|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **NOMBRE DE LA PRÁCTICA** | **REPORTES 12 Y 13** | | | **No.** |  |
| **ASIGNATURA:** | **MÉTODOS NÚMERICOS** | **CARRERA:** | **ISIC 3402** | **PLAN:** |  |

**ERIKA YAZMIN DIONISIO VELASCO 3402 ISIC**

**I. COMPETENCIA(S) ESPECÍFICA(S):**

**II. MATERIAL EMPLEADO:**

* **Laptop**
* **Aplicación C++**
* **Laminas 12 y 13**

**III. DESARROLLO DE LA PRÁCTICA:**

Para comenzar nuestra práctica tendremos que entender antes que nada cada uno de los conceptos que estaremos abordando durante la elaboración de nuestro manual.

Dividiremos a nuestro manual en dos partes primero abordaremos **cadenas** y por ultimo **Funciones.**

Así que contemplando los temas, comenzamos:

**LAMINA 12. CADENAS**

**¿Qué es una cadena?**

Una cadena es un arreglo de caracteres. En donde, por lo general el último elemento deberá ser el carácter ‘\0’.

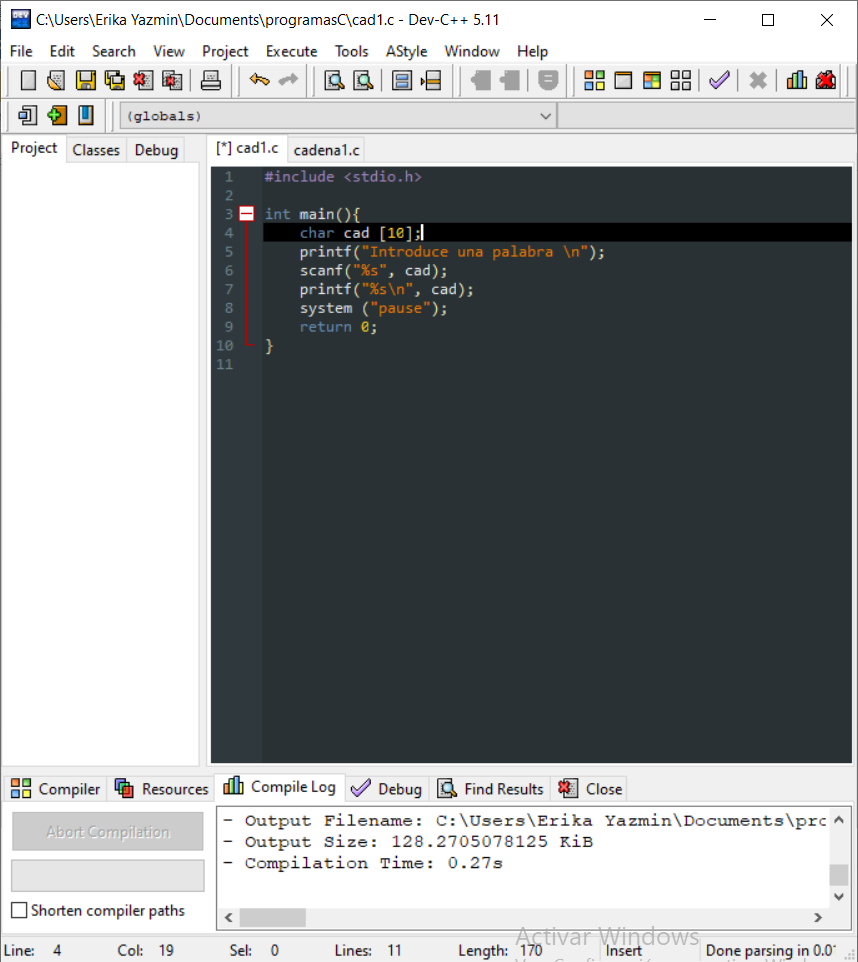
Debemos recordar que un carácter siempre tiene que estar declarado por un arreglo para que este pueda guardar el tamaño de nuestra cadena.

PROGRAMA 1. **¿Cómo declarar una cadena?**

Declaramos un arreglo de tipo **char** con 10 posiciones.

Imprime un mensaje en pantalla

Método principal

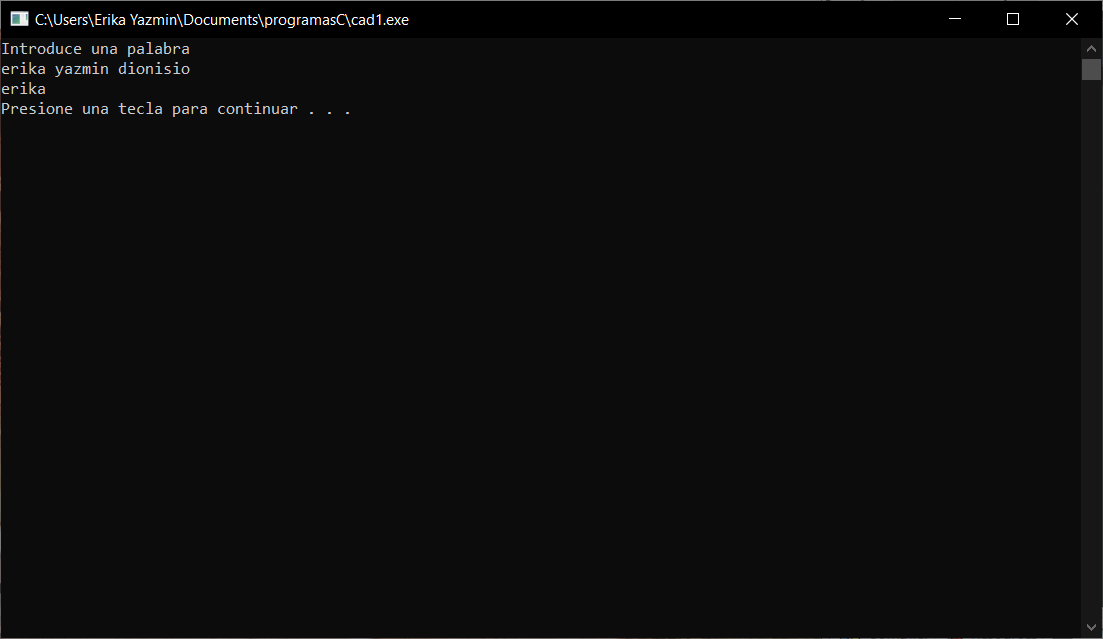


Imprime lo que leyó anteriormente

Lee datos del teclado de acuerdo a un formato especificado, y asigna los datos de entrada a una o más variables del programa.

Imprime el término del programa y retorna un valor 0 o null.

Podemos observar que la cadena que declaramos es de 10 posiciones por lo que al momento de imprimir la palabra que hayamos colocado solo abarcara las 10 posiciones asignadas, así sea que la palabra salga cortada o no.



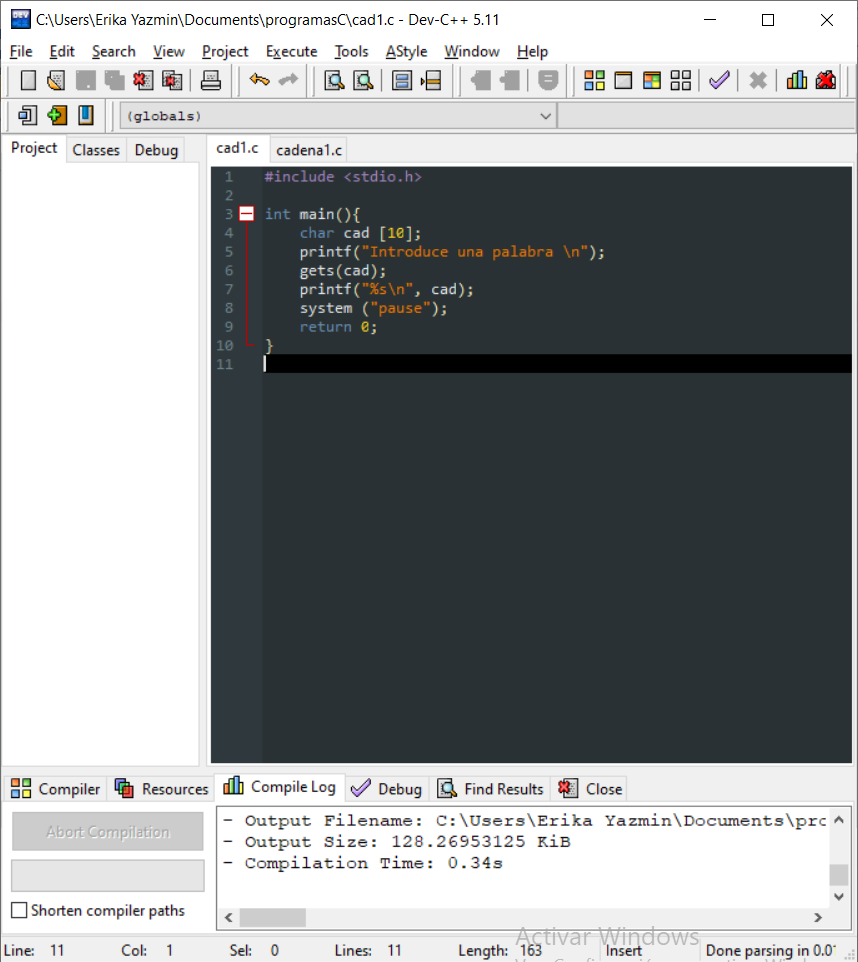
La palabra que asignamos abarca más de 10 posiciones por lo que solo tomara en cuenta la primera.

Ahora, si hacemos este mismo programa pero con una forma distinta de leer la cadena escrita desde teclado utilizando la sentencia de  **gets**:

Declaramos un arreglo de tipo **char** con 10 posiciones.

Método principal

Imprime un mensaje en pantalla



La función **gets** retorna **cadena.**

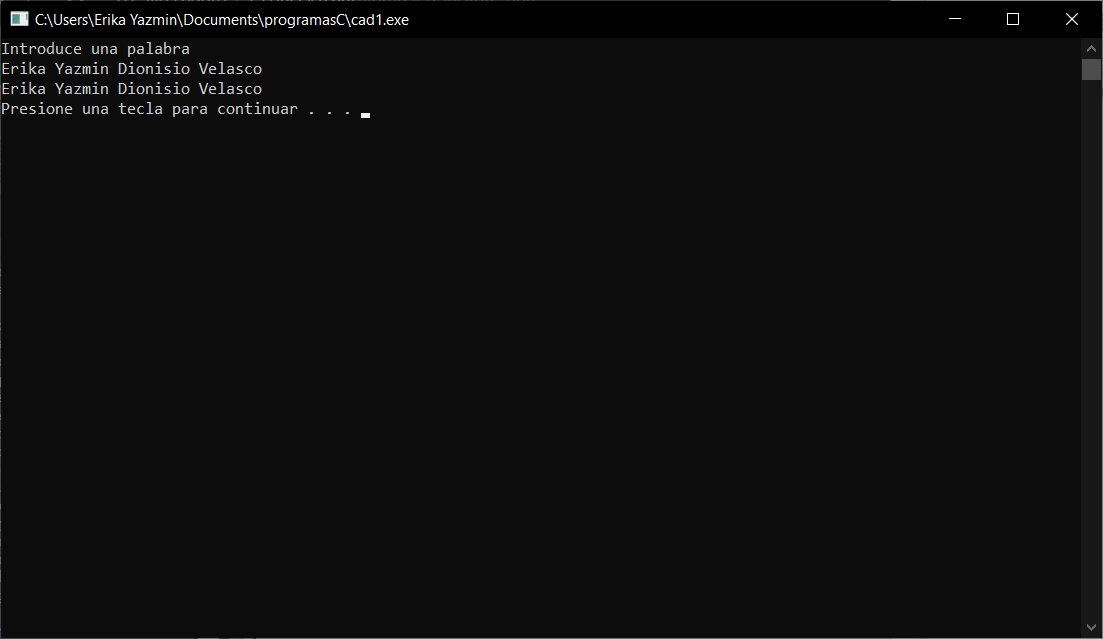
El contenido del array o arreglo permanece invariable y un puntero nulo es retornado, esto quiere decir que lo escrito en pantalla en automático devolverá la cadena asignada.

Si ocurre un error de lectura durante el proceso, el contenido del array es indeterminado y un puntero nulo es retornado.

Imprime el término del programa y retorna un valor 0 o null.

