

Instituto Politécnico Nacional

Escuela Superior de Cómputo

Práctica No 1. Manejo de Arreglos y Cadenas

Asignatura: Estructuras de Datos

Profesor: Daniel Cruz García

Alumnos:

* Juárez Bouchan José Leonardo
* Martínez Miranda Erubey Gustavo
* Rodríguez Rosas Daniel

Grupo: 1CV4

Introducción

Un arreglo es una colección finita y constante (que no cambia su tamaño) de elementos del mismo tipo, es también es conocido como vector, matriz, array o lista, dependiendo de la dimensión del arreglo.

Se pueden crear arreglos de cualquier tipo de dato en C, es decir, tipos de dato simple, estructurado, archivos e incluso de apuntadores, pero es más común utilizarlos para manipular colecciones de datos simples.

Cada ítem del arreglo se le denomina elemento del arreglo. Todos los elementos que constituyen un arreglo están ordenados de manera secuencial en memoria del programa y se encuentran numerados consecutivamente comenzando en **0** hasta **n-1** donde:

**0:** es el primer elemento del arreglo,

**n-1:** es el último elemento del arreglo y

**n:** es el tamaño total del arreglo.

A los valores de **0** a **n-1** se les denomina índices o subíndices del arreglo. Y al igual que otra variable primero se debe declarar antes de utilizarse.

1. **Arreglos Unidimensionales**

La sintaxis para declarar un arreglo unidimensional es:

<tipo\_de\_dato> identificador[tam\_arreglo];

Dónde:

<tipo\_de\_dato>: Es el tipo de dato con el que se van a definir todos los elementos que van a formar parte del arreglo.

identificador: Es nombre que va a recibir el arreglo.

[tam\_arreglo]: Los corchetes “[]” determinan que el tipo de dato que se está definiendo es un arreglo y tam\_arreglo es el tamaño del arreglo.

Por ejemplo:

int arreglo[ 10 ];

En el ejemplo anterior se está definiendo un arreglo unidimensional llamado arreglo y tiene un tamaño de 10, es decir, el arreglo está constituido de 10 variables enteras. Esta declaración hace que el compilador le indique al sistema operativo que reserve espacio suficiente en memoria para almacenar 10 valores enteros.

1. Arreglos Multidimensionales

Los arreglos multidimensionales son los que tienen más de una dimensión, los más usuales son los de ***dos dimensiones*** que también conocidos como **tablas o matrices**.

Un arreglo de dos dimensiones tiene 2 índices que nos ayudan a ubicar un elemento dentro del arreglo.

La sintaxis para la declaración de un arreglo de dos dimensiones seria:

<tipo\_dato> identificador [tam\_filas][tam\_columnas];

Por ejemplo:

int datos[3][3];

Esta línea de código declara un arreglo en dos dimensiones de 3 X 3 elementos, es decir, este arreglo contendrá en su interior 9 elementos de tipo entero. Para acceder a alguno de sus elementos se utilizan los dos índices, es decir:

datos[0][2] = 12;

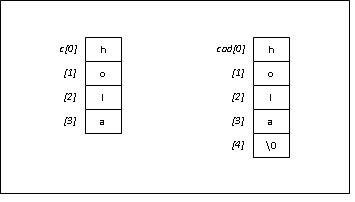
datos[2][0] = 0;

datos[2][2] = 30;

Por lo tanto, a diferencia de un arreglo unidimensional donde solo se utiliza un índice para acceder a sus elementos, para un arreglo bidimensional se utilizan dos índices.

1. **Cadena de caracteres**

A diferencia de otros lenguajes de programación que emplean un tipo de dato denominado cadena string para manipular un conjunto de símbolos, en C, se debe simular mediante un arreglo de caracteres que contiene al final el carácter nulo (‘\0’), esta es la razón por la que es necesario declarar los arreglos con un carácter más que la frase más grande. Por ejemplo, a continuación se muestra la diferencia entre un arreglo de caracteres y una cadena de caracteres.



