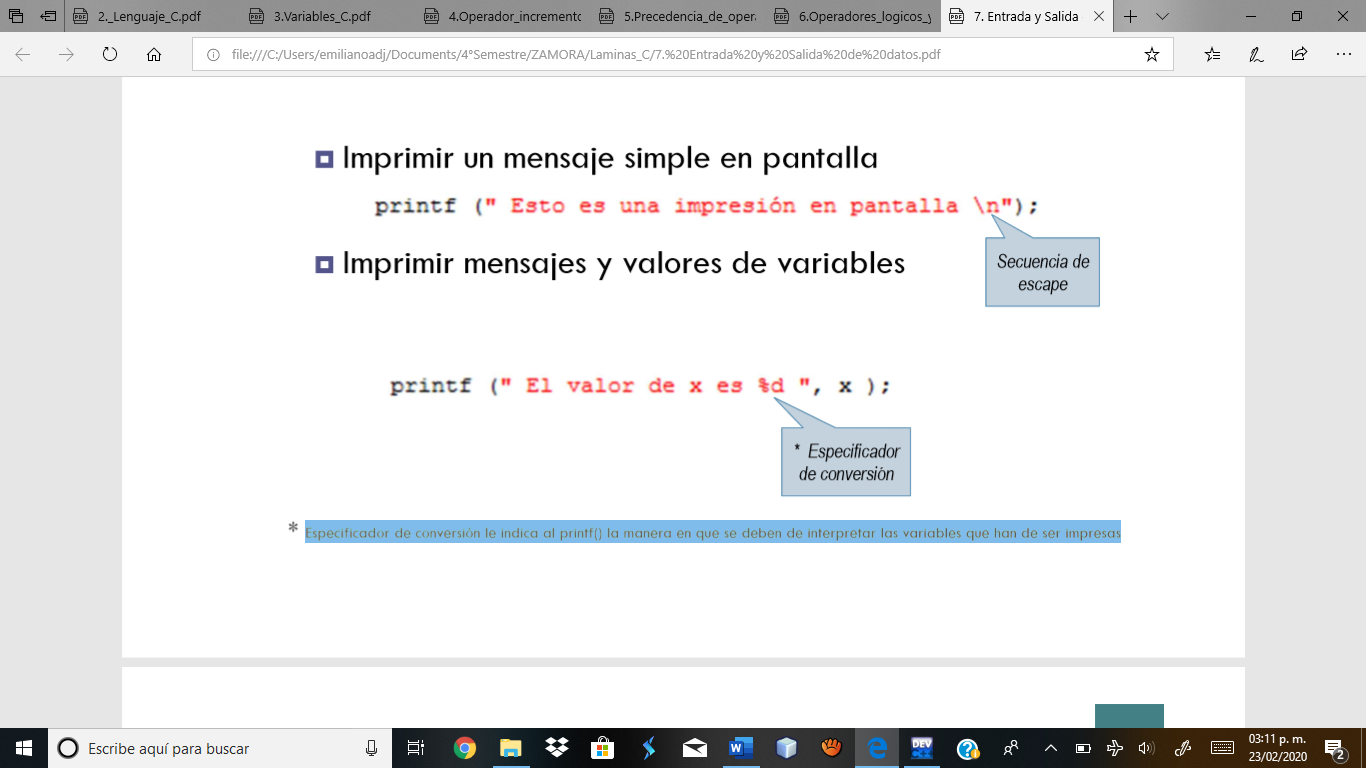
**En la linea 9 hace uso de operadores aritmeticos (suma +) ademas de que utiliza tres especificadores de conversion (%f).**

**Especificadores de conversión.**

****

**Imprimir mensajes y valores de variables.**

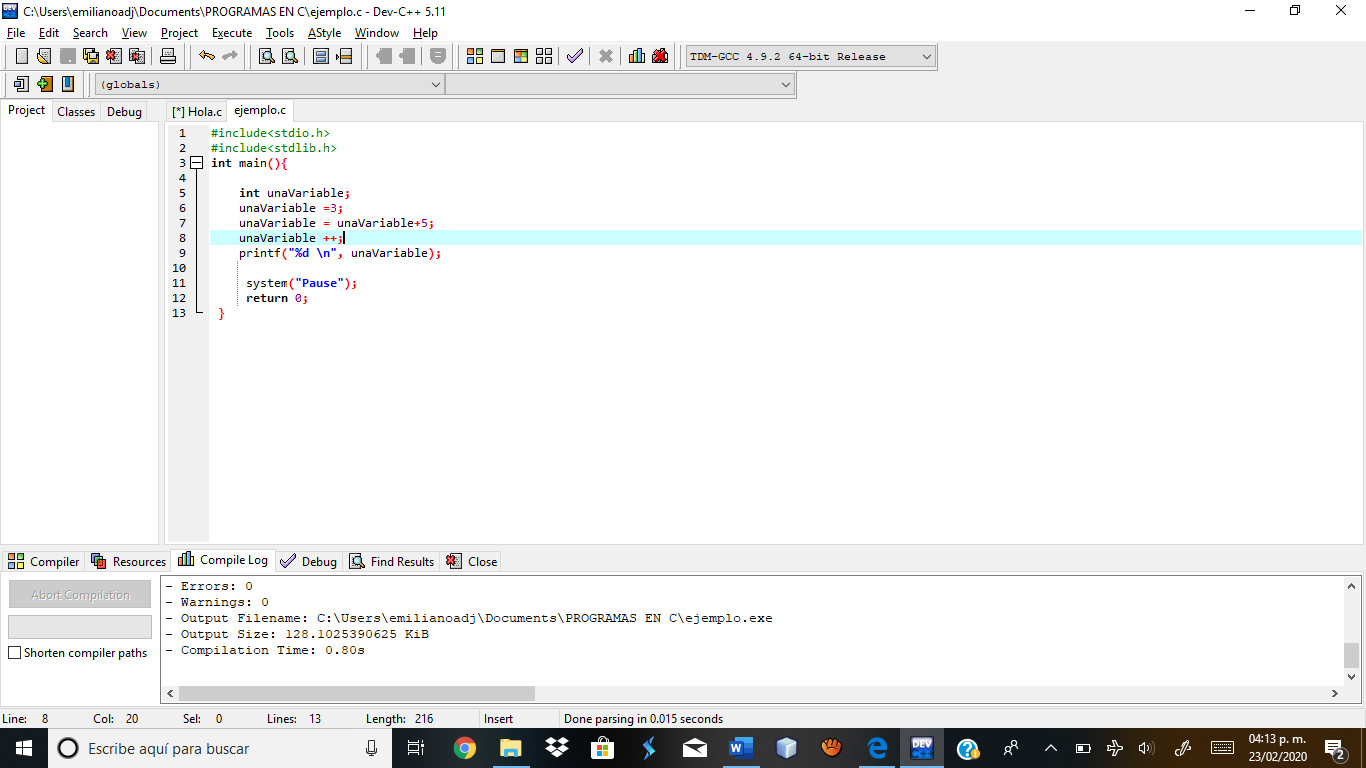
****

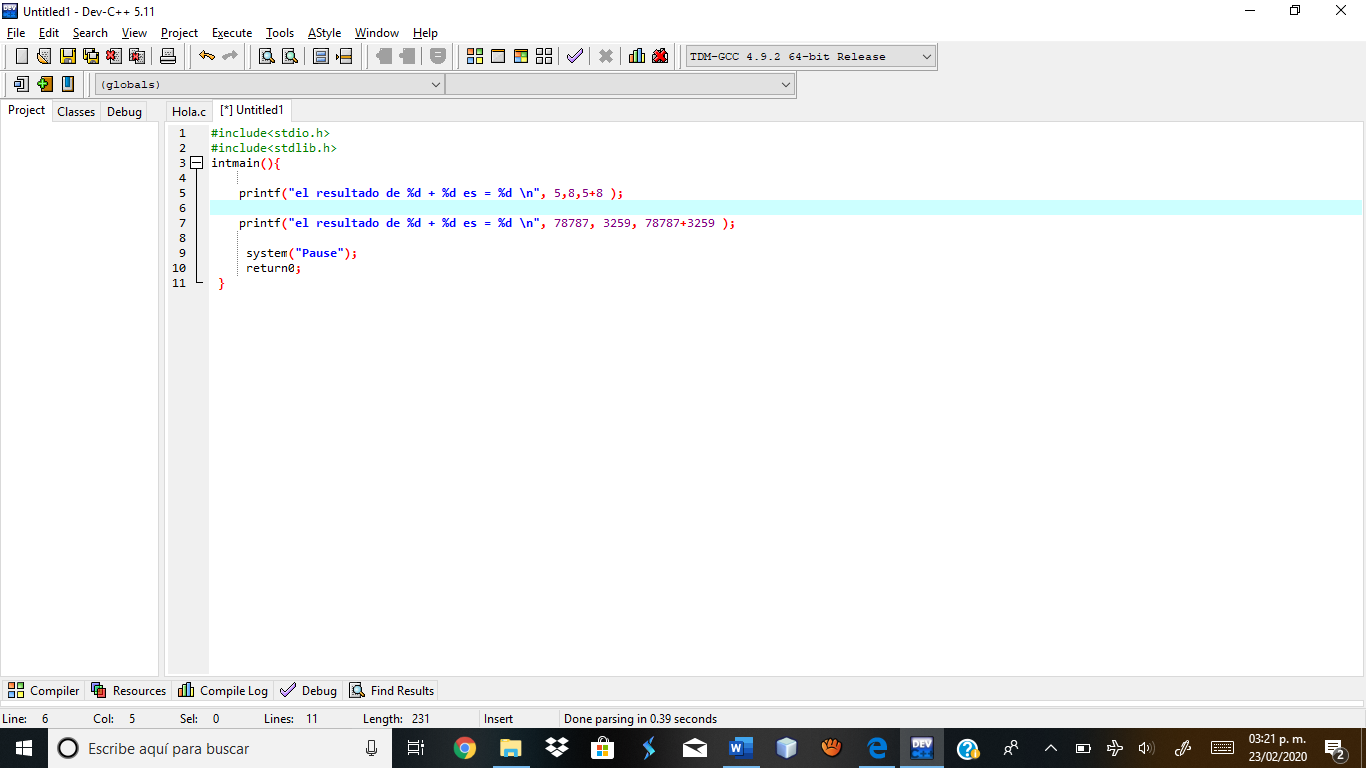
****

**Imprimir mensajes y valores de variables.**

**Imprimir un mensaje simple en pantalla.**

**Presentaremos algunos ejemplos para observar el funcionamiento de lo antes mencionado.**

* **En el método principal escribir el código para la suma de: 5+8 y 78787+3259.**