Imprime lo que leyó anteriormente

Ejecución de nuestro programa:



**¿Cómo declarar una cadena?**

Crearla como arreglo sin tamaño

char cad [] = “Es una cadena”;

**¿De cuántos elementos consta el arreglo cad?**

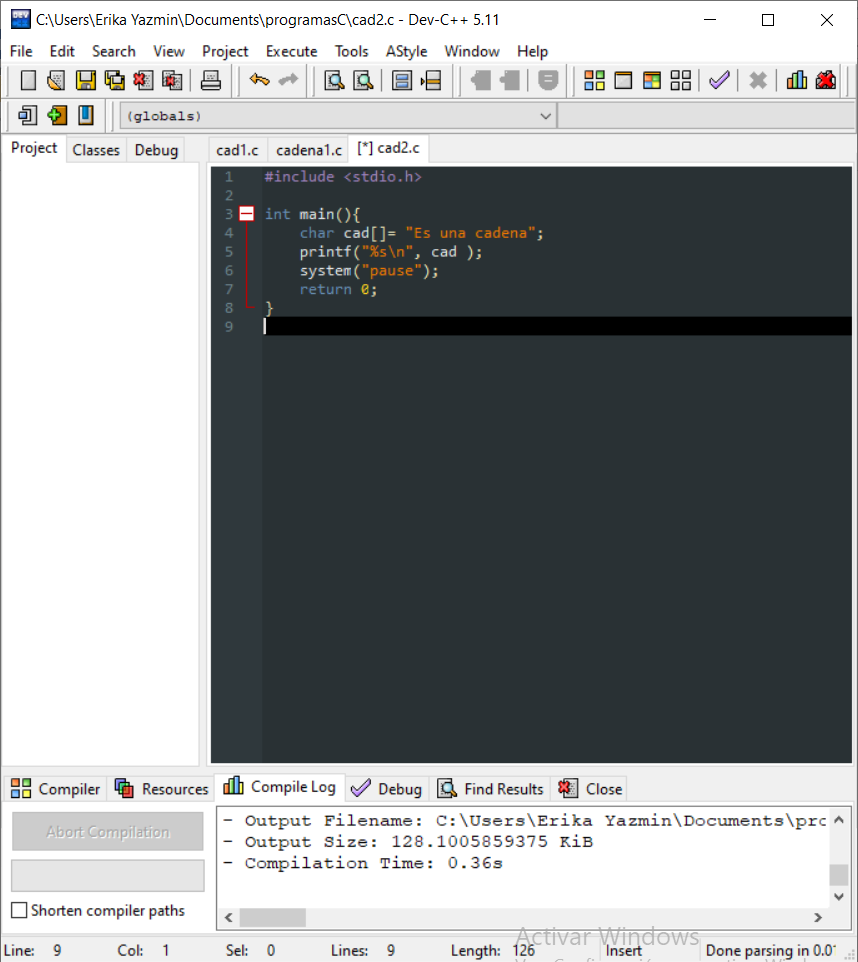
Este arreglo consta de 13 elementos contando espacios

**PROGRAMA 2 Cadena**

Asignación de un arreglo llamado **cad** en donde le asignamos una cadena de 13 posiciones

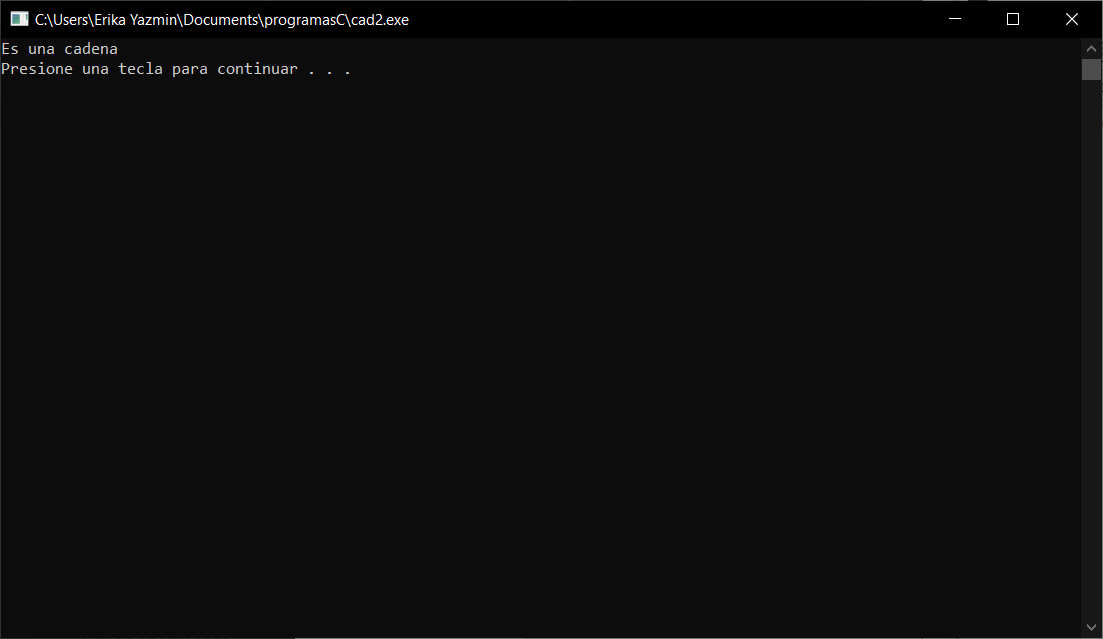
Método main

Imprime lo que está dentro del arreglo, con el especificador de conversión **%s (cadena de carácter)**



Termina el programa

EJECUCIÓN DEL PROGRAMA



**FUNCIONES DE CADENA: STRLEN**

Devuelve la longitud de la cadena sin tomar en cuenta el carácter de final de cadena.

**strlen (<cadena>)**

**PROGRAMA 3 “strlen”**

Declarar un arreglo de tipo **char,** llamado **cad** con una cadena de carácter.

Método main

Declarar una variable de tipo entero **len.**

En la variable **len**, asignar el tipo de cadena **strlen** en la cual medirá la longitud de la cadena, seguido del arreglo declarado.



Imprimir el arreglo con la cadena de carácter, seguido de nuestra variable en donde guarda la longitud de la cadena, asignando su especificador de conversión de cada uno.

Fin del programa

EJECUCIÓN DEL PROGRAMA



**Programa 4**

**Escribe un programa que reciba una palabra por teclado.**

**De acuerdo a la longitud de la palabra (N) que se ingresó por teclado imprime un cuadrado de asteriscos de (N x N).**

Asignación de 4 variables **len,** para la longitud de cadena

**J e i**, para los for anidados de posición y asignación de los asteriscos

Método main

**v**



Asignamos a la variable **len** el tipo de cadena **strlen** la cual cuenta la longitud de la palabra la cual esta guardada en nuestro arreglo.

**Puts** , manda el mensaje en pantalla y **Scanf** lee lo escrito y lo recopila, asignándoles el especificador de conversión seguido del arreglo.

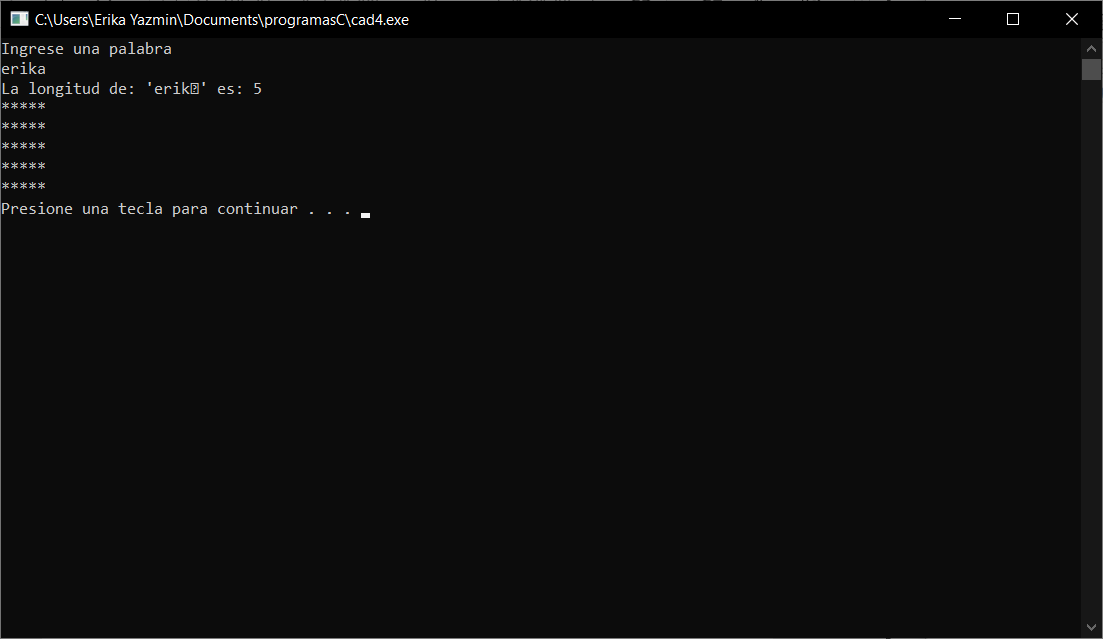
Declaración de nuestro arreglo sin posiciones, solo el espacio libre entre comillas para recibir una cadena de carácter.

Fin del programa

Afuera del primer ciclo colocaremos el salto de línea para poder armar el cuadro de nxn

Colocamos los ciclos for anidados para saber la posición y las veces en las que se repetirá el asterisco de acuerdo a la palabra escrita, todo esto tendrá que ser menor a **len** la variable que mide la longitud de la palabra

EJECUCIÓN DEL PROGRAMA



**FUNCIONES DE CADENA: STRCPY PROGRAMA 5**

Copia el contenido de:

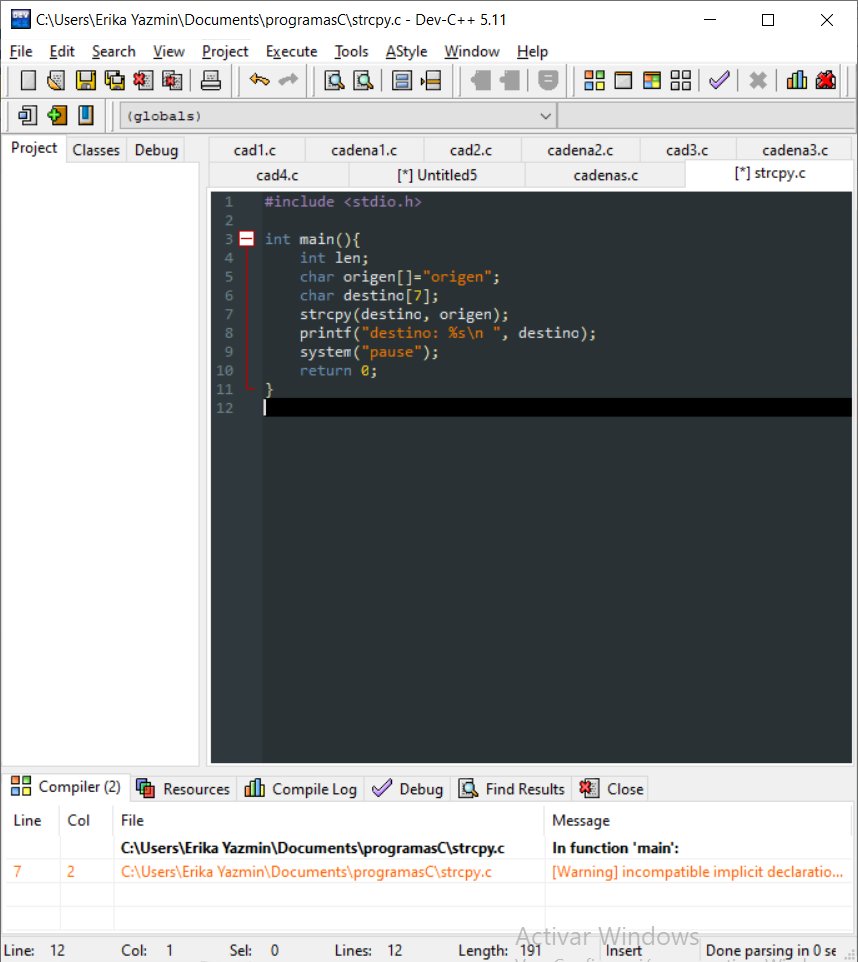
<cadena\_origen> en <cadena\_destino>.

Asignar dos arreglos de tipo **char** origen y destino, en origen le asignamos una cadena de carácter y en **destino** solo colocamos el tamaño de nuestro arreglo.

**strcpy** (<cadena\_destino>, <cadena\_origen>)

Asignar una variable de tipo entero **len**

Método main

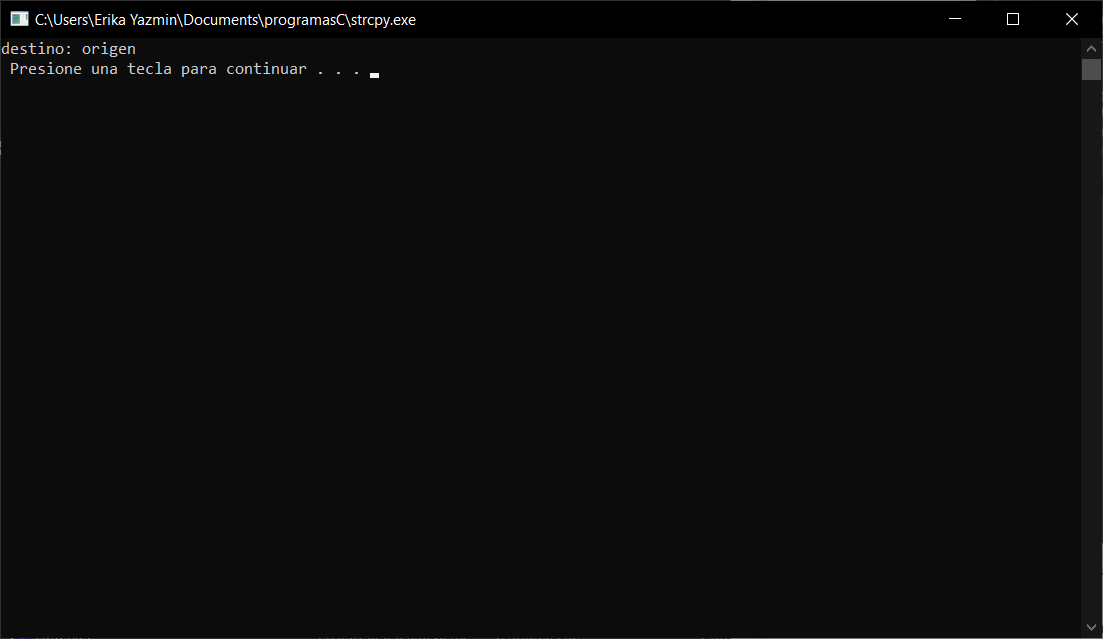


**strcpy,** solo copiara el contenido del arreglo origen y lo posicionara en destino por lo que mandara a llamar primero al arreglo destino y siguiente de este a rigen

Imprimirá al arreglo destino y por ende este tendrá dentro a la palabra de origen.