Cadena de

caracteres

Arreglo de

caracteres

La manera de definir una cadena es la siguiente:

char <identificador> [<longitud\_máxima>];

Dónde:

char: Es el tipo de dato con el que se debe definir la cadena.

identificador: Es nombre que va a recibir la cadena.

[<longitud\_máxima>]: Los corchetes “[]” determinan que el tipo de dato será un arreglo y longitud\_máxima es el tamaño de la cadena.

Por ejemplo:

char cadena[ 10 ];

El lenguaje C no maneja cadenas de caracteres, como se hace con enteros o flotantes, por lo que se usan funciones especiales que permiten manipular y hacer operaciones con ellas.

**Manejo de cadenas**

**string.h** es un archivo de la biblioteca estándar que contiene la definición de macros, constantes, funciones y tipos de utilidad para trabajar con cadenas de caracteres.

Las funciones declaradas en **string.h** funcionan en cualquier plataforma que soporte ANSI C. Sin embargo, existen algunos problemas de seguridad con estas funciones, como el desbordamiento de arreglos. Para poder utilizar las funciones del manejo de cadenas es importante preservar el carácter de terminación **NULL ‘\0’**, ya que con éste es como C define y maneja las longitudes de las cadenas.

La siguiente tabla muestra algunas funciones para el manejo de cadenas, sin embargo, existen más funciones. Para mayor referencia googlear string.h

|  |  |
| --- | --- |
| **Nombre de la función** | **Descripción** |
| **strcat()** | char\* strcat (char\* destino, const char\* fuente)  Añade la cadena fuente al final de la destino. Devuelve la cadena destino. |
| **strlen()** | size\_t strlen( const char\* s )  Devuelve la longitud de la cadena s |
| **strncpy()** | char\* strncpy( char\* dest, const char\* fuente, size\_t n)  Copia n caracteres de la cadena fuente a la cadena destino |
| **strncat()** | char\* strncat( char\* s1, const char\* s2, size\_t n)  Añade los primeros n caracteres de S2 a S1. Devuelve s1 si n>=strlen(s2), entonces strncat tiene el mismo efecto que strcat. |
| **strchr()** | char\* strchr( const char\* s1, int ch)  Devuelve un apuntador a la primera ocurrencia de ch en s1, devuelve NULL si ch no se encuentra en s1. |
| **strcmp()** | int strcmp( const char\* s1, const char\* s2)  Compara alfabéticamente la cadena s1 con s2 y devuelve un dato de:  cero si s1 == s2  menor que cero si s1 < s2  mayor que cero si s1 > s2 |
| **strcspn()** | size\_t strcspn( const char\* s1, const char\* s2)  Devuelve la longitud de la subcadena más larga de s1 que comienza con el carácter s1[0] y no contiene ninguno de los caracteres de la cadena s2. |

Otras funciones útiles que permiten el uso fácil de cadenas son gets y puts.

* gets( cadena ) captura una cadena de caracteres de la entrada estándar y la coloca en el arreglo de cadena, un salto línea de o “enter” marcan el fin de la captura, pero el salto de línea no es añadido a la cadena, en su lugar coloca el carácter de fin de cadena**‘\0’**. Cabe mencionar que gets es una función que permite capturar cadenas de caracteres con espacios, cosa que no se podría hacer con la función scanf.
* puts( cadena ) manda a la salida estándar una secuencia de caracteres almacenadas en la variable cadena o una cadena representada entre “ ” y coloca al final de los elementos de la cadena un salto de línea ‘\n’.