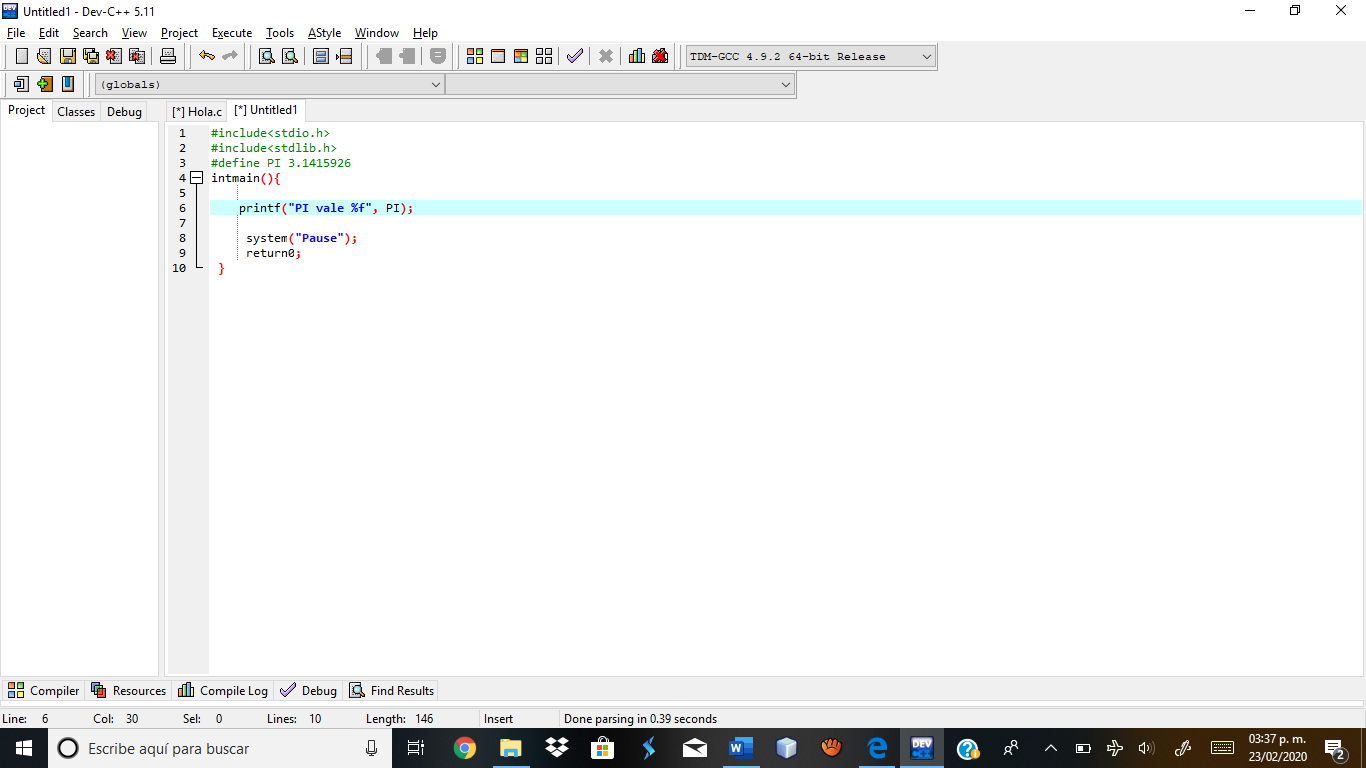
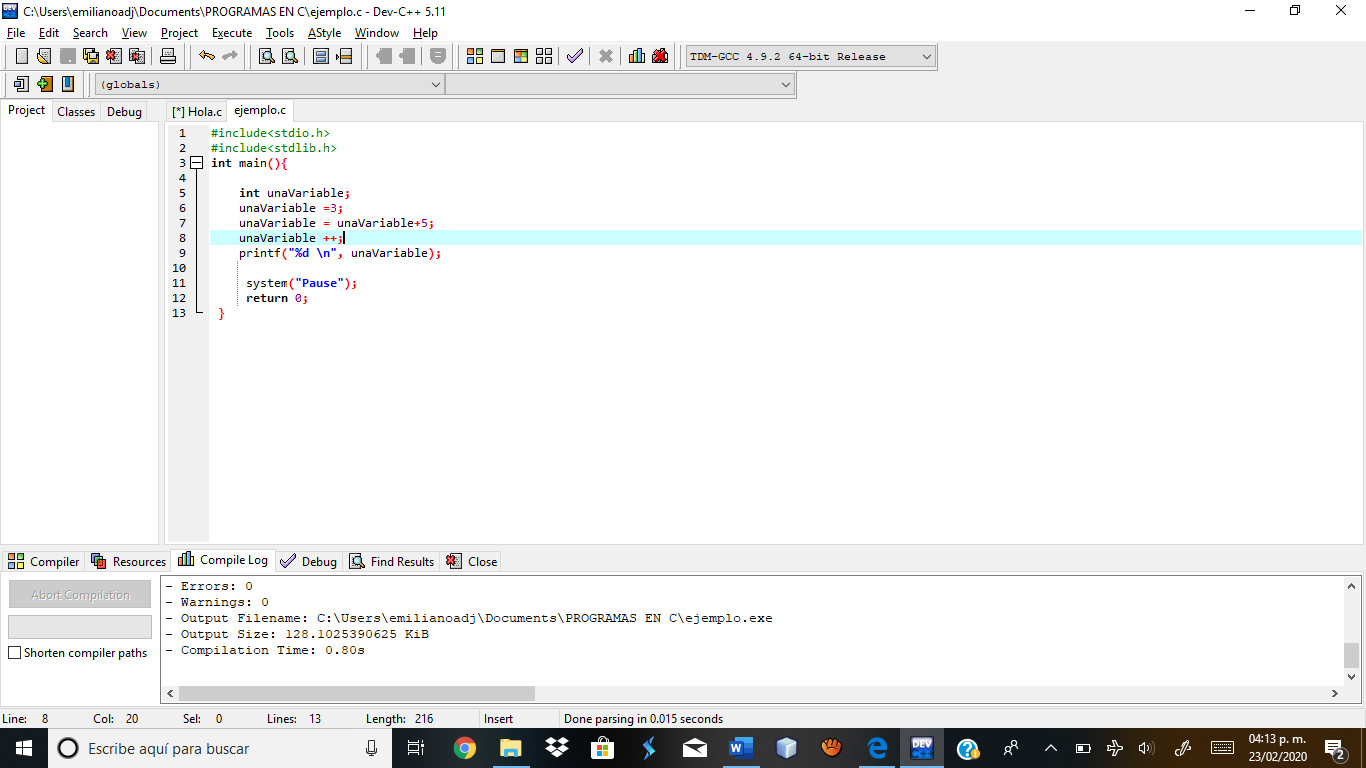
**Este código realiza dos sumas utilizando tipos de dato entero.**

**Cuando utilizamos varios especificadores de conversión, vamos a separarlos por una coma, es decir, si tenemos una suma de 2+2 tendremos que escribir:**

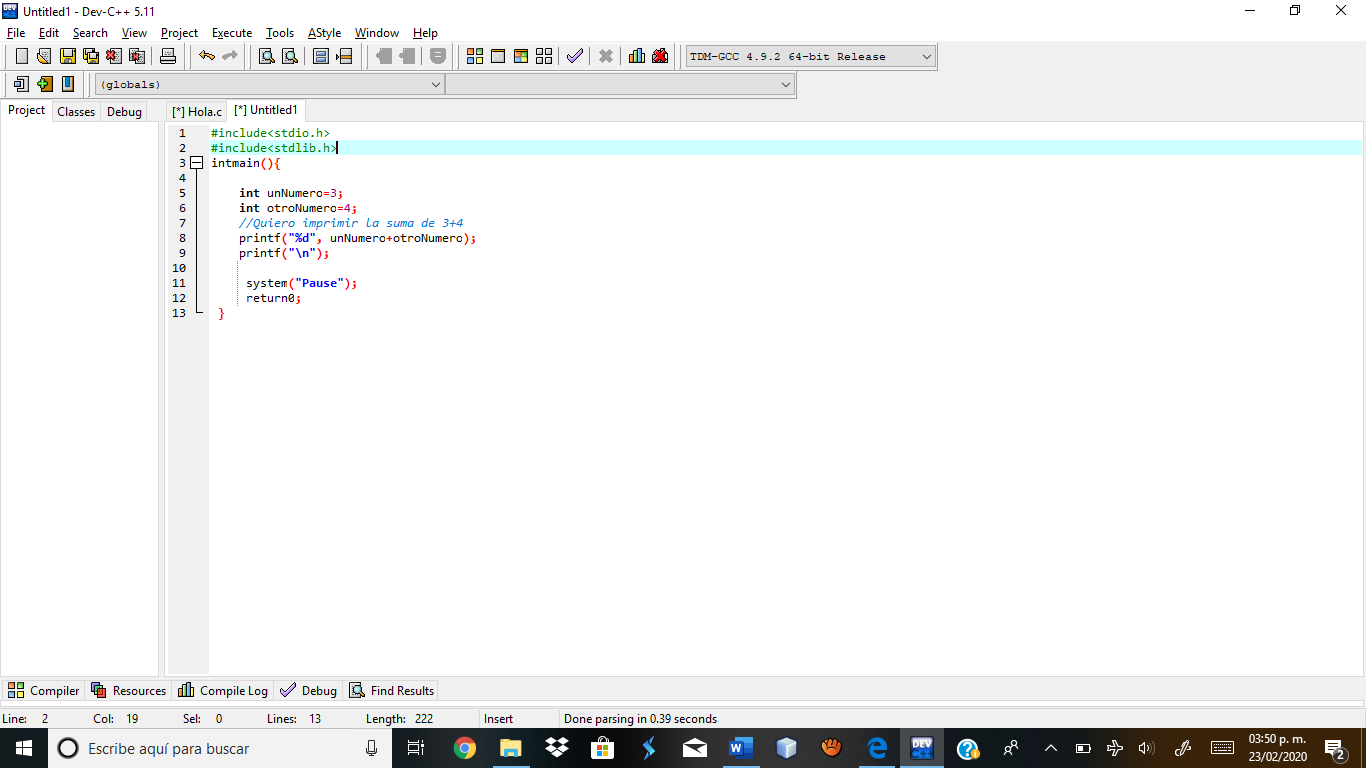
**(“la suma de %d + %d es: ”, 2, 2, 2+2);**

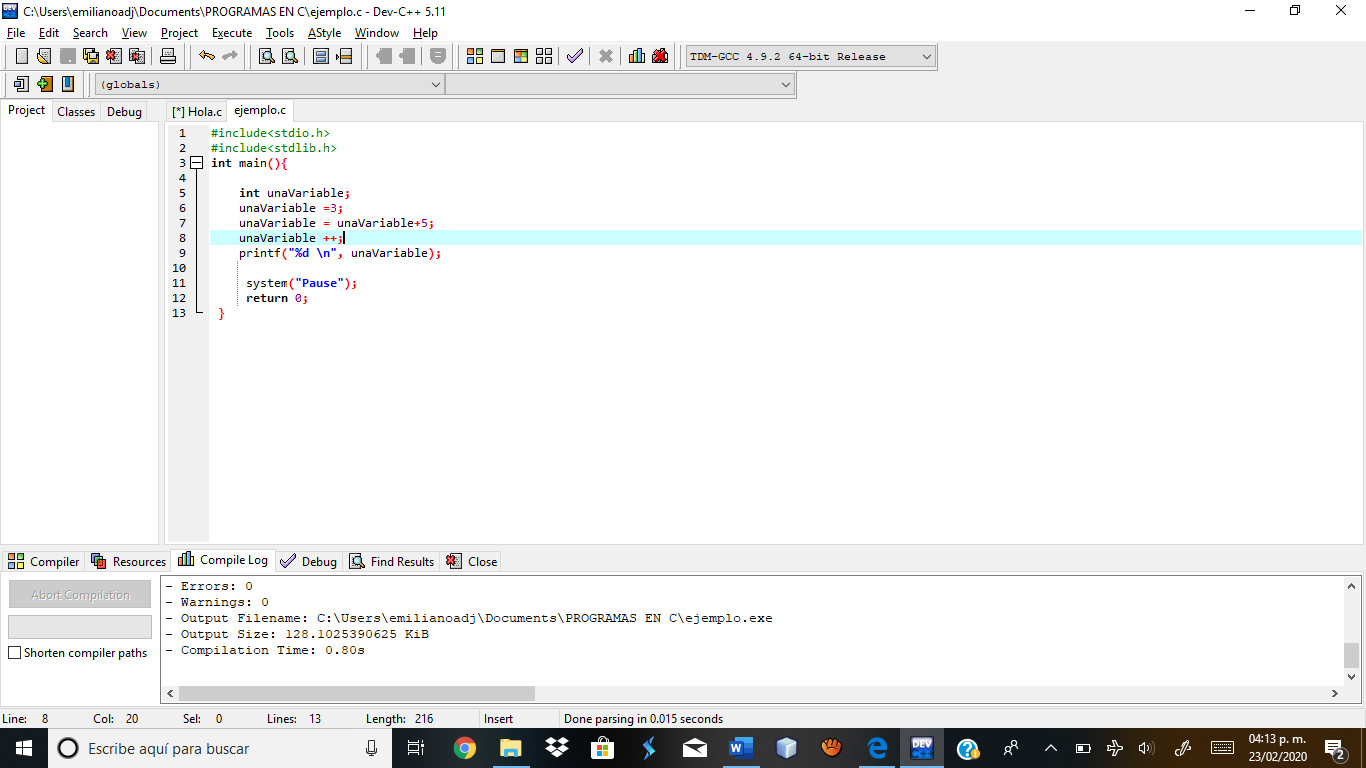
**Como se muestra en la imagen de la izquierda. (línea 5 y 7).**

* **Lenguaje C nos permite definir constantes con valor numérico por medio de macros.**

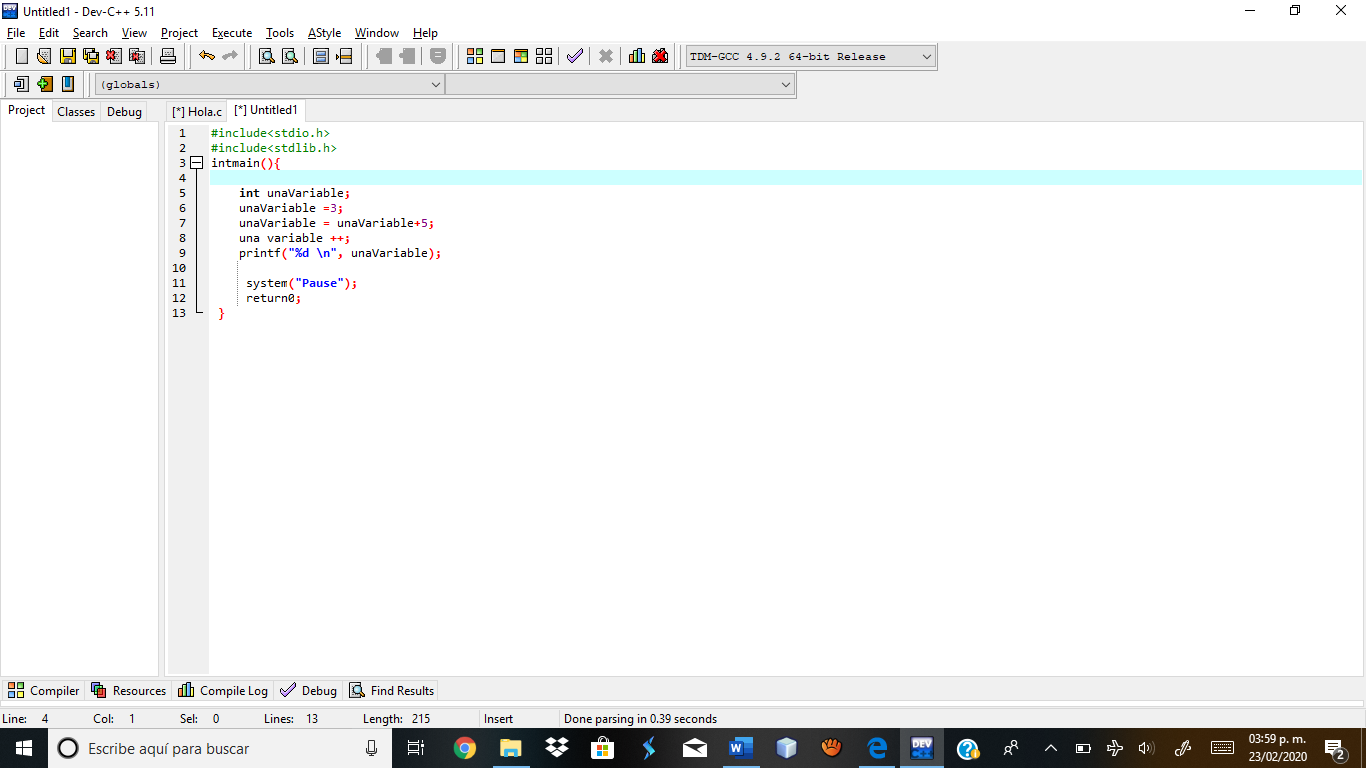
**PI es definida como constante, por lo tanto se pueden realizar operaciones con la misma.**