Fin del programa

EJECUCIÓN DEL PROGRAMA



**Programa 6**

Escribe un programa que reciba por teclado dos palabras y cada una de ellas las almacene en un arreglo.

Después intercambia sus contenidos. Imprime el antes y el después

 Ej. **Antes después**

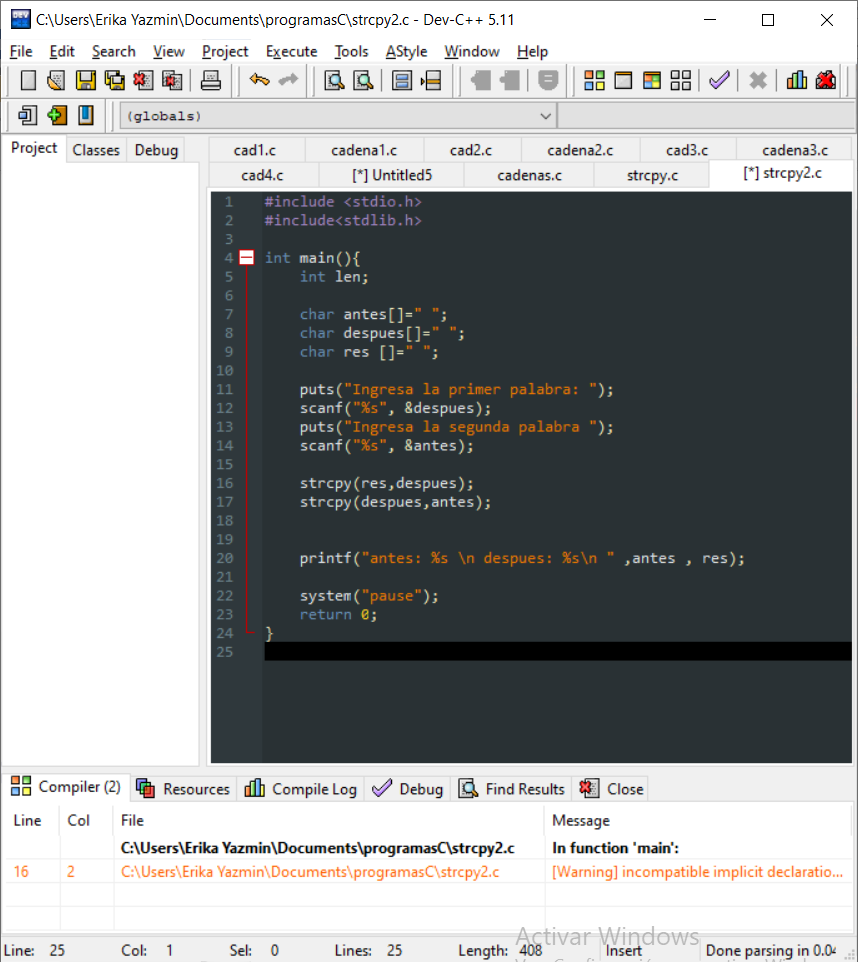
Palabra\_1 = ' Programación' Palabra\_1 = 'Computadora'

Palabra \_2= 'Computadora' Palabra \_2= 'Programación'

Asignar tres arreglos de tipo **char** sin asignarles algún número de posiciones, ya que estas serán llenadas más adelante por los datos introducidos desde teclado, pero una ter era solo guardara lo de un arreglo.

Método main

Asignamos una variable de tipo entero **len.**



**strcpy ,** guardara lo que está en el arreglo **después** en el arreglo **res** y ahora lo que está en el arreglo después pasará o se guardara al arreglo antes

Mandaremos dos mensajes en pantalla para ingresar las dos palabras y estas se guardaran en los dos primeros arreglos con ayuda de **Scanf** para leer lo que se haya introducido.

Imprimirá con un mensaje a **antes** seguido del especificador de conversión de cadena **%s** , así como después **%s** y seguido de esto a los arreglos en donde se guardaron las cadenas de carácter de cada una de ellas.

Fin del programa

EJECUCIÓN DEL PROGRAMA



FUNCIONES DE CADENA: **STRCAT “PROGRAMA 7”**

Concatena el contenido de **<cadena\_origen> al final de <cadena\_destino>**

**strcat** (<cadena\_destino>, <cadena\_origen>)

Arreglos de tipo **char** en donde **origen** se le asigna una cadena de carácter y a **destino** también solo que además de asignar una cadena de carácter, le asignaremos un arreglo de 11 posiciones

Variable de tipo entero **len**

Método main



**strcat,** concatenara el resultado de origen y lo asignara a destino, para ello las posiciones asignadas a destino

Fin del programa

Imprimirá a destino junto con lo que estaba guardado en origen y así de las 11 posiciones, en total solo se ocuparon 9.

EJECUCIÓN DEL PROGRAMA



**PROGRAMA 8**

Escribe un programa que reciba por teclado dos palabras.

Y concatene N veces la segunda palabra a la primer palabra.

Donde N es la longitud de la primera palabra.

Palabra1: 'para'

Palabra2: 'brisas‘

Palabra1: parabrisasbrisasbrisasbrisas

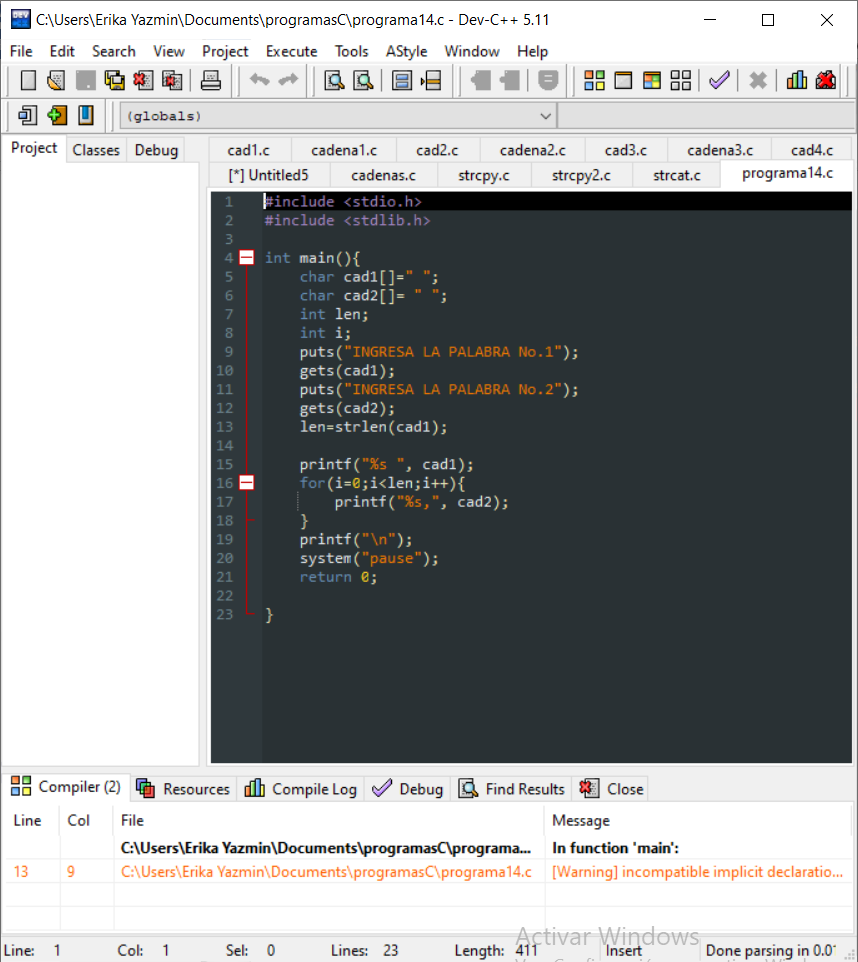
Declarar 2 arreglos de tipo **char** sin posiciones.

Así como dos variables de tipo entero **len** para guardar la longitud de cada palabra ingresada e **i** , para inicializar la variable utilizada dentro de nuestro **for.**

Método main

**puts,** Manda el mensaje en pantalla

**gets,** retorna la cadena exactamente como la hayan ingresado.

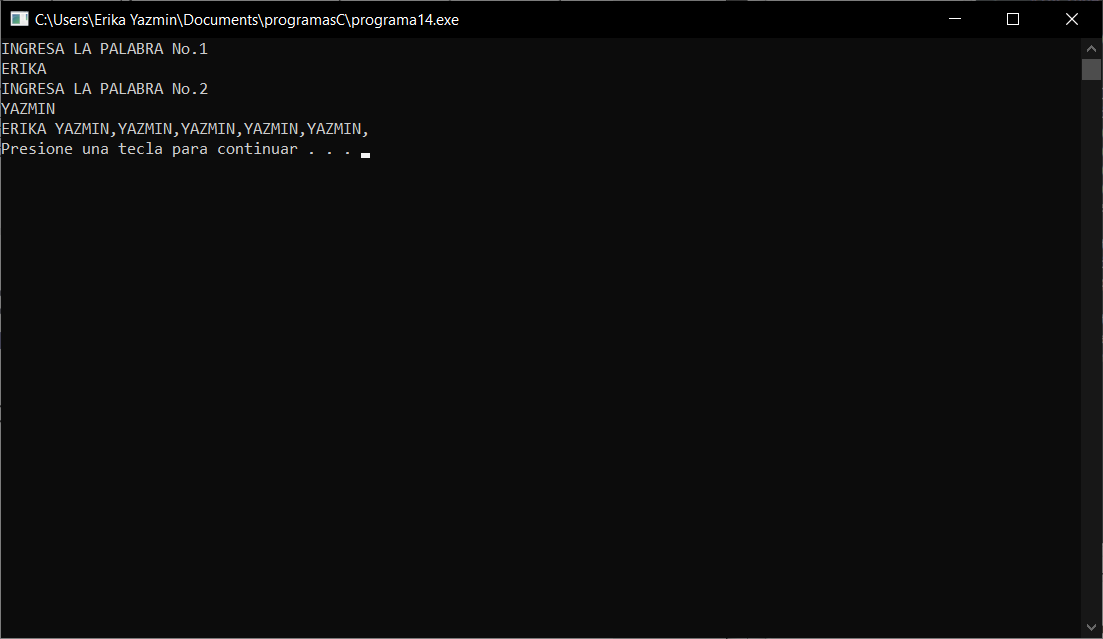


Con base a la longitud de la primer palabra, colocaremos a nuestro ciclo **for**, la cual imprimirá la segunda palabra de acuerdo a las posiciones que la primer palabra contenga.

**Len,** Manda la longitud de la primer palabra

Fin del programa

EJECUCIÓN DEL PROGRAMA



Funciones de cadena: **STRCMP**

Compara las dos cadenas y devuelve un 0 si las dos cadenas son iguales.

Un número negativo si <cadena1> es menor que <cadena2>

Un número positivo (mayor que cero) si <cadena1> es mayor que <cadena2>.

**strcmp (**<cadena1>, <cadena2>)

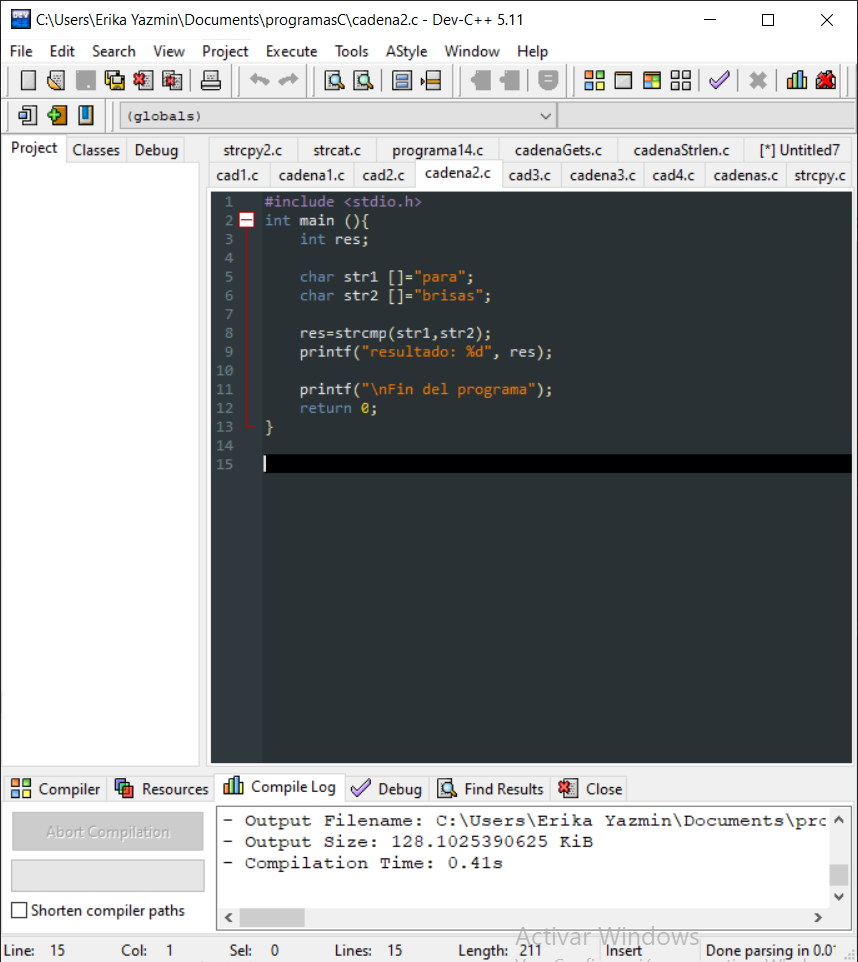
**PROGRAMA 9**

Declarar 2 arreglos de tipo **char,** en donde se le asignara a cada una cadena de carácter.