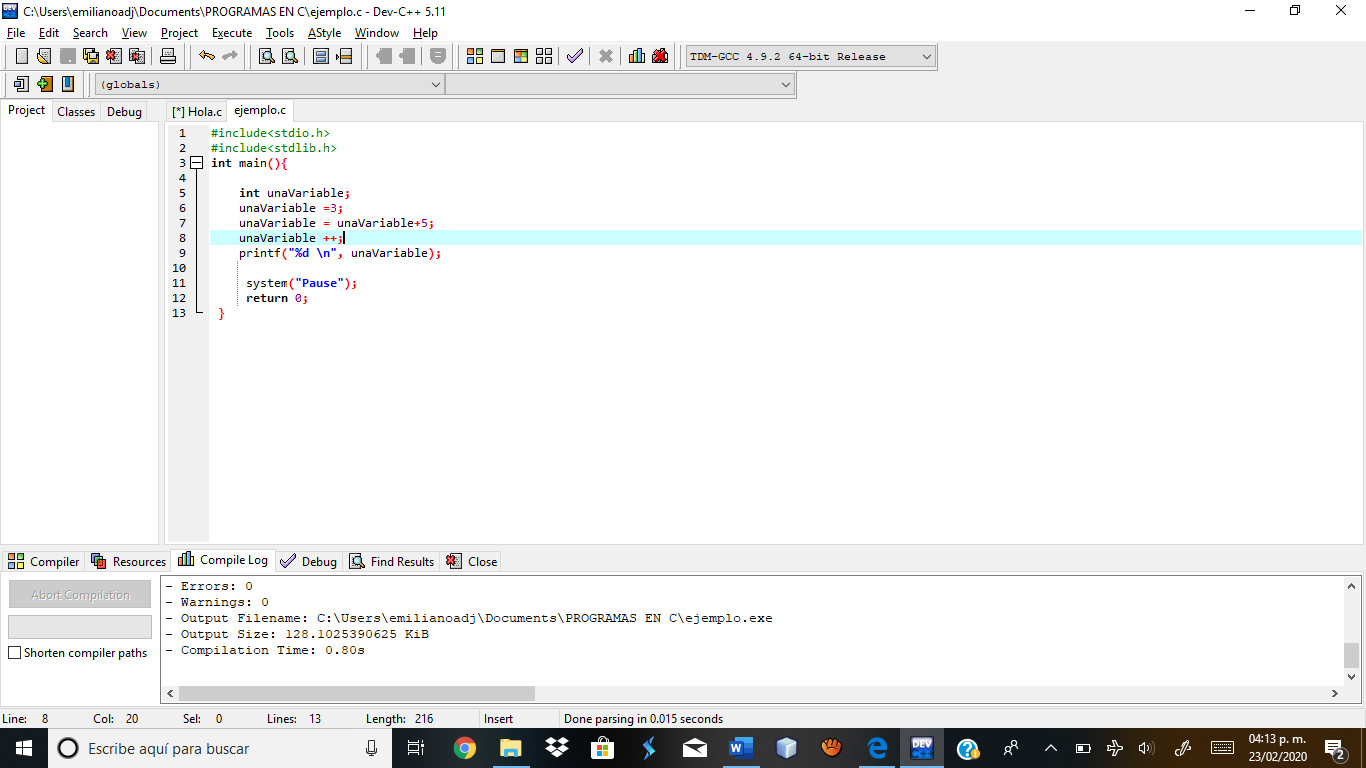
**Utilizamos un especificador de conversion ya que estamos trabajando con numeros enteros y decimales.**

* **El siguiente ejemplo muestra básicamente la suma de dos variables ya declaradas con un tipo de dato entero, por lo cual el especificador de conversión deberá ser decimal con signo (%d).**

****

* **Una variable es una referencia a una sección de memoria.**

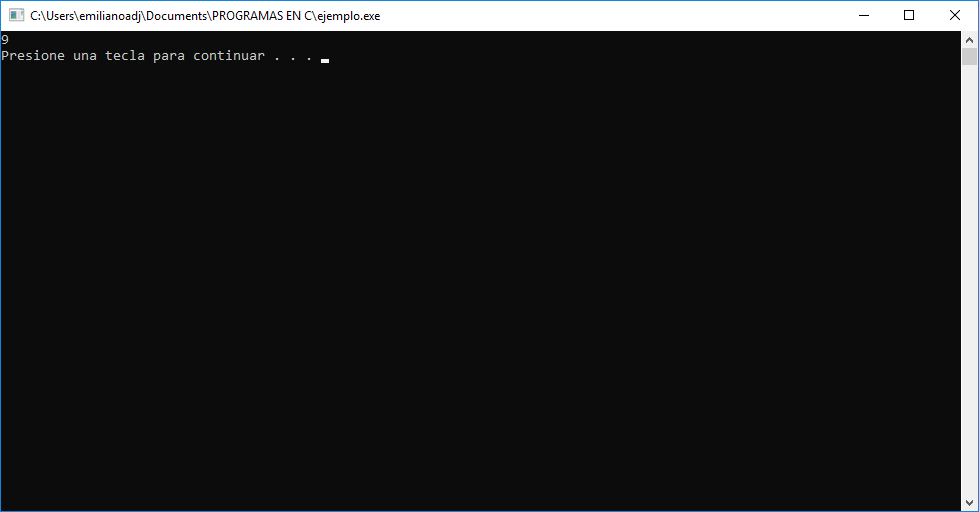
****

****

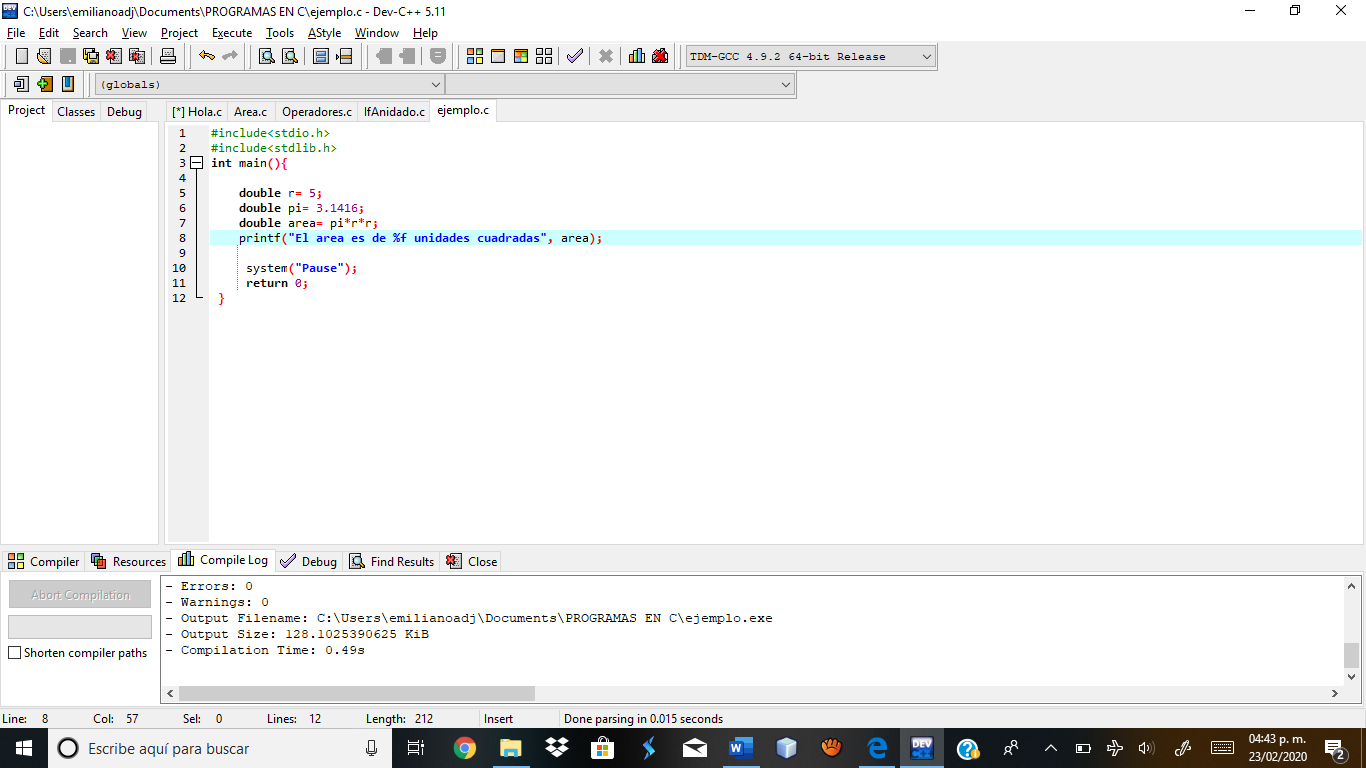
**unaVariable : se ha declarado una variable de tipo entero y se le asigna un valor de 3 (línea 5-6).**

**Posteriormente se realiza una suma que consiste en sumarle 5 a la variable ya declarada la cual es igual a 3 (línea 7). Después se incrementa en uno el nuevo valor de la variable (unaVariable ++; ), (línea 8).**

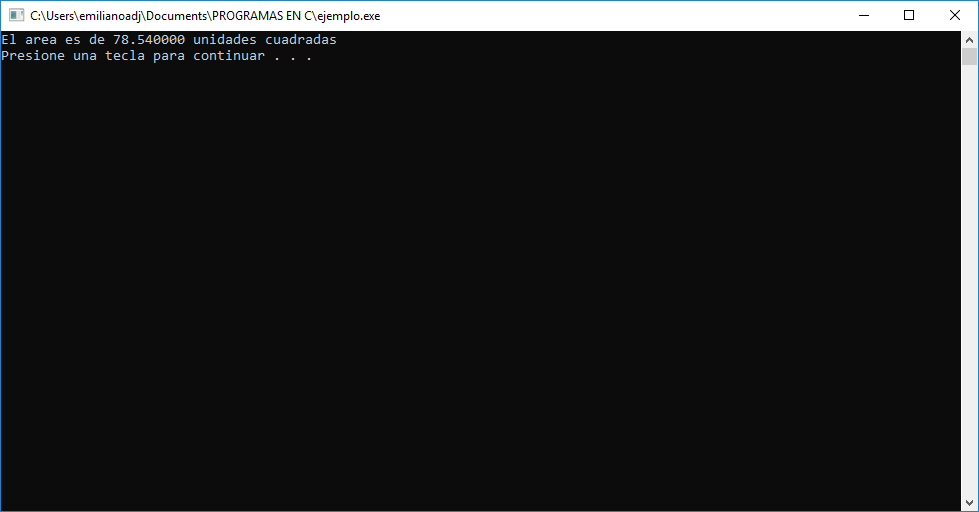
**Finalmente se imprime el resultado de las operaciones anteriores, el resultado debe ser 9.**

****

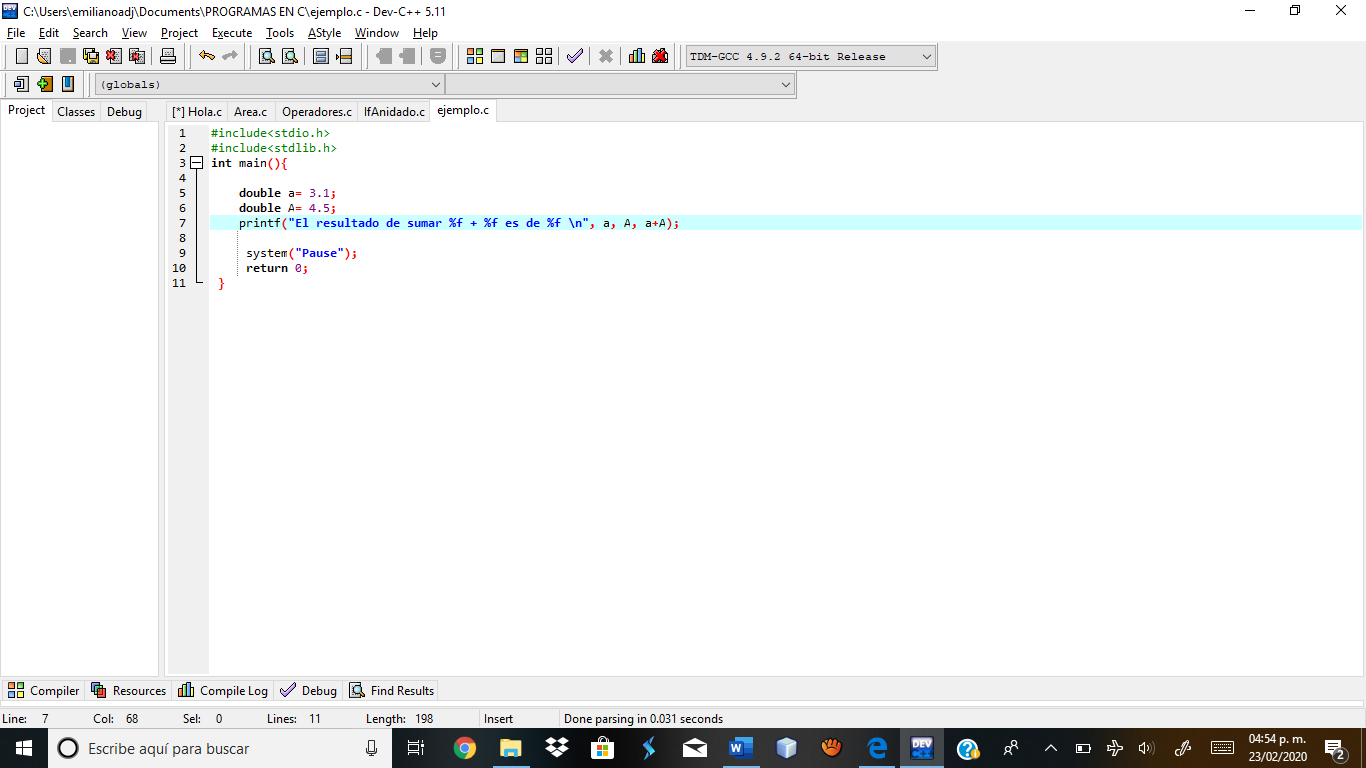
**\*\*\*Las asignaciones de valores a variables SIEMPRE son de derecha a izquierda. \*\*\***