Crear una variable de tipo entero **res.**

Método main

**res ,** es la variable en donde se guardara el resultado de **strcmp** quecomparara si las cadenas son iguales imprimirá un 0, si la cadena 1 es mayor que la primera imprime un número negativo y si la cadena 1 es mayor que la 2 imprime un número mayor que 0



Imprimirá el resultado final dentro de la variable **res**

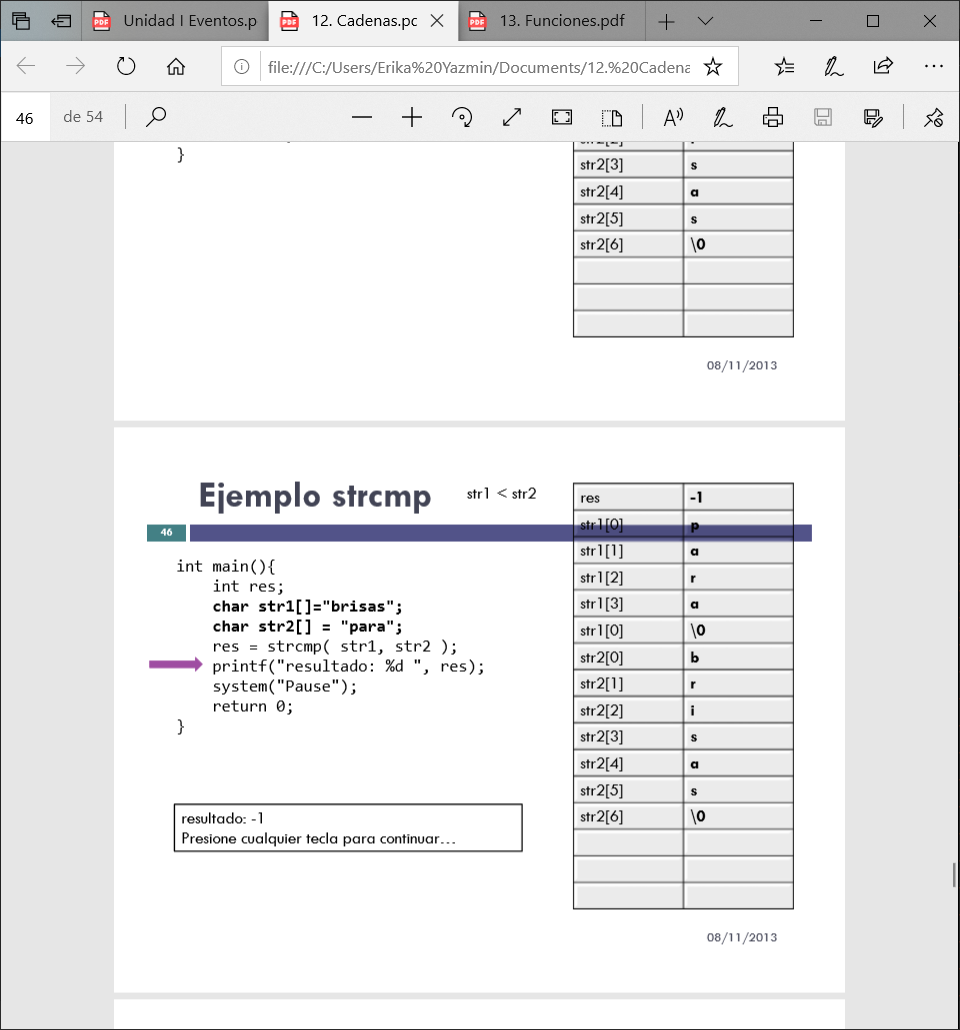
Fin del programa

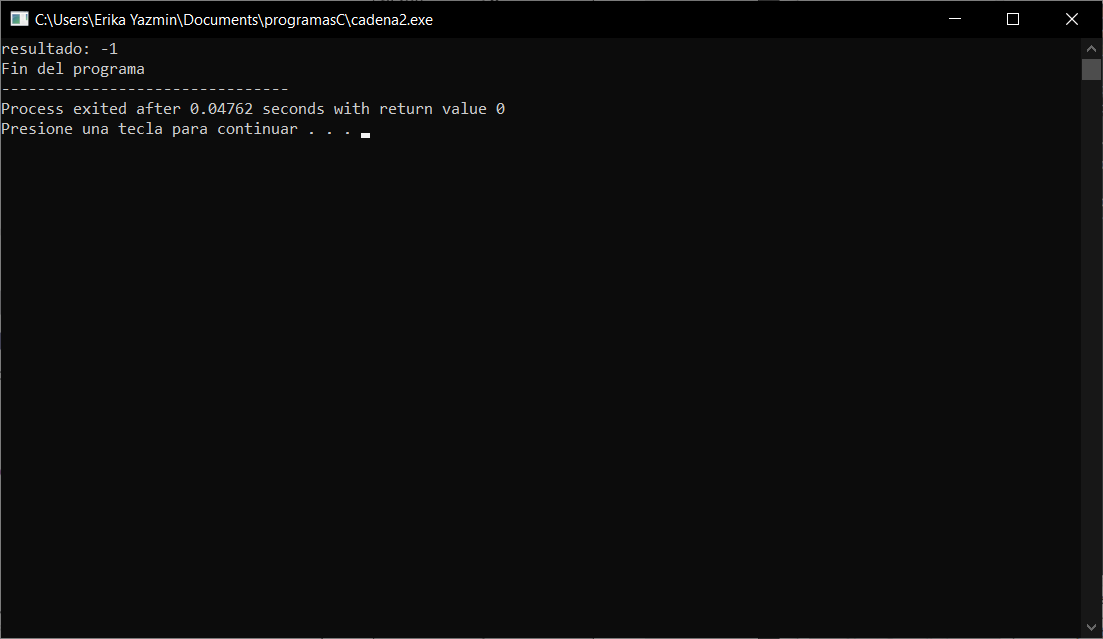
EJECUCIÓN DEL PROGRAMA

En efecto la palabra que esta almacenada en la cadena 1 es menor que la cadena 2.

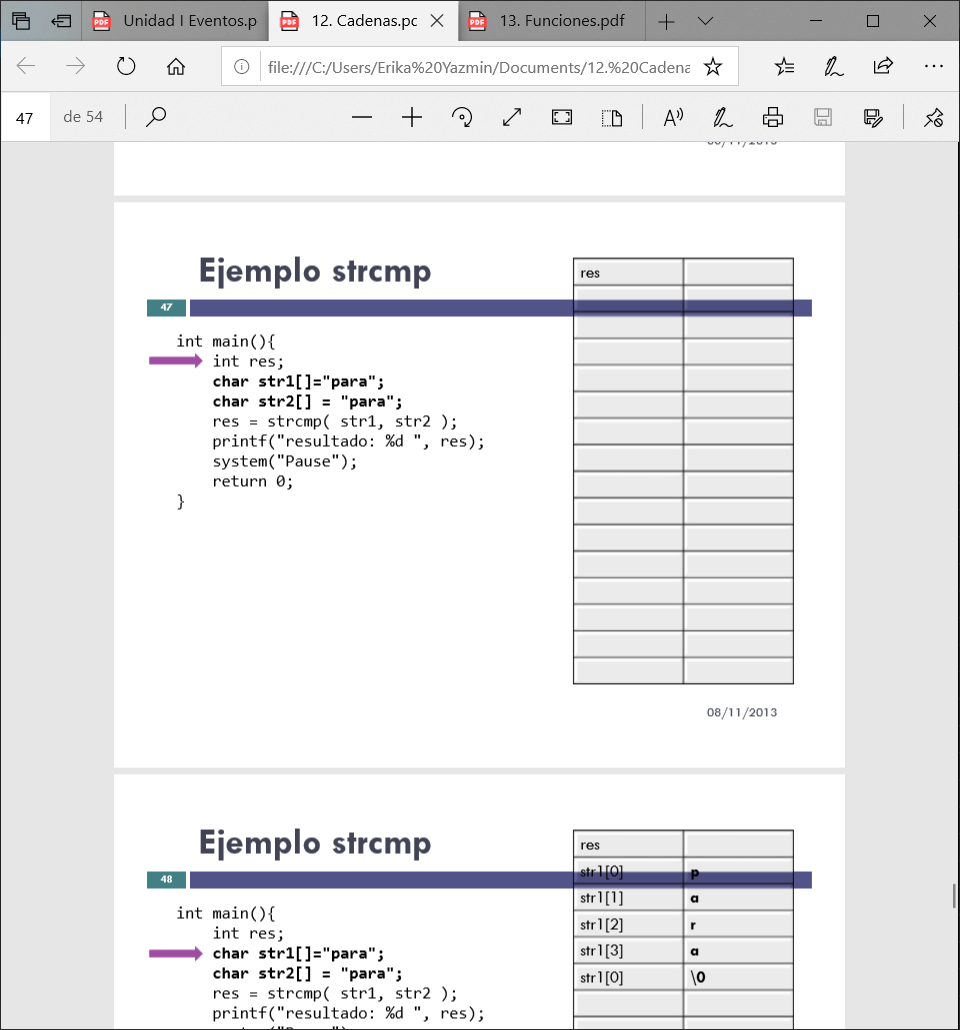


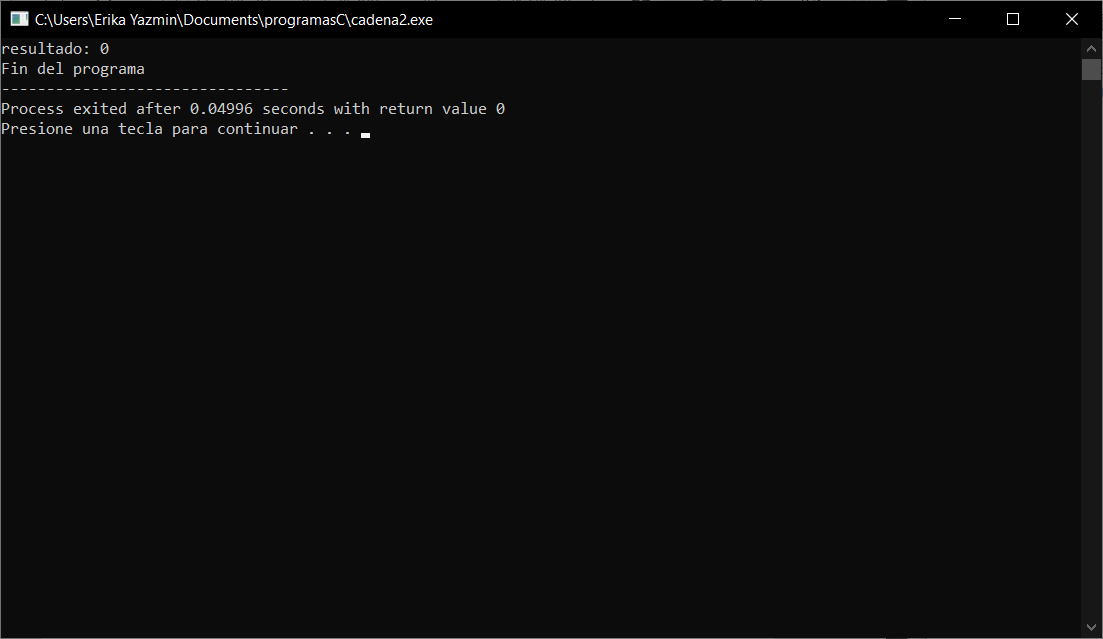
Si colocamos la primera cadena más grande que la segunda en efecto saldrá un número negativo.





Y por último , si colocamos las dos palabras con la misma cantidad de caracteres, saldrá a 0.





**PROGRAMA 10**

Escribe un programa que reciba por teclado dos palabras y te indique cual de ellas es mayor y cual es la menor.

En caso de ser iguales, que imprima la leyenda 'ambas palabras son iguales'.

Compara las dos cadenas y devuelve un 0 si las dos cadenas son iguales.

Un número negativo si <cadena1> es menor que <cadena2>

Un número positivo (mayor que cero) si <cadena1> es mayor que <cadena2>.

UTILIZANDO EL OPERADOR TERNARIO

El operador condicional ?:, también conocido como **operador condicional ternario**, evalúa una expresión booleana y devuelve el resultado de una de las dos expresiones, en función de que la expresión booleana se evalúe como true o false.

La sintaxis del operador condicional es la siguiente:

condition ? consequent : alternative

La expresión **condition** debe evaluarse como true o false. Si condition se evalúa como true, se evalúa la expresión **consequen**t y su resultado se convierte en el resultado de la operación. Si **condition** se evalúa como false, se evalúa la expresión **alternative** y su resultado se convierte en el resultado de la operación. Solo se evalúan **consequent o alternative**.