* **El siguiente código muestra una operación la cual tiene el proposito de obtener el area de un circulo, esto se puede deducir visualizando la formula de la linea 7.**

**El resultado que arrojara sera de tipo flotante, ya que estamos trabajando con enteros y decimales. Para ello se utiliza un especificador de conversion.**

****

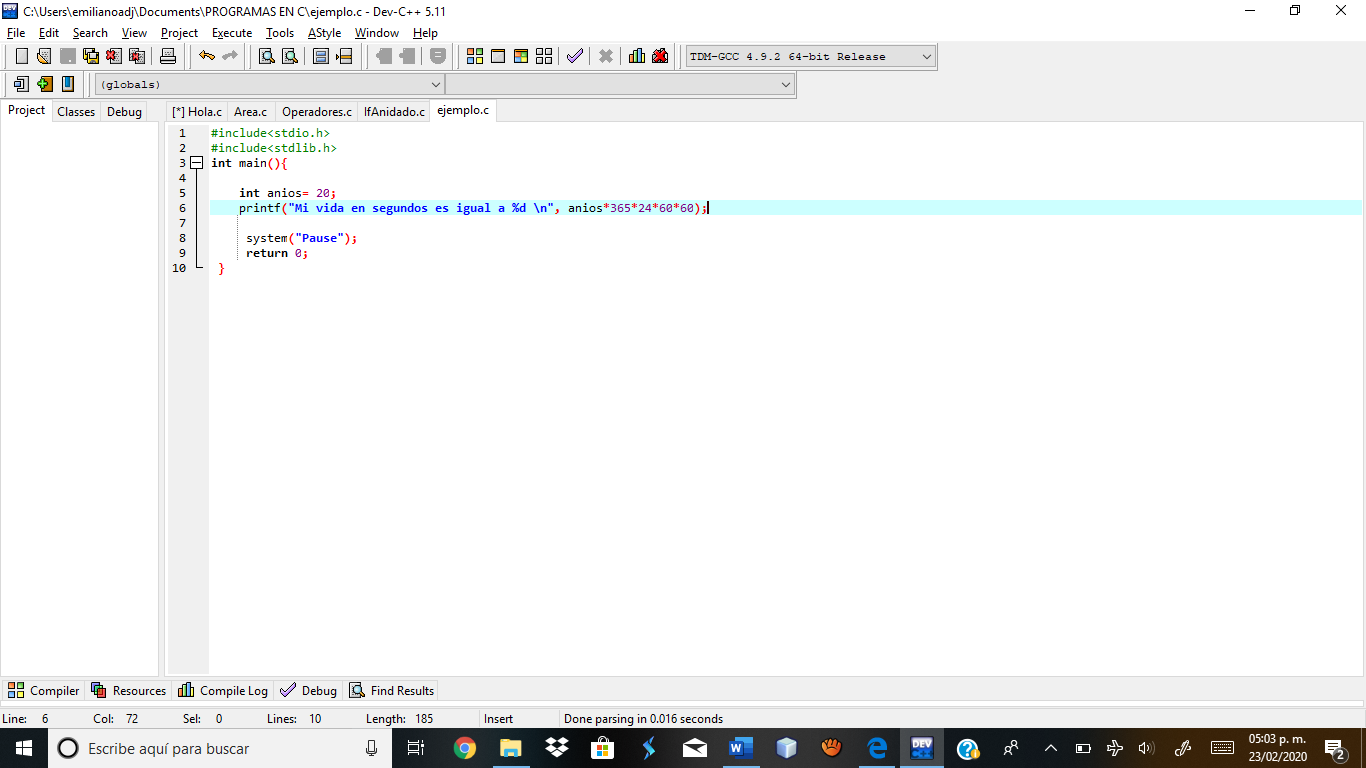
* **A continuación, se muestra un ejemplo de una suma de números con punto decimal, para lo cual utilizaremos dos variables de tipo double y un especificador de conversión %f.**

****

**C distingue entre mayúsculas y minúsculas.**

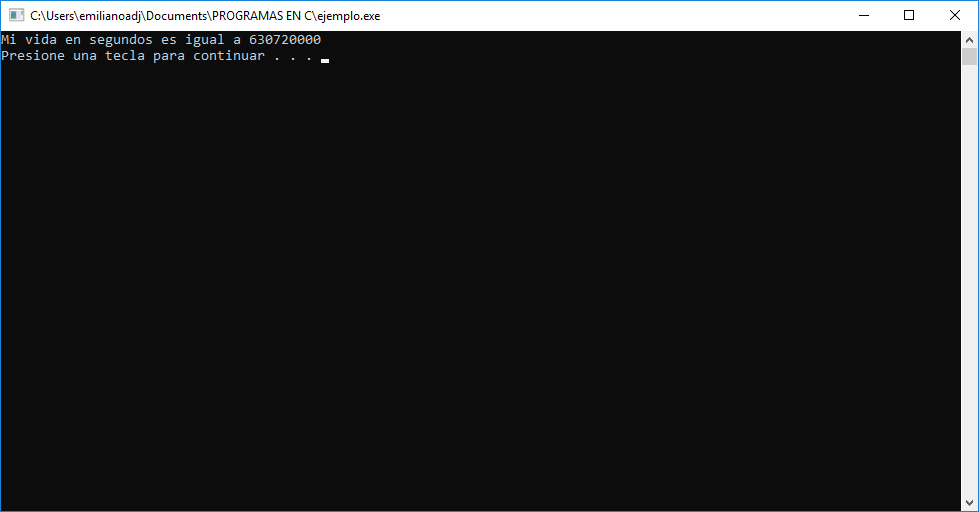
**A no es lo mismo que a.**

**Podemos observar que estamos utilizando tres especificadores de conversión de tipo decimal con punto flotante (%f).**

* **El siguiente programa calcula los segundos de vida que ha vivido una persona haciendo referencia a su edad actual (años).**

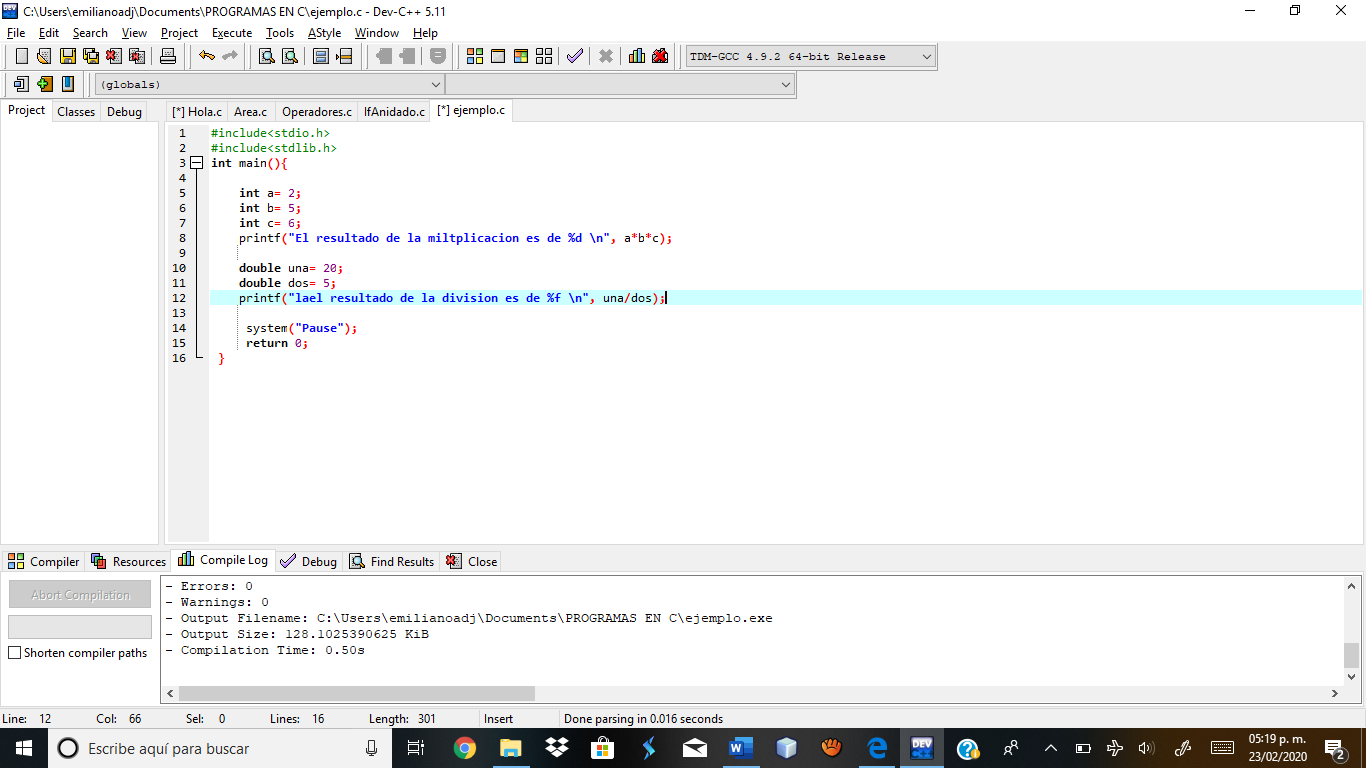
**Solo utilizamos una variable de tipo entero la cual almacenara los años actuales de vida de una persona para después realizar la siguiente operación (línea 6):**

**anios\*365 (los días que tiene un año ordinario)\* 24(las horas que tiene un día)\*60(minutos)\*60(segundos).**

**El resultado final será de tipo decimal.**

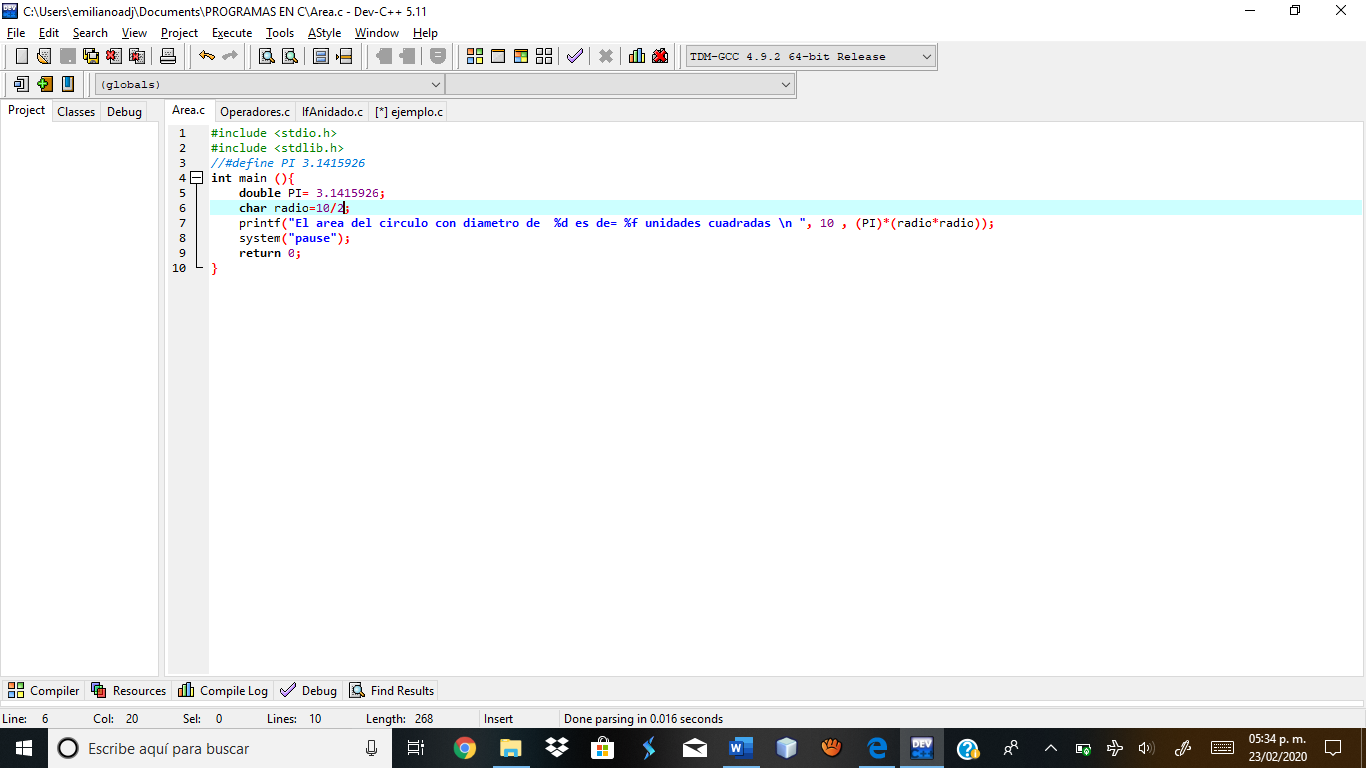
* **Ejercicio:**

1. **Declarar tres variables y asignarles un tipo de dato entero. (mostrar el resultado de la multiplicación de las tres variables).**
2. **Declarar dos variables doubles (reales) y asignarles valores diferentes de 0. (Mostrar el resultado de dividir la primera entre la segunda).**
3. **Declaración de tres variables, posteriormente se realiza la multiplicación de las mismas y se muestra el resultado.**

****

**2. Se declaran dos variables double y se divide la primera entre la segunda mostrando así el resultado.**

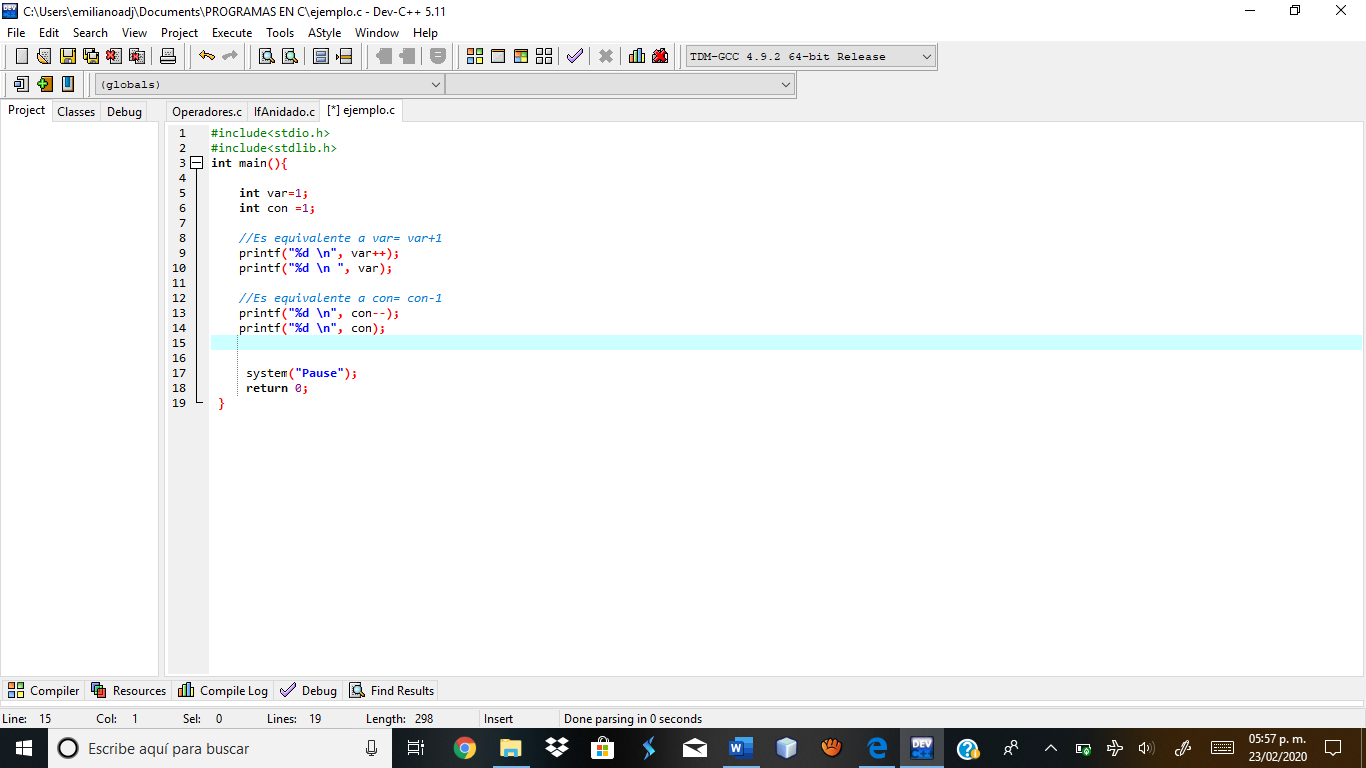
* **Crea un programa que calcule el área de un círculo que tiene 10 metros de diámetro.**

****

**A comparación del programa anterior del área, en este solo tenemos la medida del diámetro, por lo cual debemos obtener el radio dividiendo 10/2 (línea 6), para posteriormente realizar una serie de tres multiplicaciones que consisten en multiplicar PI (declarada como constante o variable) \*radio\*radio.**

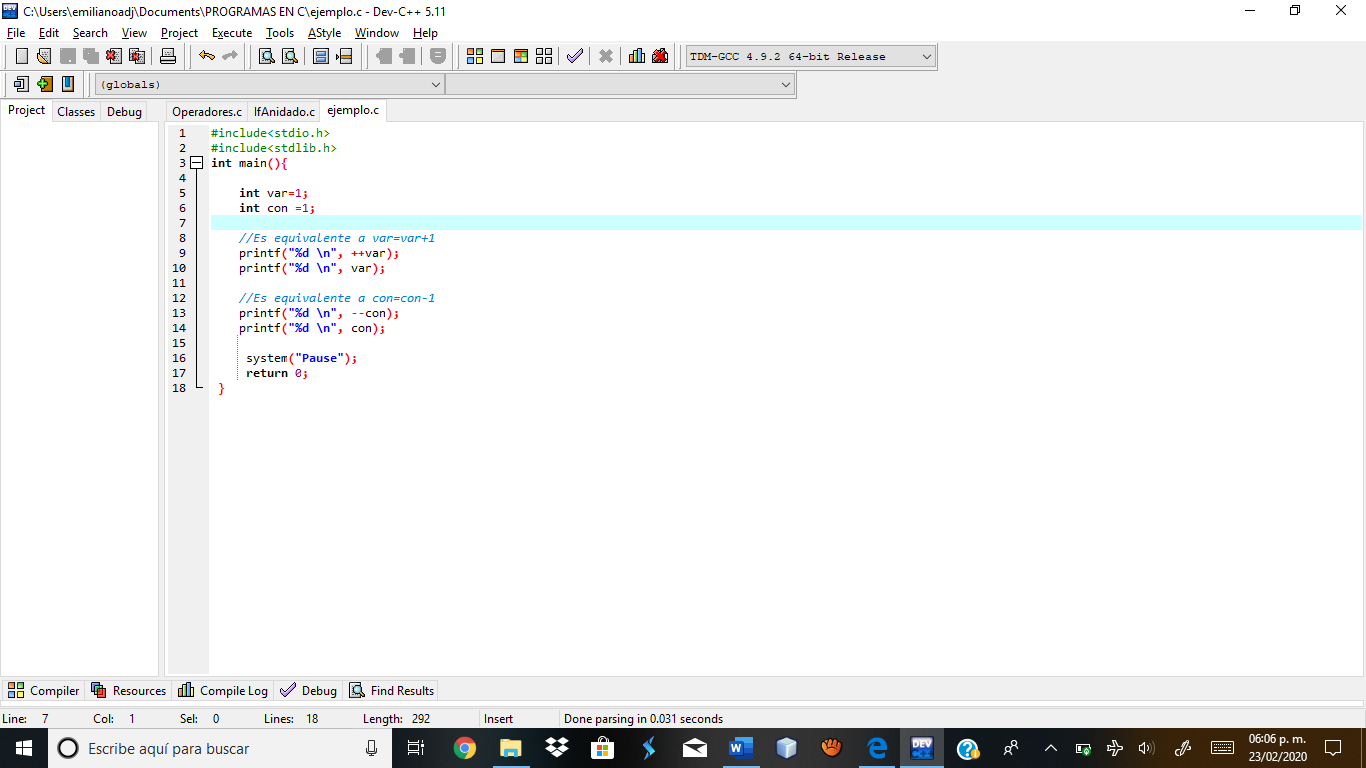
**Recordemos que la fórmula para hallar el área de un círculo es π\*r2 .**

* **Operador POST incremento / decremento.**

**Todos pensaríamos que en la línea 9 imprimiría el numero 2, lo cual es erróneo y en ocasiones confuso.**

**Lo que hace esa línea es imprimir el valor de var (1) y después lo incrementa, pero no lo imprime, es decir, imprimirás 1 y después lo incrementa para imprimirlo en la línea 10.**

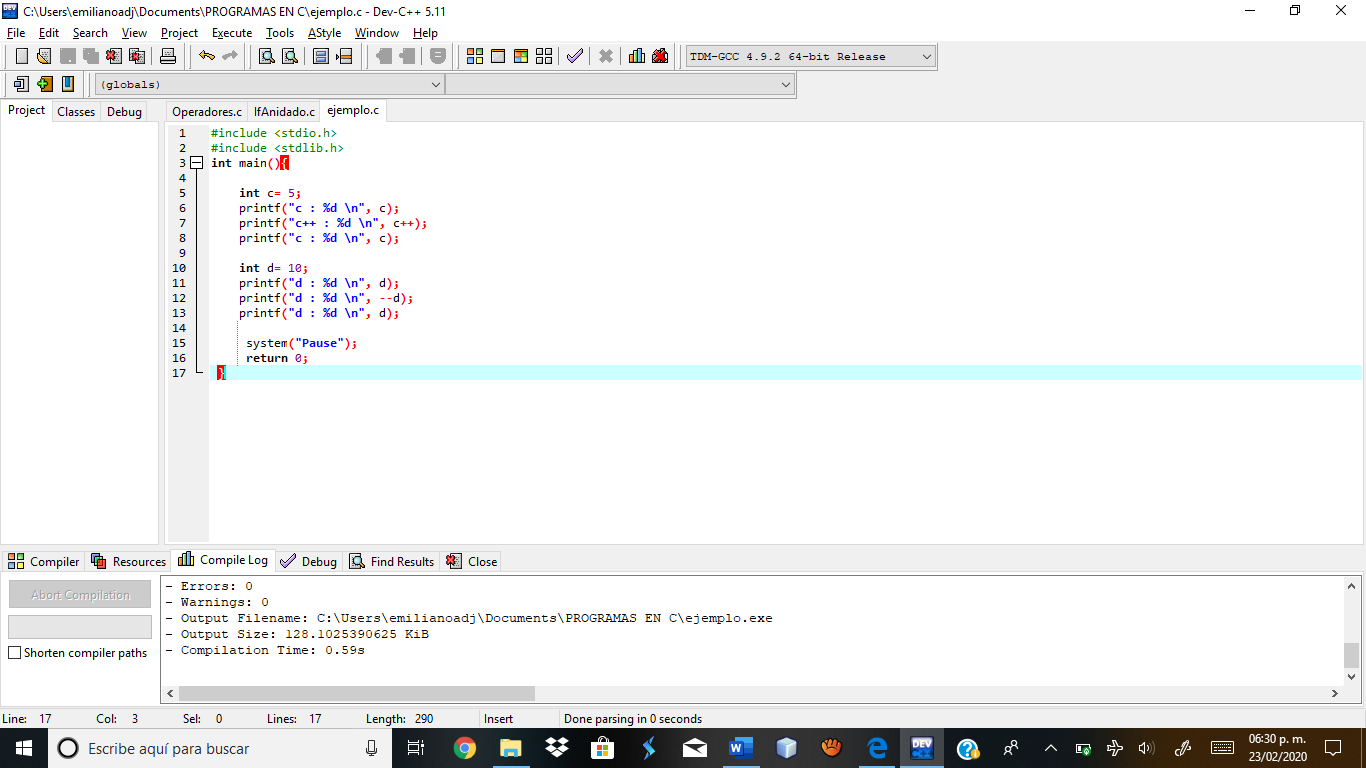
**Lo mismo pasaría con la variable con, pero en este caso esta decrementando su valor original.**

* **Operador PRE incremento / decremento.**

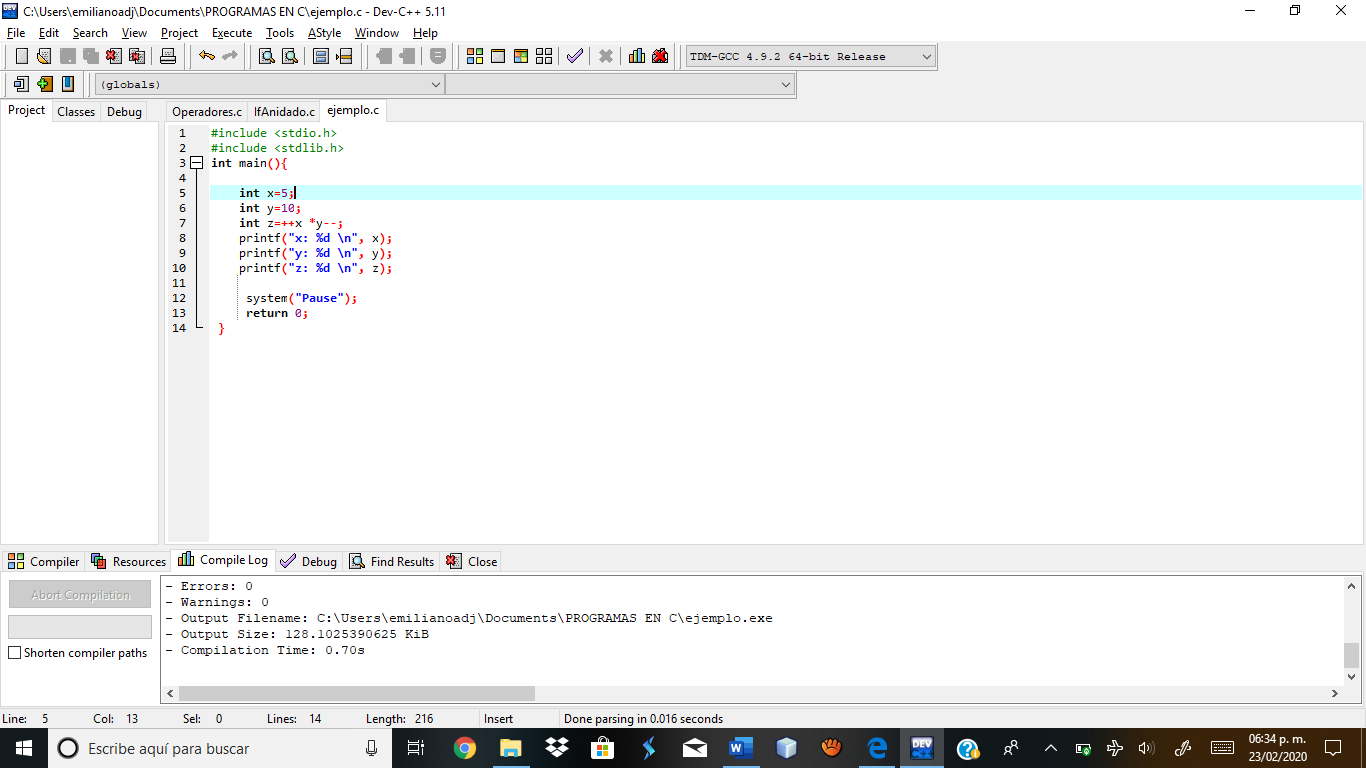
**Similar al programa anterior, con la diferencia que en la línea 9 si esta haciendo primero el incremento, de esta manera imprimirá 2 y en la siguiente línea (10) 2.**