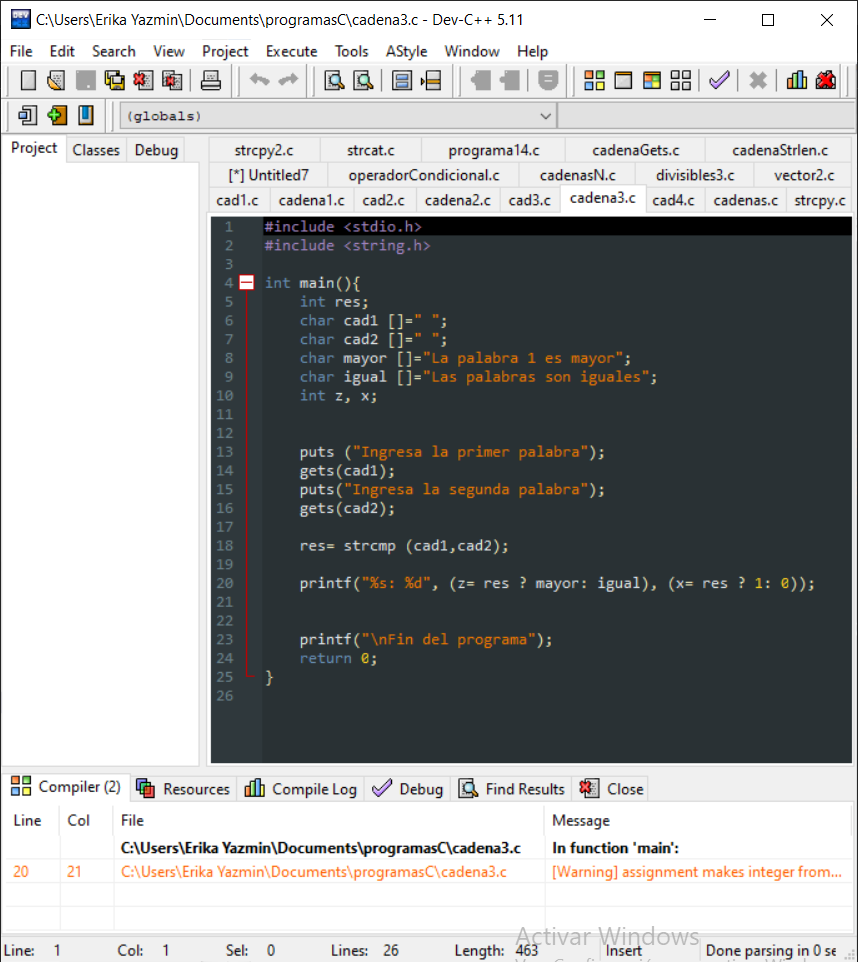
Los tipos de consequent y alternative deben coincidir o debe haber una conversión implícita de un tipo al otro.

Crear 2 arreglos con alguna cadena de carácter, en las cuales se guardaran el mensaje de si la palabra es mayor o son iguales.

Crear 2 arreglos sin tamaño, solo con las comillas para poder recibir alguna cadena como carácter.

Asignar una variable de tipo entero **res**

Método main



Fin del programa

Ahora imprimirá a un valor booleano y a un valor entero por lo que el especificador de conversión es **%s y %d,** y por siguiente al operador ternario será guardado en una variable, seguido de **res,** la variable en donde hicimos la comparación, después “**?”** y seguido de este mandara a llamar a los arreglos en donde guardamos la cadena de carácter de acuerdo a si la primera palabra es mayor o las dos son iguales.

Y por último nuevamente utilizamos el operador ternario pero está vez solo para imprimir un 0 (iguales) o 1 (palabra 1 mayor). Aquí se cumple con lo que habíamos dicho que solo evalúa a dos condiciones.

**puts,** manda el mensaje en pantalla

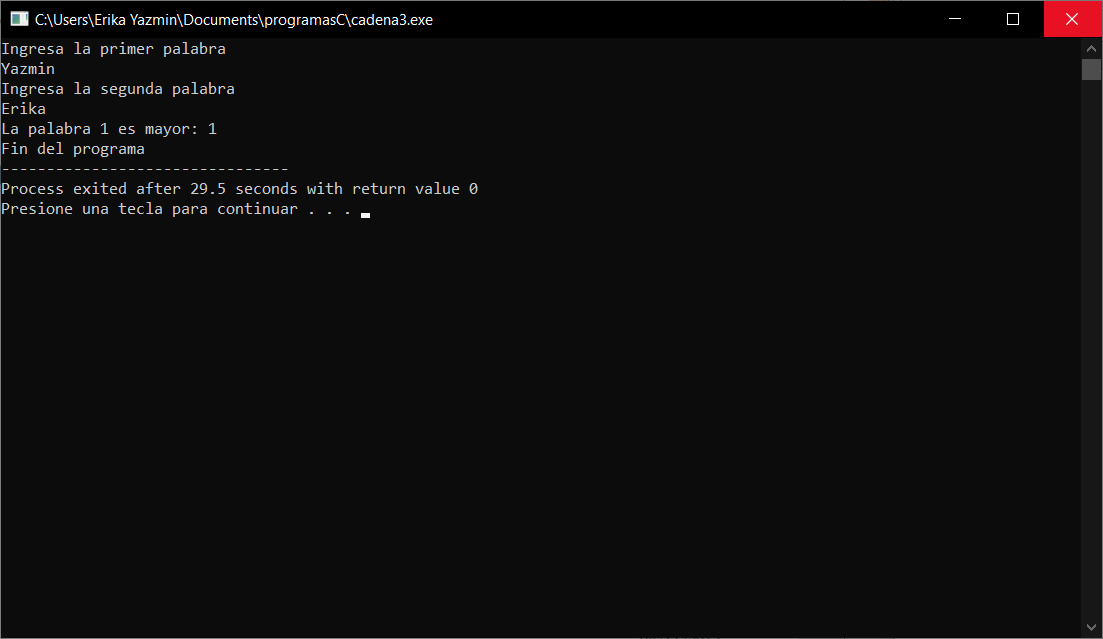
**gets,** se encarga de retornar una cadena exactamente como se encuentre y esta se guardara en los dos arreglos que no tenían tamaño.

**res,** se encarga de guardar el resultado a lo que la cadena **strcmp** logre hacer, por lo que este compara si la cadena 1 y la cadena 2 son iguales manda 0, si la 1 es mayor que la 2 manda un 1 y si la cadena 1 es menor que la dos manda -1

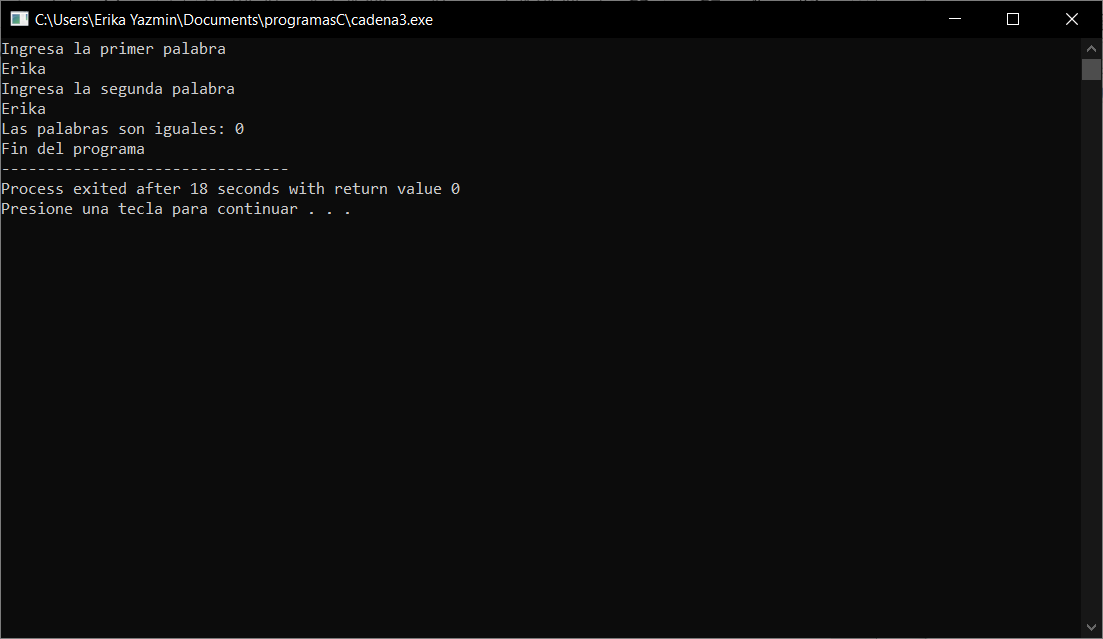
Ahora parar hacer uso del operador ternario, podemos decir que este solo podrá evaluar a dos condiciones **si la palabra 1 es mayor o si las dos palabras son iguales**, ya que si la palabra dos es mayor esta no entra dentro de nuestra condición.

EJECUCIÓN DEL PROGRAMA

*Si la primer palabra ingresada es mayor mandara un 1*



*Si las dos palabras ingresadas son iguales mandara un 0*



*Y si por el contrario ingresas la primer palabra menor que la segunda por ende saldrá error, ya que no estaremos comparando nada*

**PROGRAMA 11**

Crea un programa que detecte una palabra palíndroma.

Los palíndromos son palabras que se leen igual de izquierda a derecha que de derecha a izquierda.

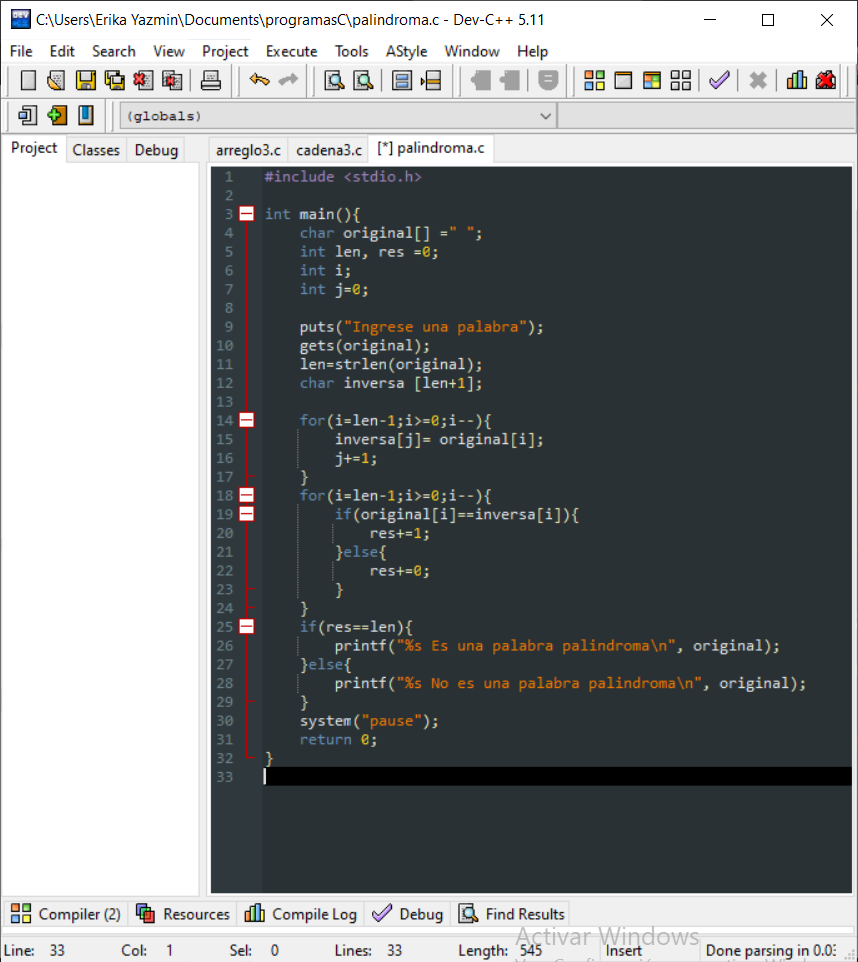
Asignar un arreglo sin tamaño de tipo **char**.

Dos variables para almacenar los resultados de tipo entero.

Dos variables de tipo entero para cada **for.**

 Ejemplo: ala, rotor, salas.

Método main



Fin del programa

Crear un ciclo **for**, inicializando con la longitud de la palabra-1, después recorrerá cada posición si esta es >=0 haciendo un decremento a cada una para hacer la comparación de cada letra de acuerdo a la palabra original

Ahora imprimirá el arreglo junto con la palabra que está almacenada dentro de la posición original.

Crear un nuevo arreglo de tipo **char,** llamado inversa y con el tamaño de la longitud de la palabra +1.