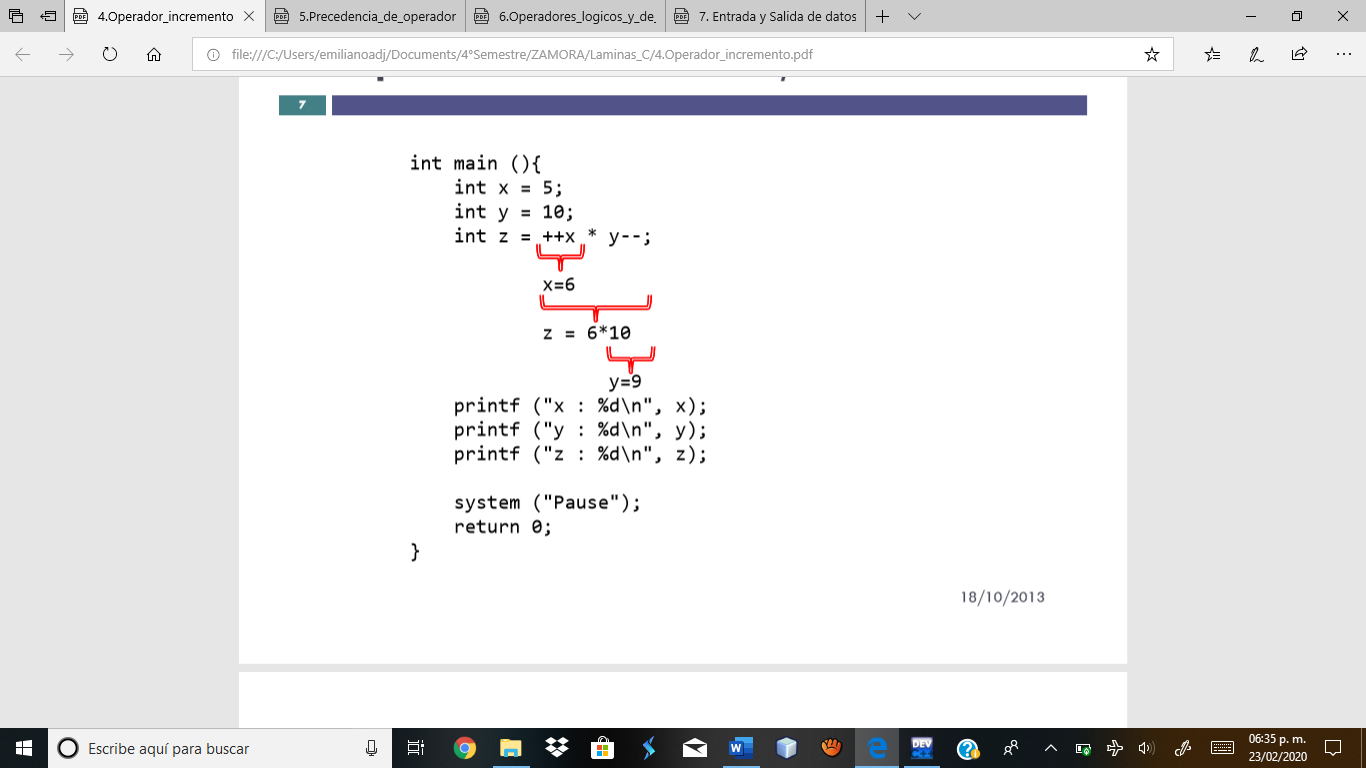
**Algo semejante con la variable con, en la línea 13 imprimirá 0 y en la línea 14 0 también.**

* **Otros ejemplos de POST incremento / decremento PRE incremento / decremento**

****

* **Algunas operaciones.**

****

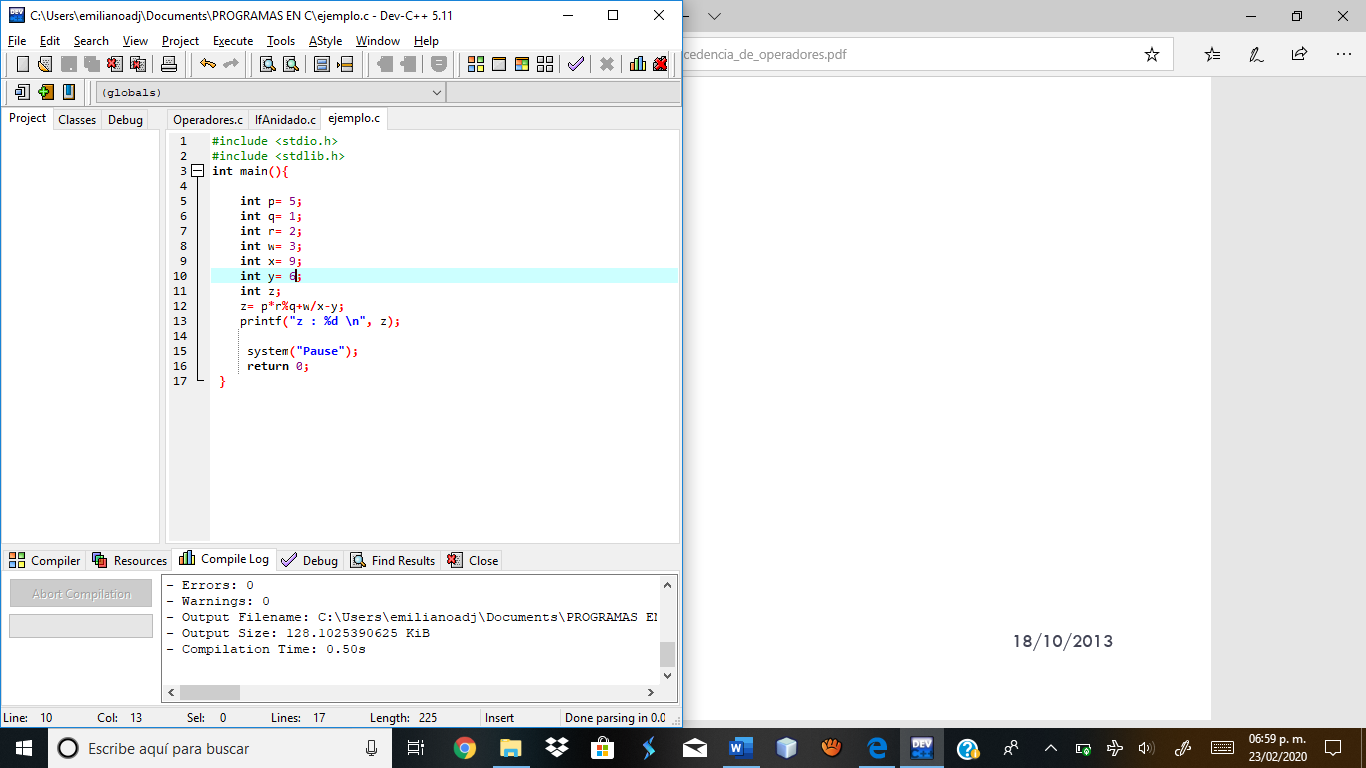
****

**printf("x: %d \n", x); : imprimirá el número 6.**

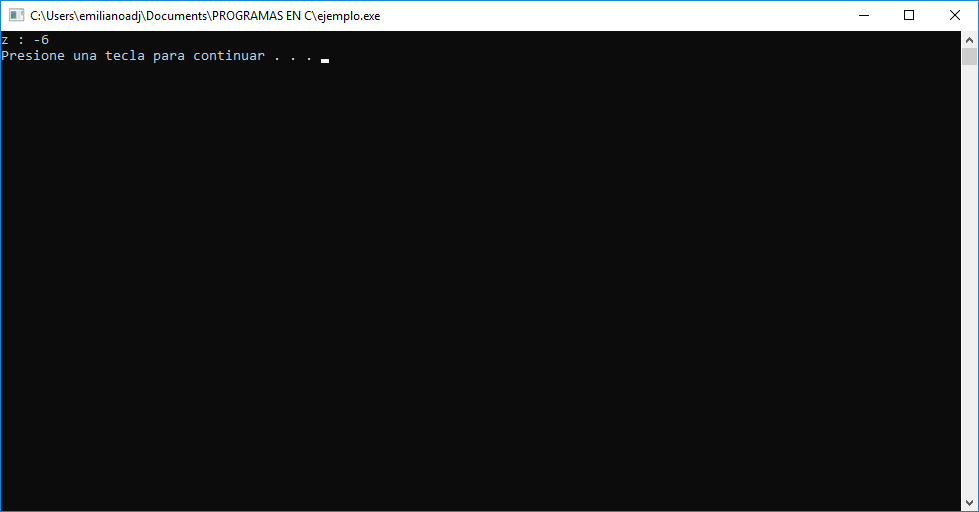
**printf("y: %d \n", y); : imprimirá el número 9.**

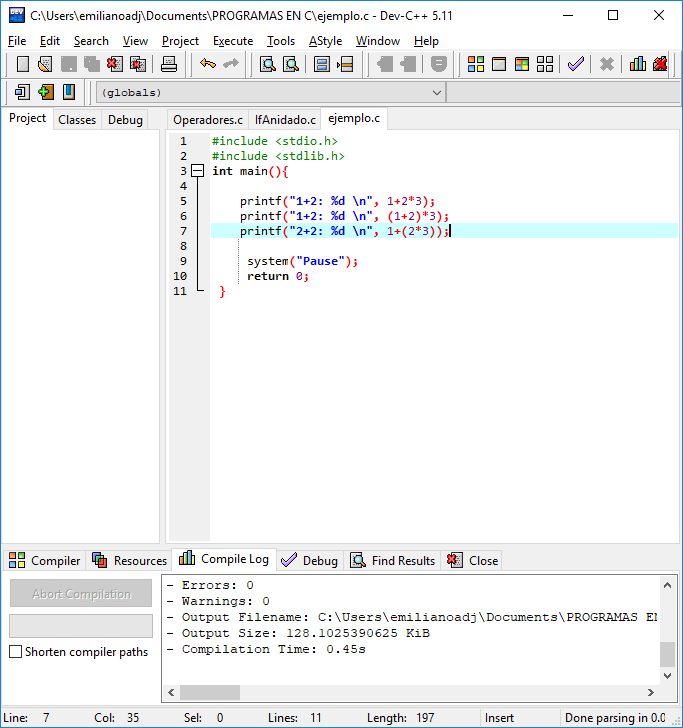
**printf("z: %d \n", z); : imprimirá el número 60.**

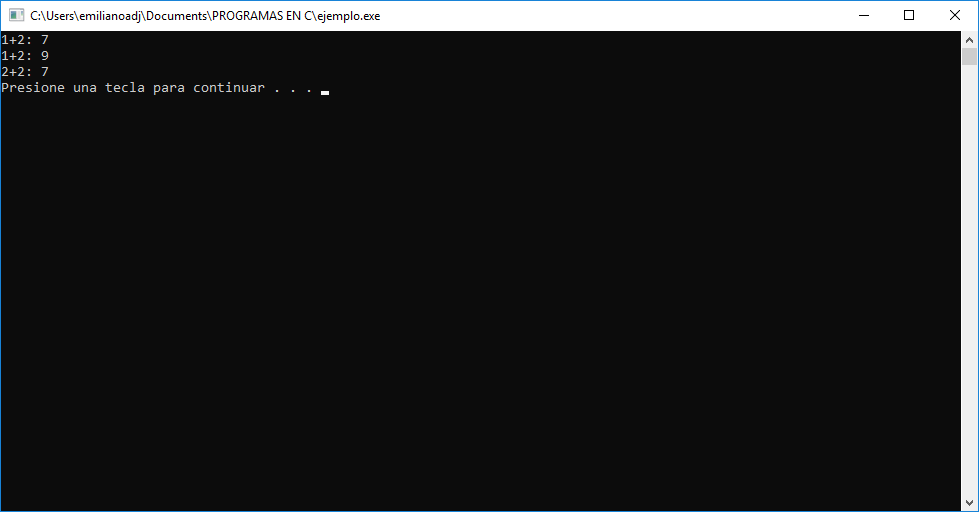
* **Ejemplo de precedencia de operadores.**

**nos encontramos con un ejempo de jerarquia de operaciones, en la linea 12 del codigo, se hace una serie de diferentes operaciones.**

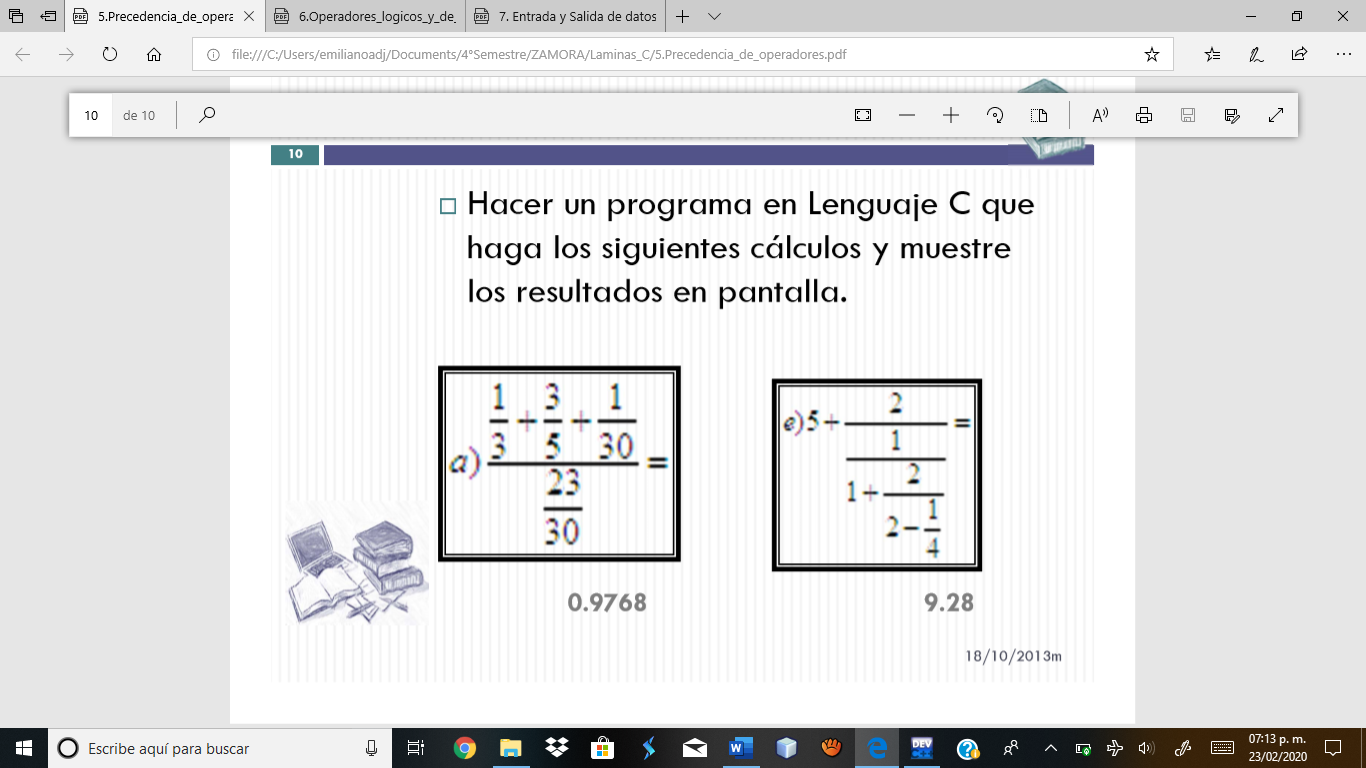
**Se ejecutará primero \*, %, /, +, -, =.**

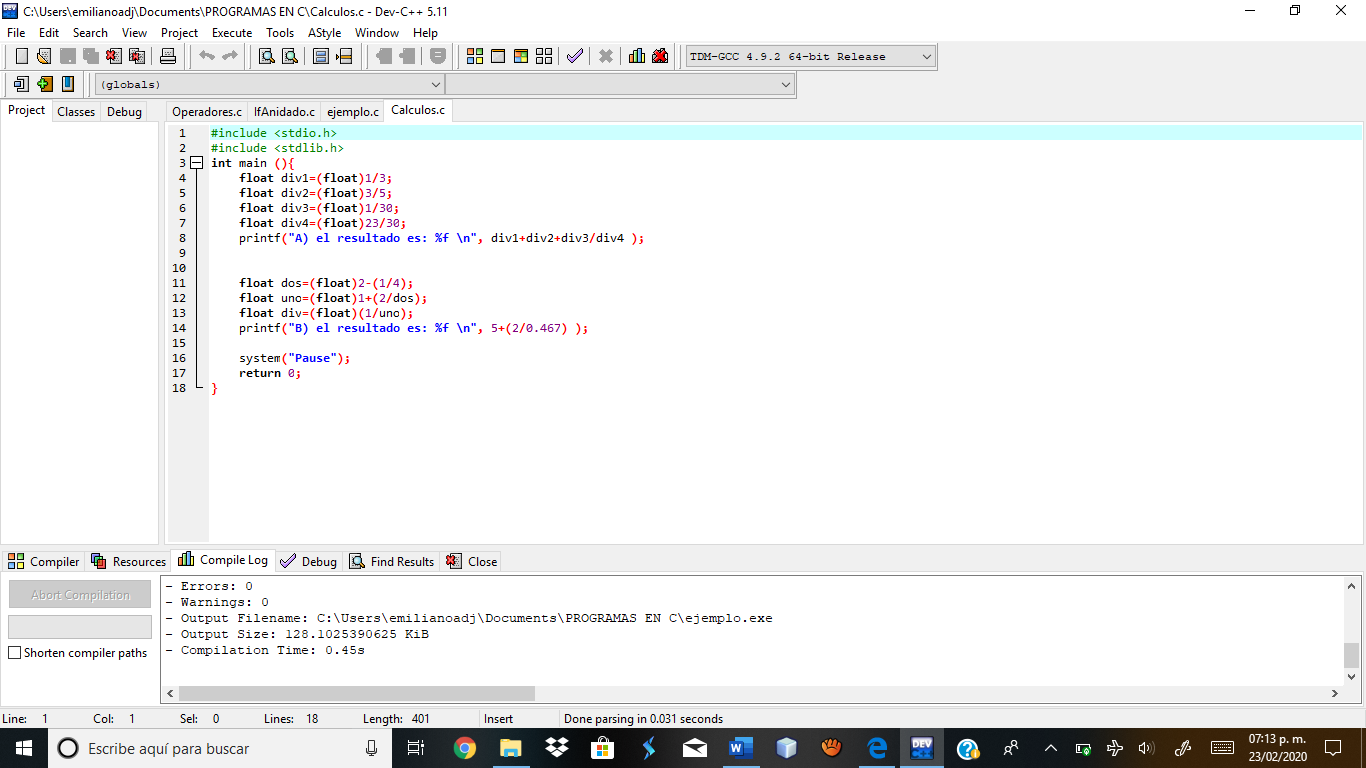
**El resultado final será = -6.**

* **Ejemplo de precedencia de operadores.**

**Los resultados cambian si se manipula la jerarquía, en este caso si se ponen paréntesis agrupando una operación, el resultado será diferente al ordinario (sin paréntesis).**

* **Hacer un programa en C que haga los siguientes cálculos y muestre los resultados en la pantalla:**

****

**Las operaciones anteriores se realizaron de abajo hacia arriba debido a que es una “regla de operaciones” de este tipo.**

**En la primera operación realizamos 3 divisiones que están como numerador de 23/30 para después dividir 23/30 y quedara como denominador. Finalmente una última división ((1/3)+(3/5)+(1/30)) / (23/30).**

**Los resultados totales son con decimales por lo tanto deben tener un especificador de conversión %f.**

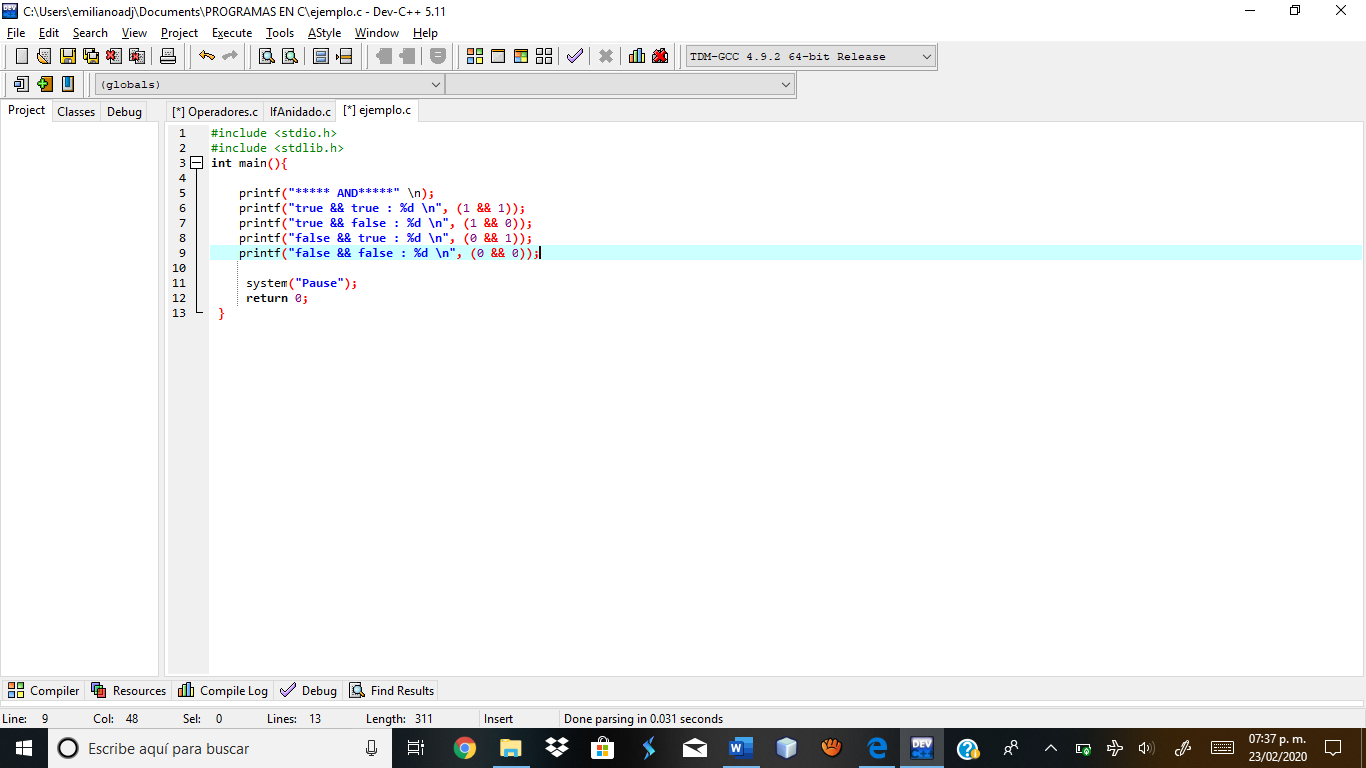
**El proceso fue el mismo para el segundo problema.**

**Para este tipo de ejercicios es necesario tener conocimiento sobre las jerarquías de operaciones y reglas algebraicas.**

* **Operadores lógicos.**

**Llegamos a la parte en donde nos vamos a topar con las tablas de verdad o algo relacionado con estas.**

**Las tablas de verdad se conforman por dos premisas. A continuación, veremos el operador lógico AND (&&).**

**Menciona que obligatoriamente las dos premisas deben ser verdaderas o falsas, es decir, si verdadero y verdadero tiene que ser verdadero o falso y falso tiene que ser falso. No se permite verdadero y falso.**

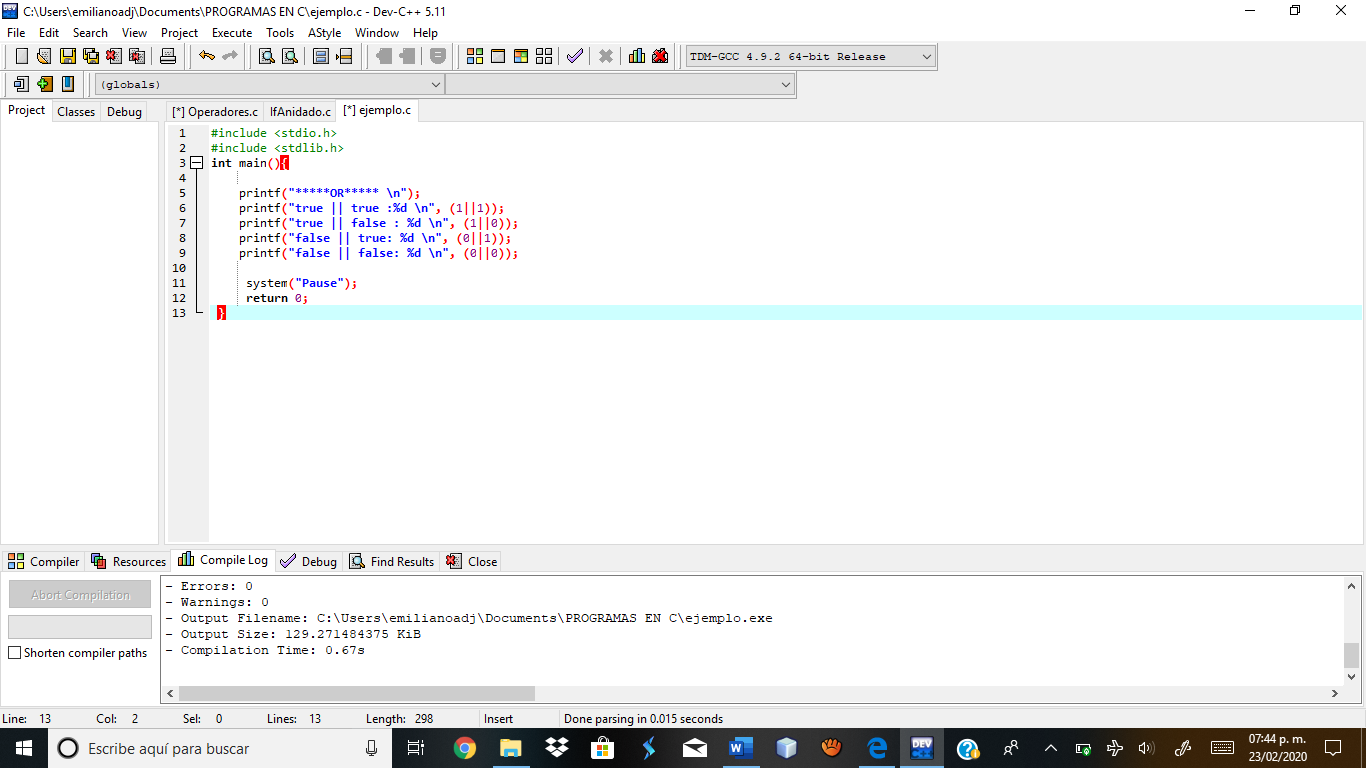
**printf("true && true : %d \n", (1 && 1)); : 1**