**puts,** Manda el mensaje en pantalla.

**Gets,** Retorna a una cadena.

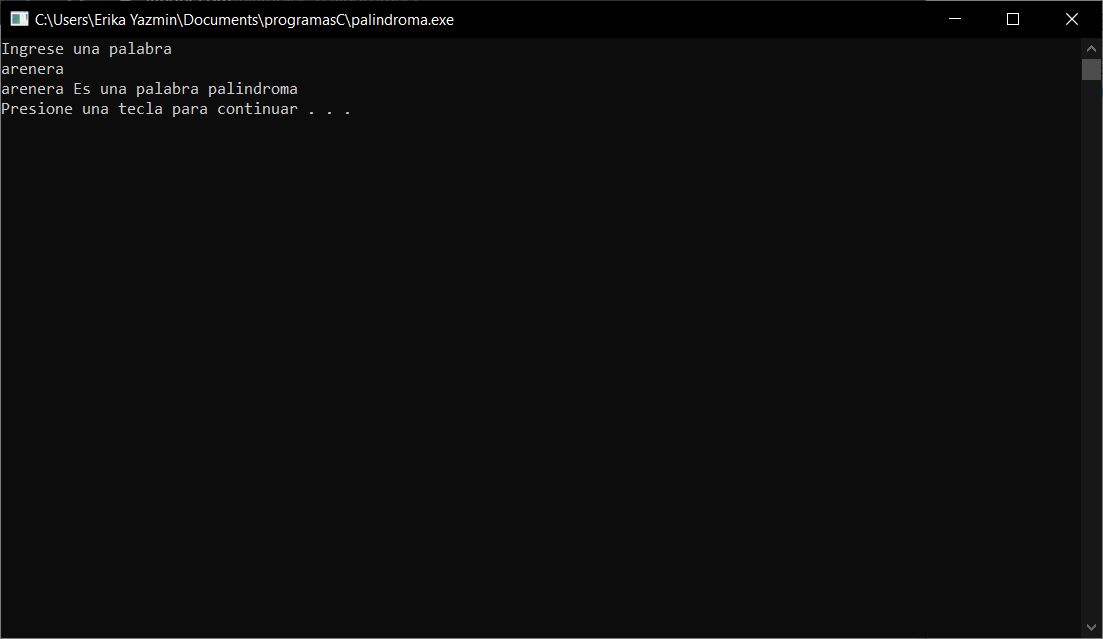
**Len,** es la variable ne donde **strlen,** mide la longitud de la cadena.

Si **res** (0)o (1) es igual a la cadena de forma inversa, entonces imprimirá que es una palabra **palíndroma** y si no entonces no es una palabra **palíndroma.**

Ahora si el arreglo original es igual a la comparación inversa se imprimirá la forma booleana 1 (true) o 0(falso).

Para el segundo ciclo **for**, inicializara con la longitud de la cadena, seguido del decremento de cada posición y esto dependerá la comparación de cada letra si es igual a la que está dentro del arreglo.

EJECUCIÓN DEL PROGRAMA



**PROGRAMA 12**

Crea un programa que cuente cuantas ocurrencias de cada letra contiene una palabra.

Ejemplo:

Palabra

P→1

a →3

l →1

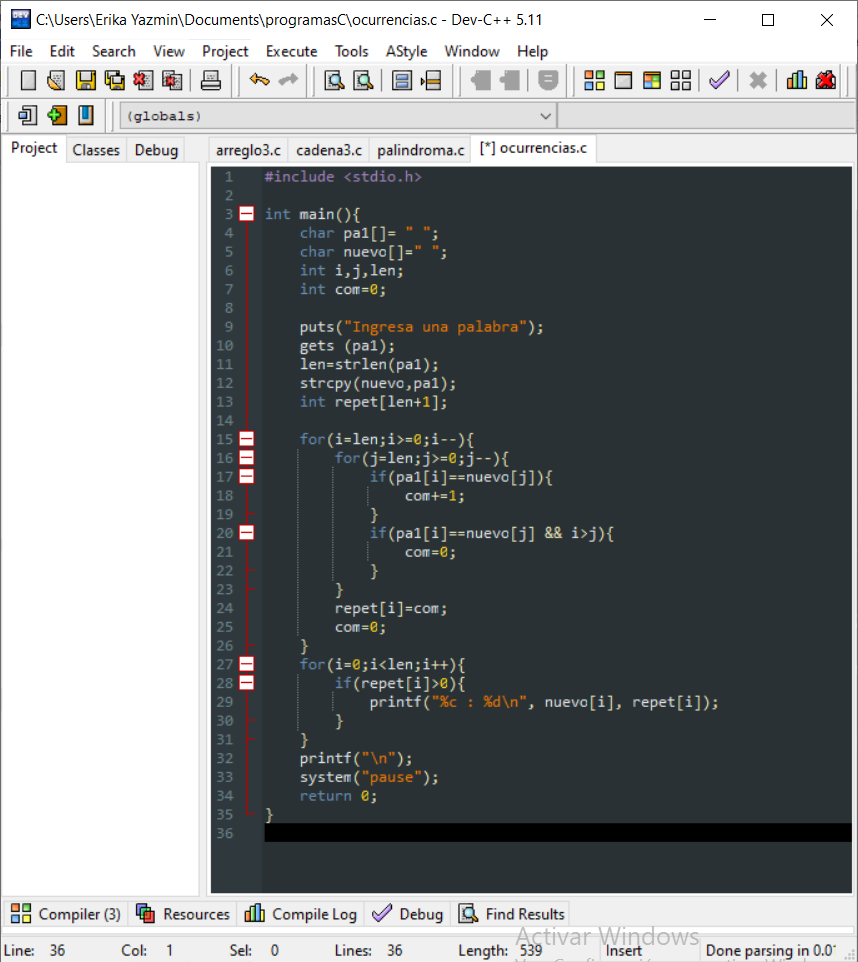
b →1

r →1

Crear dos arreglos sin tamaño de tipo **char**.

Declarar 4 variables de tipo entero en la que se encuentran las variables inicializadas para los ciclos **for**, y las demás para guardar la longitud de la palabra.

Método main



**Puts,** manda un mensaje en pantalla.

**gets,** retorna a la cadena

**len,** guarda la longitud de la palabra.

**strcpy,** copia el contenido de la cadena y lo guarda en un nuevo arreglo.

Crear un nuevo arreglo con la el tamaño asignado de la longitud de la cadena.

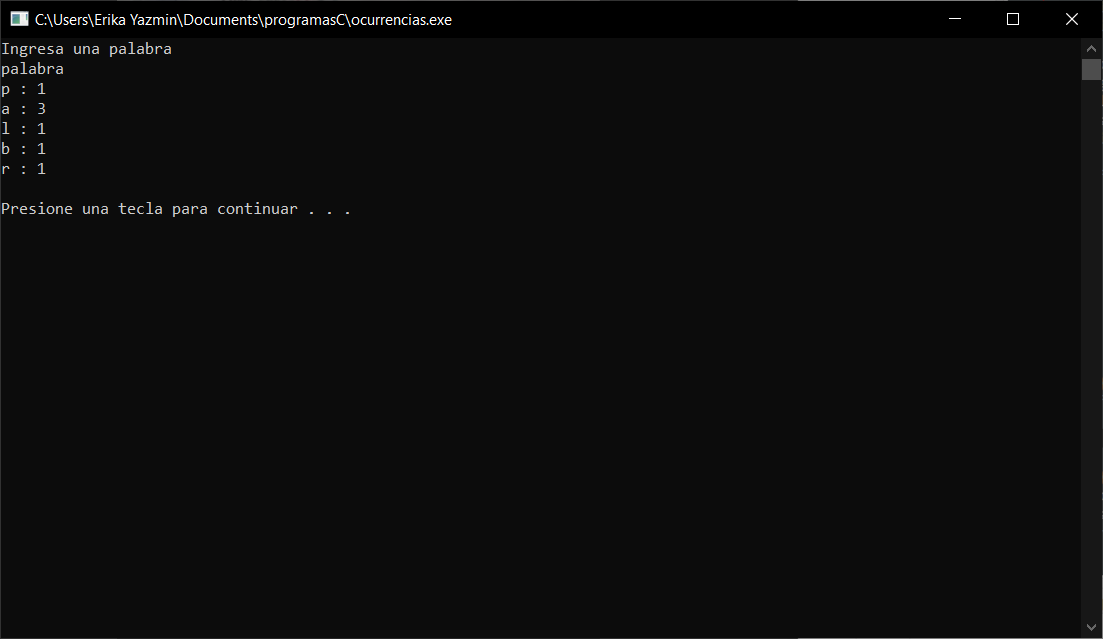
Declarar a un for anidado, para corroborar la posición y asignación de la cadena.

Si esta cadena en la posición 0 es igual a la comparación de la misma en otro arreglo, entonces mandará un resultado booleano (true o falso)

Ahora si la posición es menor que la longitud y si esta es mayor a 0, imprimirá la repetición de cada palabra junto con las veces repetidas de acuerdo a la posición y comparación realizada a la cadena.

Fin del programa

EJECUCIÓN DEL PROGRAMA



**LAMINA 13. FUNCIONES**

***Definición de función matemática***

Matemáticamente una función es una **operación** que toma uno o más valores llamados **argumentos** y produce un valor llamado **resultado**.

Una función es un bloque de código reconocido por un identificador que realiza un trabajo específico.

Su propósito es dividir los programas en módulos manejables separados (divide y vencerás).

**Ventajas**

1. Facilita el diseño descendente.

2. Los procedimientos dentro de ellas se pueden ejecutar varias veces.

3. Facilita la división de tareas.

4. Se pueden probar individualmente

5. Con funciones apropiadamente diseñadas, es posible ignorar como se realiza una tarea, sabiendo qué es lo que hacen.

**Modo de uso**

1. Funciones diseñadas para realizar operaciones a partir de sus argumentos y devolver un valor basado en sus cálculos.

2. Funciones que no reciben argumentos, realizan un proceso y devuelven un valor.

3. Funciones que no tienen argumentos ni valor de retorno explícito, operan sobre el entorno de variables globales o atributos del sistema operativo.

EJEMPLO DE FUNCIONES

|  |  |
| --- | --- |
| int suma (int a, int b){  int c; c = a + b;  return c;  } | float promedio(int a, int b, int c){  int sum = a + b + c;  float prom = sum/3;  return prom;  } |

***DECLARACIÓN DE FUNCIONES***

El formato para la declaración de funciones es:

**tipo** nombre\_funcion (lista de parámetros) {

cuerpo de la función

}

**tipo** : especifica el tipo de valor que devuelve la función. Si no se especifica tipo, el compilador asume que es entero (int).

***DECLARACIÓN DE FUNCIONES 2***

tipo nombre\_funcion(**lista de parámetros**){

cuerpo de la función

}

**lista de parámetros :** es la lista de nombres de variables separados por comas con sus tipos asociados que reciben los valores de los argumentos actuales de la llamada a la función

Entre llaves se encuentra el cuerpo de la función.

***DECLARACIÓN DE FUNCIONES (3)***

* Forza la salida inmediata de la función en que se encuentra.
* Una función puede retornar valor sólo cuando el tipo de retorno no es void.
* Devuelve un valor a la función que realizó la llamada.
* return (expresión);

***DECLARACIÓN DE FUNCIONES (4)***

* Tradicionalmente en C se declaran como prototipos al inicio del programa. Después se declara la función main, y después se hace la declaración formal de las funciones.
* También pueden declararse las funciones al inicio del programa y después declarar la función main sin declarar prototipo.

EJEMPLO DE FUNCIIONES

|  |  |
| --- | --- |
| int suma (int a, int b){  int c;  return int  c = a + b;  { | float promedio(int a, int b, int c){  int sum = a + b + c;  float prom = sum/3;  float prom;  } |

*LLAMADAS A FUNCIONES*

Para llamar a una función se especifica su nombre y la lista de argumentos sin poner el tipo de dato.

*Nombre\_funcion (var1,var2,varn);*

En una llamada habrá un argumento real por cada argumento formal, respetando el orden de declaración.

**Argumento formal:** Los que aparecen en la definición de la función.

**Argumento real:** Los que se pasan en la llamada a la función.

*PASO DE PARÁMETROS POR VALOR*

* Se hace una copia del valor del argumento en el parámetro formal.
* La función opera internamente con estos últimos.
* Los parámetros formales se crean al entrar a la función y se destruyen al salir de ella, cualquier cambio realizado por la función en los parámetros formales no tienen ningún efecto sobre los argumentos.

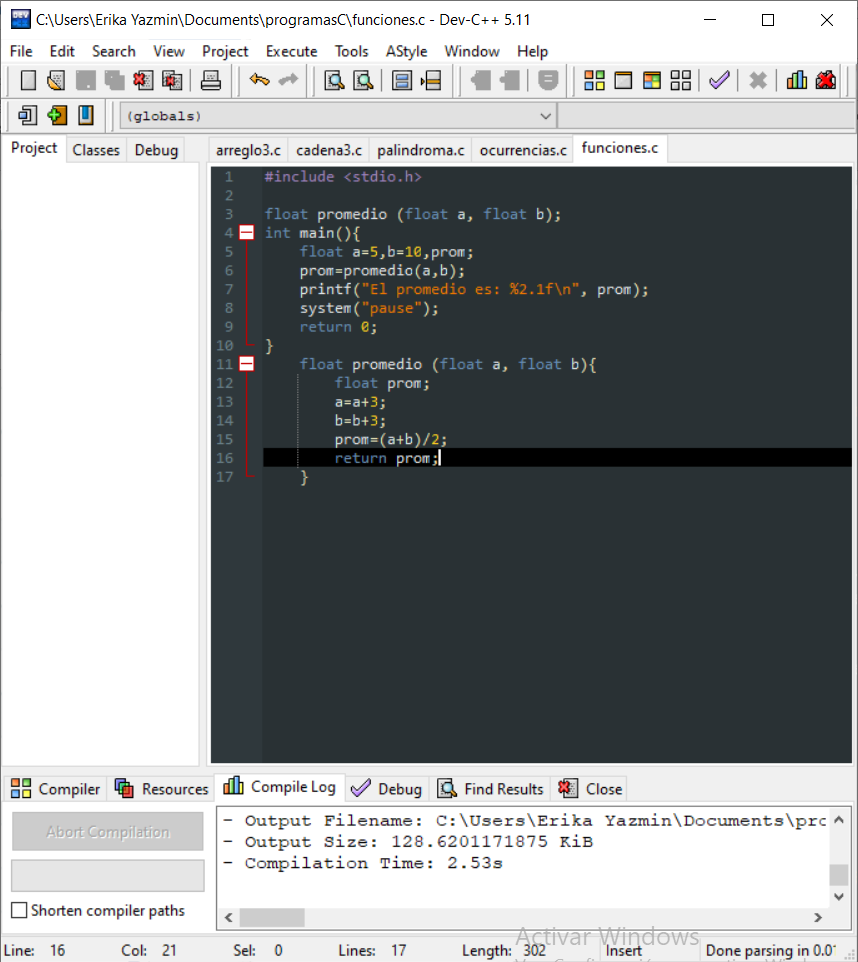
Variable de tipo **float**, llamado **prom** en donde guardara al método **float promedio** llamando consigo a sus argumentos inicializados en un principio., teniendo en mismo orden cada argumento

**PROGRAMA 1**

Declaración de dos variables de tipo **float** asignando un valor numérico a cada una de ellas

Declaración de la función con nuestros argumentos

Método main



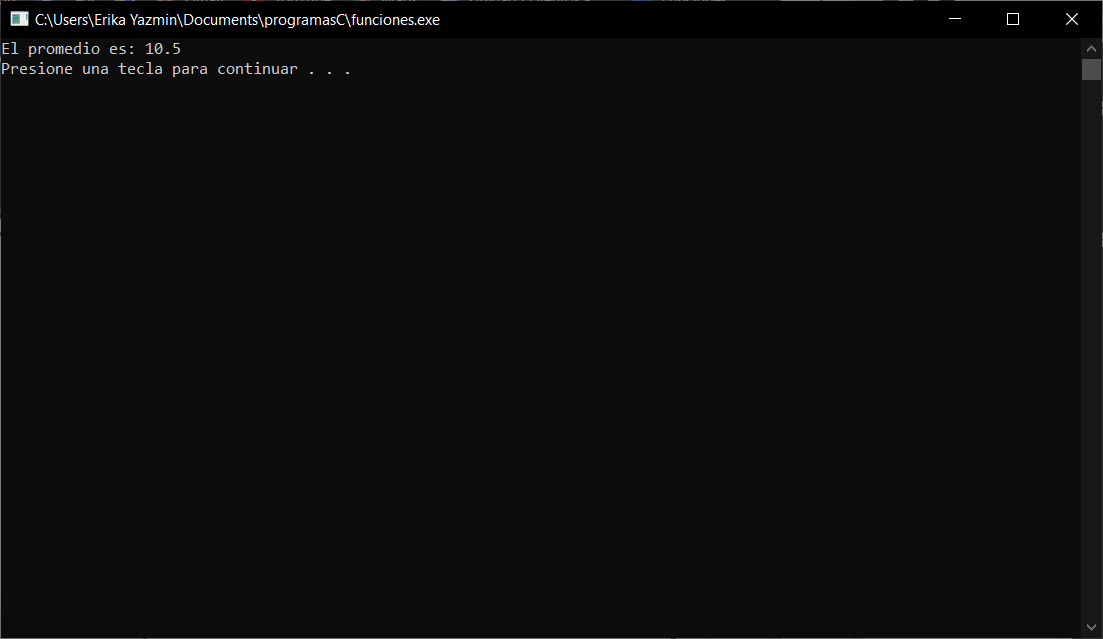
Ahora que la variable retorno con el valor de la operación, está solo imprime diciéndole que recibirá a un valor de 2 enteros y 1 decimal (2.1), seguido del especificador de conversión y la variable donde se encuentra el resultado.

En el método **float promedio** asignará a una nueva variable con el mismo nombre que la primera pero en este caso llamará a los argumentos principales tomándolos como acumuladores de valor. Y así asignándole un nuevo valor a cada uno.

En la variable **prom** se realizará la operación para calcular el promedio y esto retornará a **prom** ya con la operación realizada.

Fin del programa

**EJECUCIÓN DEL PROGRAMA**



**Variables locales y globales**

**Variables Locales:**

Se declaran dentro de la función y sólo están disponibles durante su ejecución.

Se crean cuando se entra en ejecución una función y se destruyen cuando se termina.

**Variables globales:**

Se declaran fuera de las funciones. Pueden ser utilizadas por todas las funciones.

Existen durante toda la vida del programa.

PROGRAMA 2

Escribir una función que se llame máximo que reciba dos número por parámetros y que regrese el mayor de ellos.

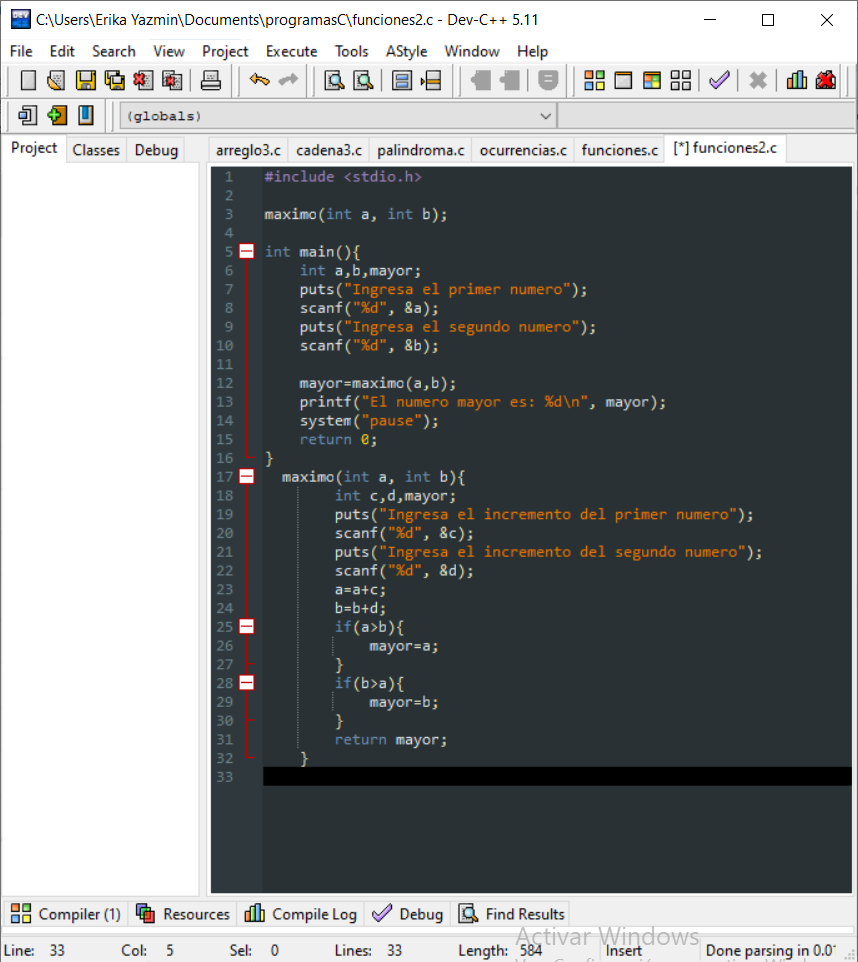
**puts,** manda el mensaje en pantalla.

**scanf,** Lee datos del teclado de acuerdo a un formato especificado , y asigna los datos de entrada a una o más variables del programa.

Declaración de la función con sus argumentos.

Declaración de tres variables de tipo entero.

Método main



Utilizaremos a los argumentos principales declarados en la función y guardaremos en cada uno la suma de los datos especificados en cada mensaje.

A las variables asignadas anteriormente guardaran un nuevo número para sumárselo a los valores iniciales.

En la segunda función la declararemos como método contando con sus argumentos.

Declararemos a tres nuevas variables de tipo entero **c ,d** y **mayor (**donde guardará el resultado final para retorno**).**

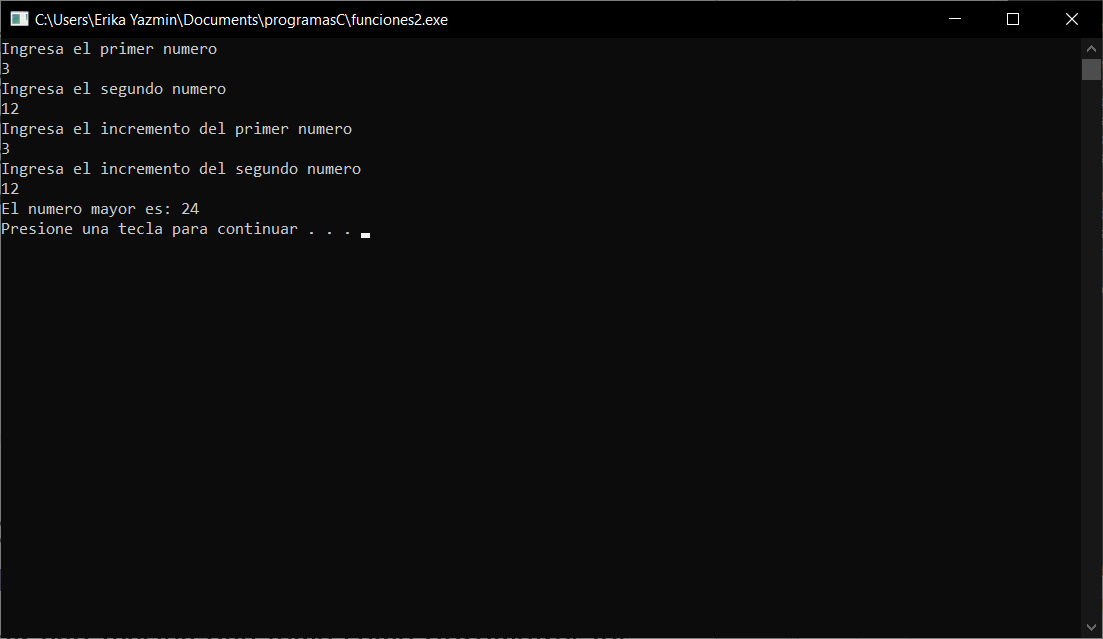
Variable de tipo **entero mayor,** guardando al método de tipo **entero** **máximo** y llamando consigo a sus argumentos al mismo orden en el que se encuentra la función inicial.

Condicionaremos **si**, el primer valor al sumarlo es mayor que el segundo y en la variable **mayor** se guardará el argumento que resulto mayor para finalmente retornarlo e imprimir el resultado.

Fin del programa

Ahora que la variable **mayor** retorno a su misma variable de arriba, imprimirá el resultado obtenido a base de las operaciones realizadas en el método **máximo.**

**EJECUCIÓN DEL PROGRAMA**



Número mayor

**PROGRAMA 3**

Escribir una función que reciba caracteres del teclado hasta recibir un espacio o un salto de línea (enter) y a continuación mostrar todos los caracteres en orden inverso.

Ejemplo:

Entrada: Hola

Salida: aloH

**printf,** imprime el mensaje en pantalla.

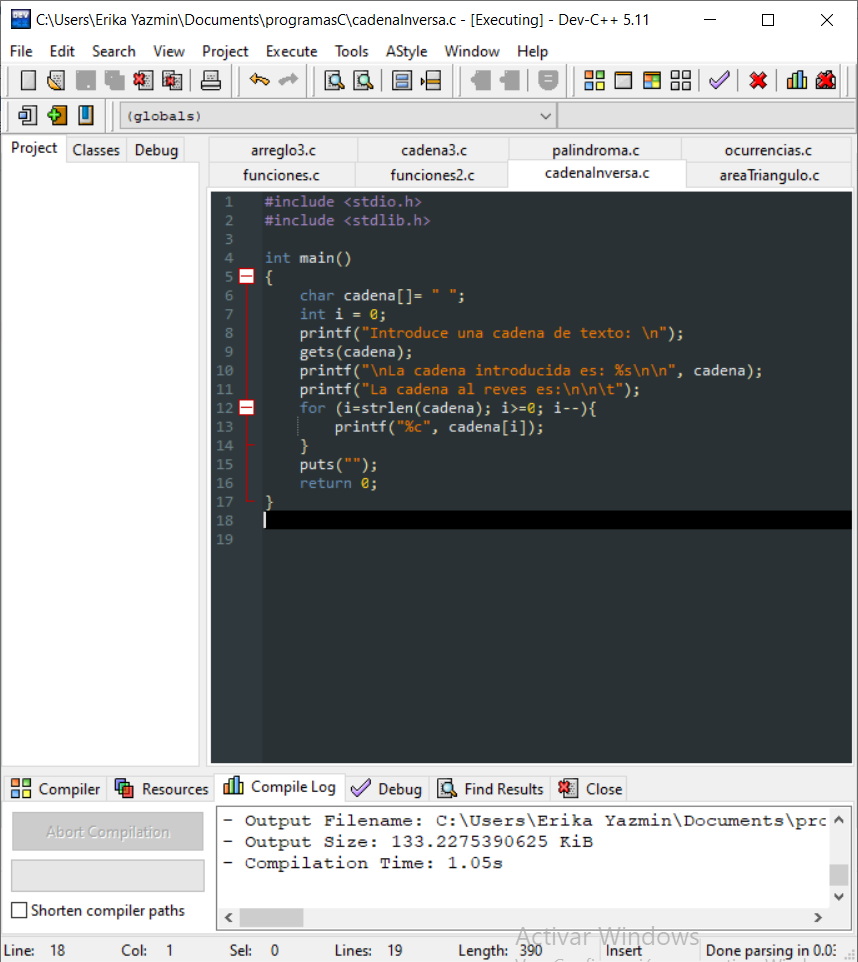
**gets,** retorna a la cadena.