**printf("true && false : %d \n", (1 && 0)); : 0**

**printf("false && true : %d \n", (0 && 1)); : 0**

**printf("false && false : %d \n", (0 && 0)); :0**

* **Operador lógico OR (||).**

**Dice que una de las dos premisas debe ser verdadera, aunque una de estas dos sea falsa el resultado debe ser verdadero.**

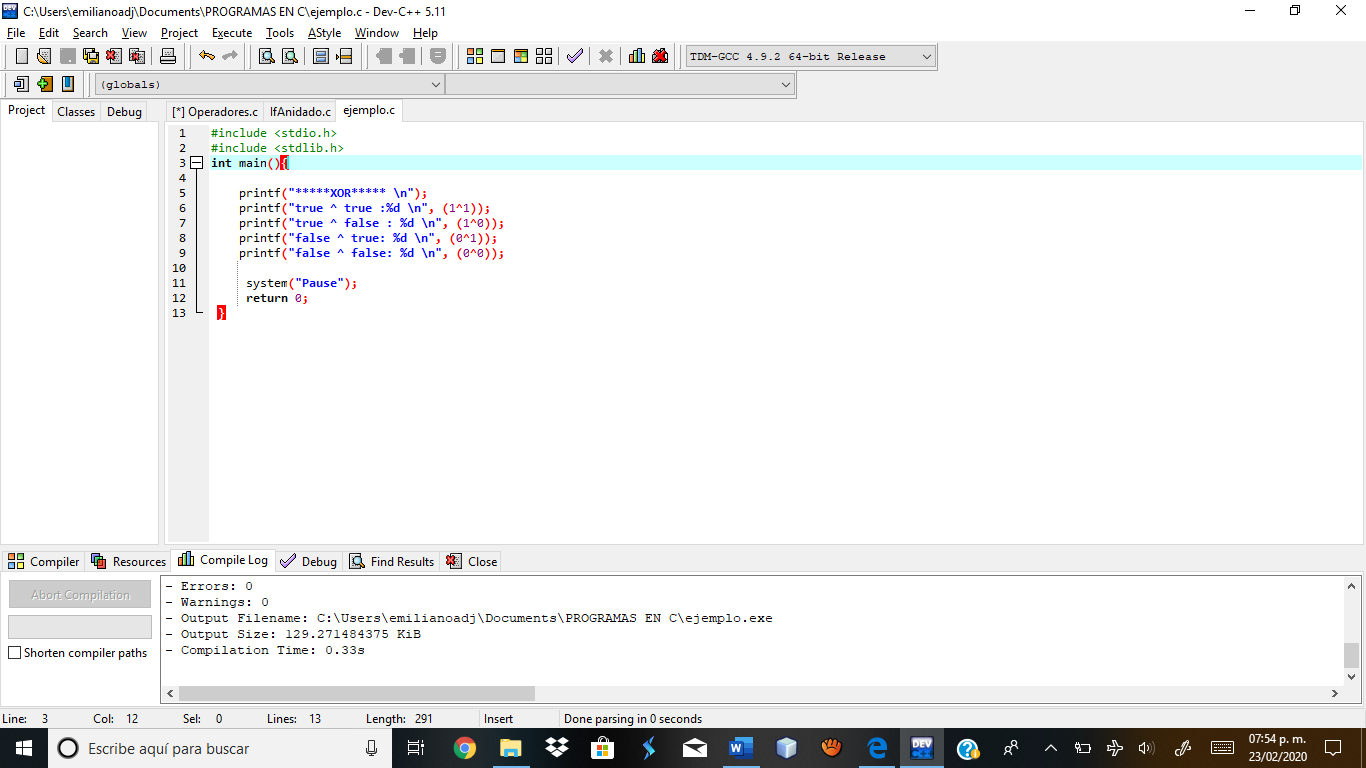
**printf("true || true :%d \n", (1||1)); :1**

**printf("true || false :%d \n", (1||0)); 1**

**printf("false || true :%d \n", (0||1)); :1**

**printf("false || false :%d \n", (0||0)); :0**

* **Operador exclusivo XOR (ˆ).**

**Este operador es exclusivo, ya que solo puede ser obligatoriamente verdadera una premisa o la otra, aunque las dos fuesen verdaderas, establece que es erroneo este caso.**

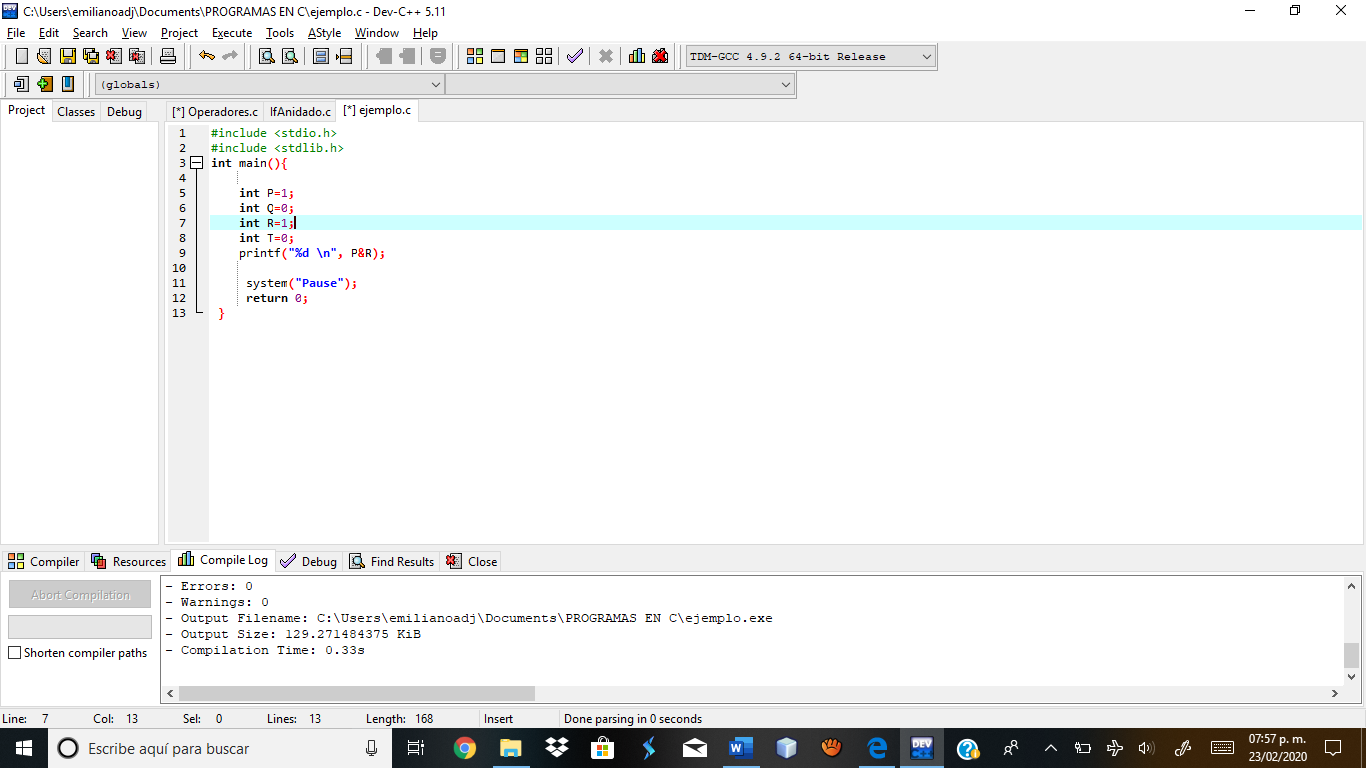
**printf("true ^ true :%d \n", (1^1)); :0**

**printf("true ^ false : %d \n", (1^0)); :1**

**printf("false ^ true: %d \n", (0^1)); :1**

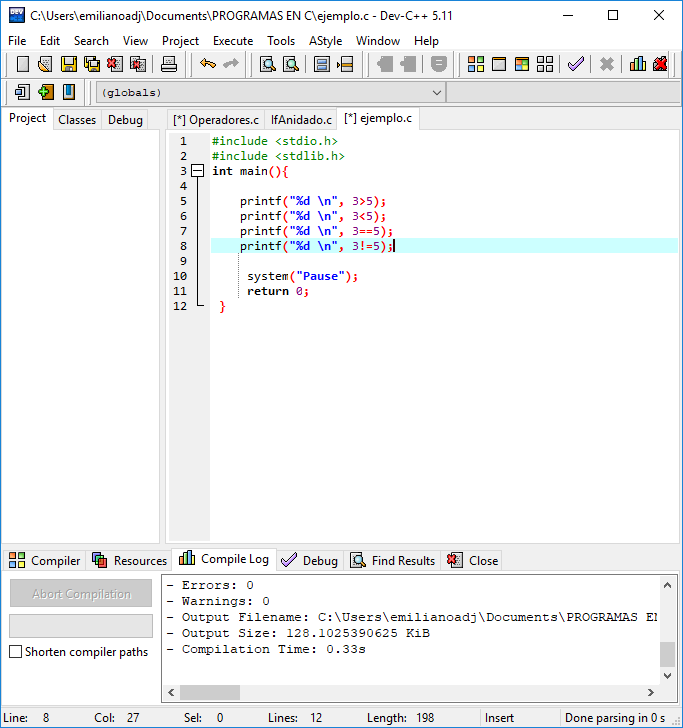
**printf("false ^ false: %d \n", (0^0)); :0**

* **Se muestra un programa donde hace uso de los operadores logicos.**

****

**Se definen 4 variables de tipo entero y en la impresión del resultado (línea 9) evalúa a P=1 && R=1.**

**Lógicamente el resultado debe ser verdadero (1).**

* **Operadores relacionales.**

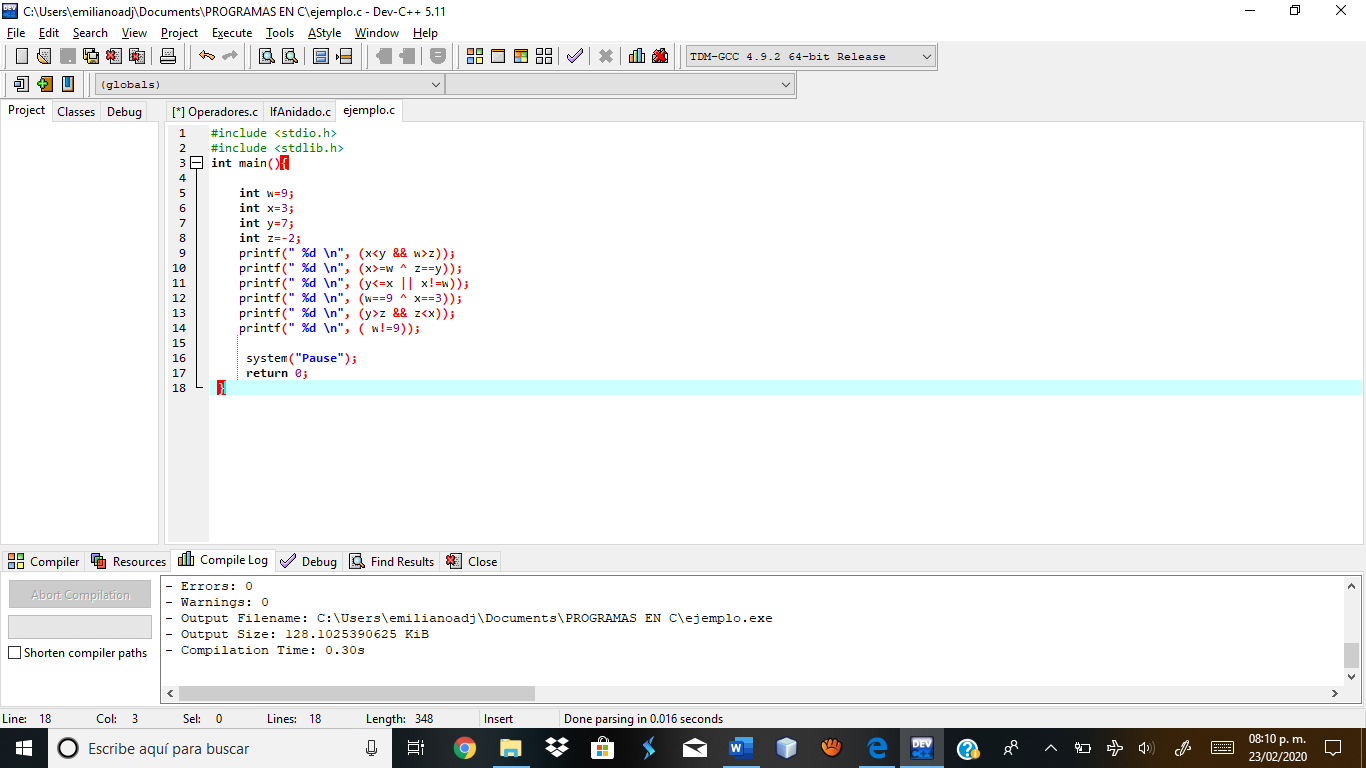
**Los resultados seran los siguientes:**

**¿Tres es mayor a 5? 0**

**¿tres es menor a 5? 1**

**¿tres es igual a 5? 0**

**¿tres no es igual a 5? 1**

* **Ejemplo de implementación operadores lógicos y relacionales.**

**Se declararon 4 variables w=9, x= 3, y=7, z=-2.**

**printf(" %d \n", (x<y && w>z)); : (3<7) && (9>-2)= 1**

**printf(" %d \n", (x>=w ^ z==y)); : (3>=9)ˆ(-2==7)= 0**

**printf(" %d \n", (y<=x || x!=w)); : (7<=3) || (3 no es = 9)= 1**

**printf(" %d \n", (w==9 ^ x==3)); : (9==9)ˆ(3==3)= 0**

**printf(" %d \n", (y>z && z<x)); : (7>-2) && (-2<3)= 1**

**printf(" %d \n", ( w!=9)); : (9 no es = 9)= 0**