Declaración de un arreglo vacío de tipo **char.**

Declaración de una variable de tipo **entero** inicializada en 0, para **for**

Método main

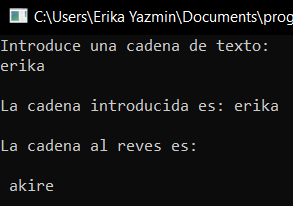
Declaración de la condición **for**, en donde la inicialización de su variable comienza con la longitud de la cadena, si esta es >=0 se realizará un decremento y comparara la posición de está para imprimirla de forma descendente.



Imprimirá el resultado final de la posición original, seguido de la palabra inversa.

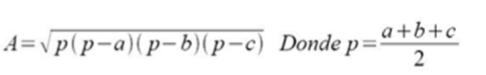
Fin del programa

**EJECUCIÓN DEL PROGRAMA**



**PROGRAMA 4**

Escribir una función que tome como parámetros las longitudes de los tres lados de un triángulo (a, b, c) y devuelva el área del triángulo.



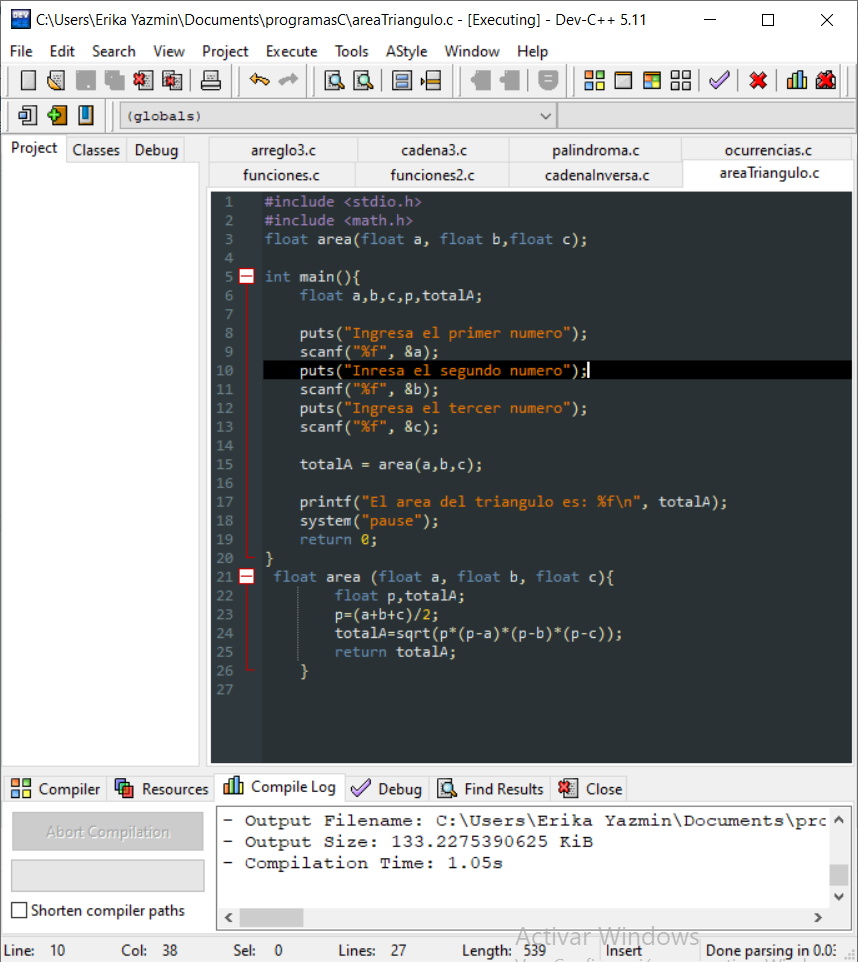
**puts,** Manda el mensaje en pantalla.

**scanf,** Lee datos del teclado de acuerdo a un formato especificado , y asigna los datos de entrada a una o más variables del programa.

Declaración de las 4 variables en donde se guardaran los resultados obtenidos a la entrada de datos.

Declaración de la función **área** con sus argumentos **a, b y c**

Método main



Una vez retornado el resultado final, este imprimirá la opción conveniente haciendo uso de la función y cada uno de los argumentos

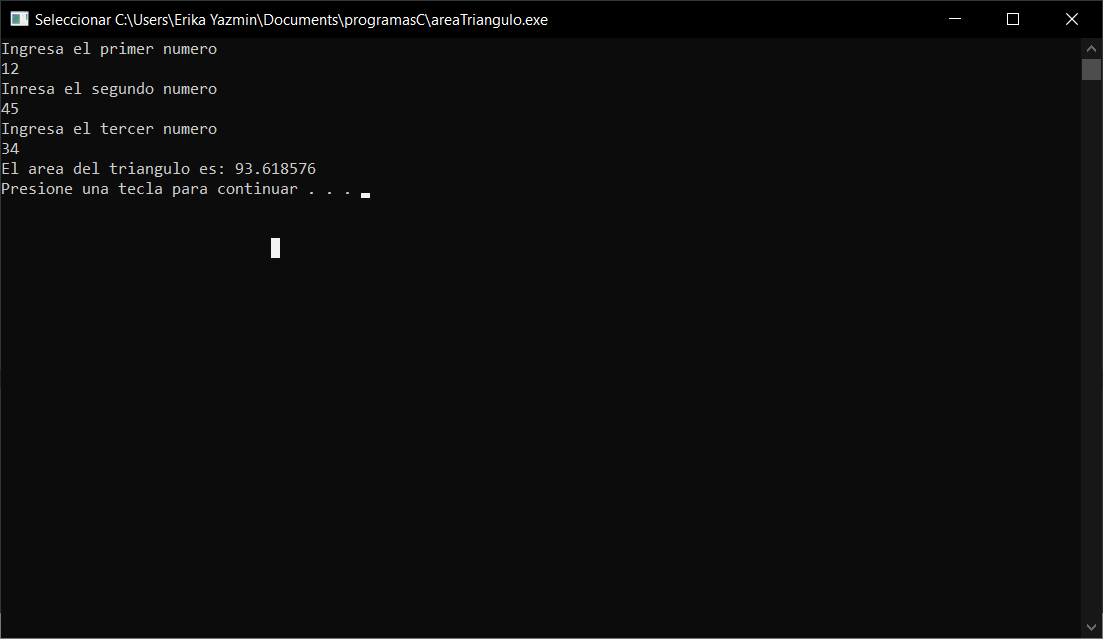
Variable encargada de retornar el resultado final del área del triangulo

Declaración de la función como método junto con cada uno de sus argumentos declarados al inicio.

Así como la declaración de la variable **totalA** donde guardará el resultado final y retornará para poder imprimir el resultado, seguido de la variable **p** donde almacenará la operación de **p=a+b+c/2** y así poder mandarla a llamar en el área total.

Fin del programa

**EJECUCIÓN DEL PROGRAMA**



***FUNCIONES RECURSIVAS***

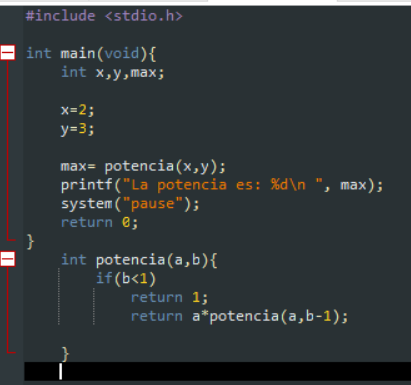
Se llaman funciones recursivas a aquellas que se llaman a sí mismas de forma repetida hasta que se cumpla alguna condición.

Cada llamada implica el almacenamiento de variables de estado y otros parámetros

Declaración de 3 variables de tipo entero, donde dos de ellas se le asignara un valor numérico. Y la tercera será el resultado final una vez hecha la operación en la declaración del siguiente método.

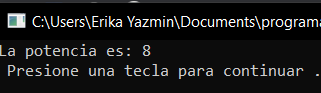
**PROGRAMA 5**

Método main



En el método declarado llamado **potencia** se le asignarán dos argumentos que tomaran el lugar de los declarados arriba (x=a,y=b), seguido de esto se declarará a la condición **if** en donde condicionará si **b** es menor que 1, si esto se cumple retornará haciendo una operación en donde **potencia** restará 1 a b y esto consecutivamente hasta que la condición se cumpla y pueda retornar nuevamente a 1, para así poder realizar la potencia, multiplicando **a\*b de forma inversa** e ir guardándolo en **potencia** para al final solo imprimir el resultado.

**EJECUCIÓN DEL PROGRAMA**



**PROGRAMA 6**

**Haz un programa con funciones recursivas que calcule el factorial de un número n ingresado desde teclado.**

**puts**, manda el mensaje en pantalla.

**scanf,** Lee datos del teclado de acuerdo a un formato especificado , y asigna los datos de entrada a una o más variables del programa.

**Ej. N = 5**

**5! = 4! \* 5**

**4! = 3! \* 4**

**3! = 2! \* 3**

Declaración de dos variables de tipo **entero** en donde guardara el número y factorial del mismo

**2! = 1! \* 2**

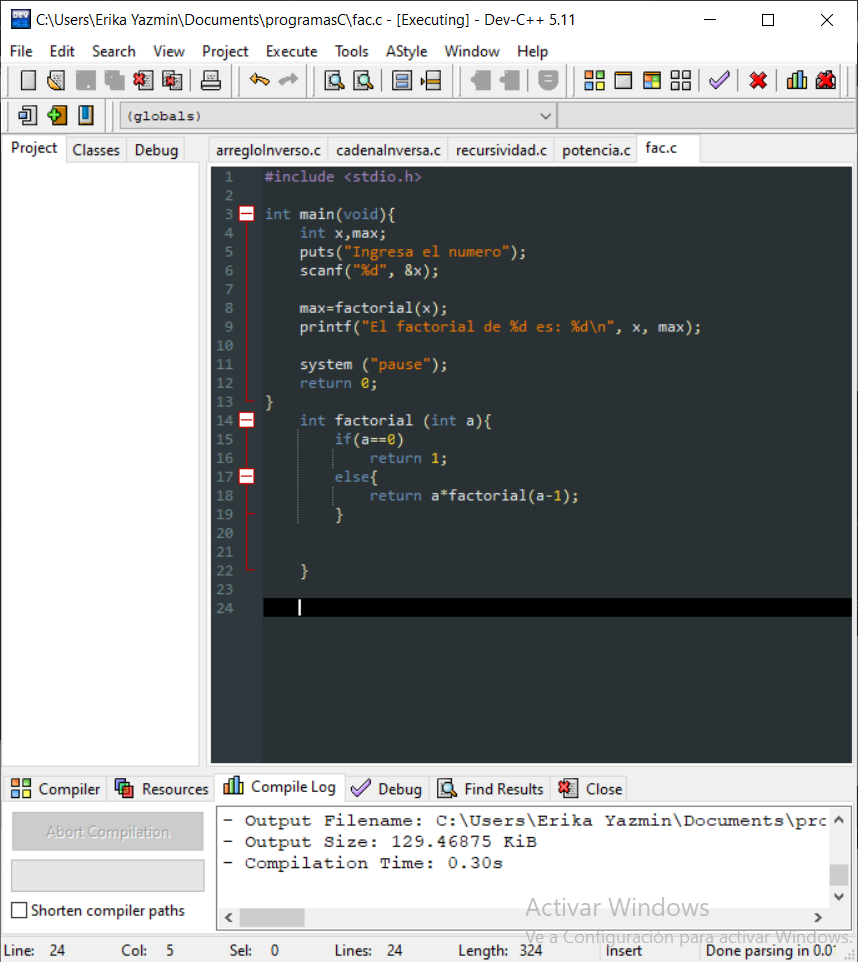
**1! = 0! \* 1**

**0! = 1**

Método main

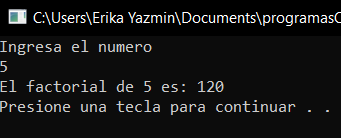
En el método llamado **factorial**, declarar un argumento.

Seguido de esto declarar la condición **si** el argumento = 0, entonces retornara a 1, **si no** realizara la condición hasta que esta se cumpla y finalmente multiplicarlo inversamente y esto retornarlo a **max** donde imprimirá el número y el factorial de este.



Fin de programa

EJECUCIÓN DEL PROGRAMA



**IV. CONCLUSIONES:**

Como conclusión podemos decir que nuestro manual de prácticas así como los temas abordados dentro de él, realmente estuvieron muy bien explicados, todo esto a base de las distintas formas en las que una cadena puede estar adaptada o en otro caso, la forma en la que cada uno de los programas abordados lo pedía, para ello a la hora de abordar cada una de los temas tuvimos que corroborar algunos significados o funciones que tal vez no existían dentro de nuestra fuente de información, pero con la finalidad de que pudiéramos darnos cuenta que las cadenas pueden estar relacionadas de distinta forma y cada forma conlleva a una manera distinta de saber como colocarla dentro de nuestro programa.

Para finalizar, diremos que los programas estuvieron un poco complejos, pero a base de los ejemplos y la buena investigación pudimos llegar al resultado esperado.