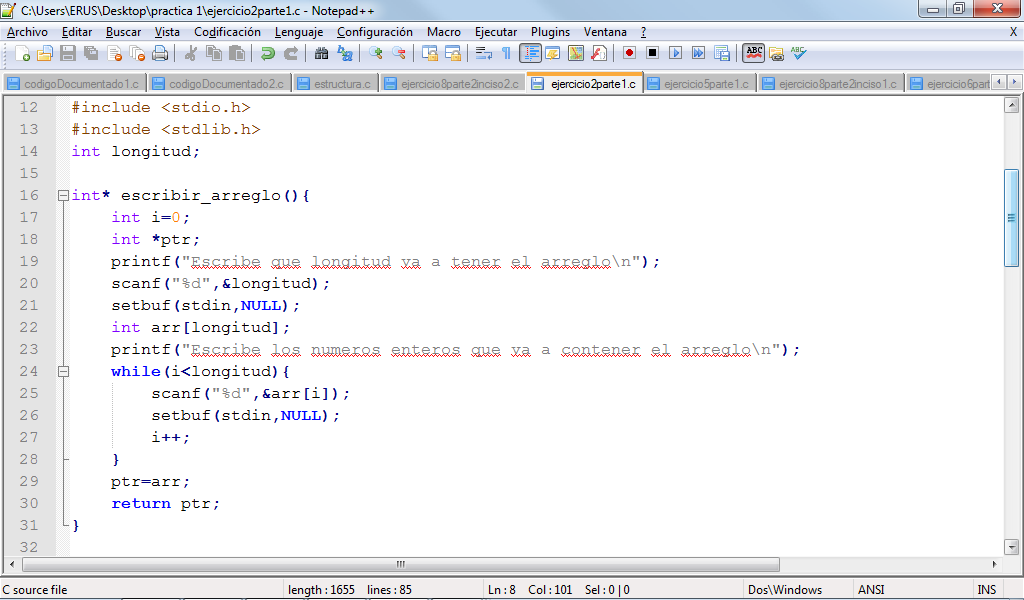
gets y puts son funciones que solo trabajan con cadenas de caracteres, pero no con otros tipos de datos como lo hacen printf y scanf de C.

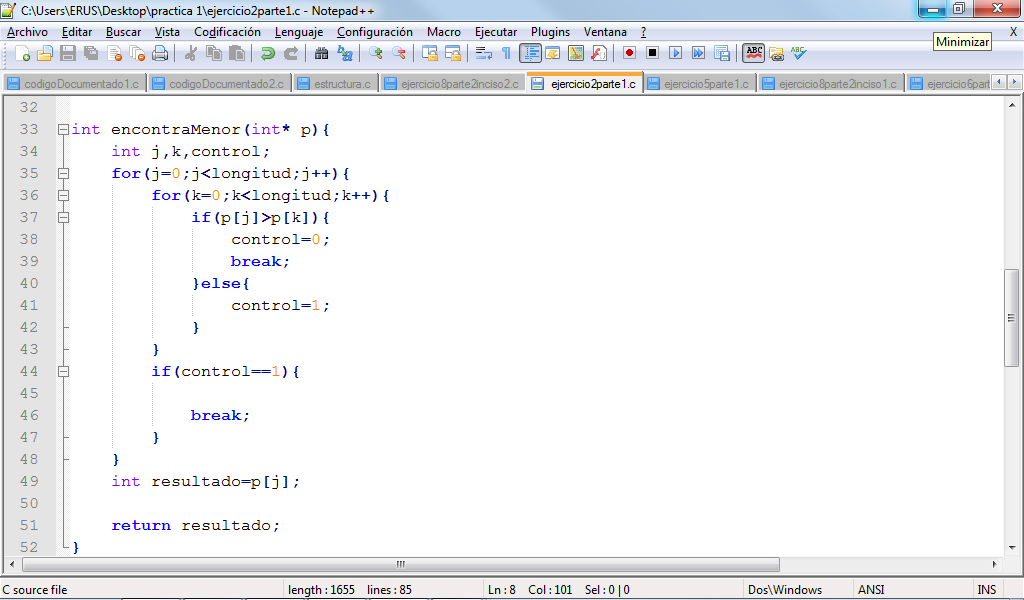
Diseño y Análisis de la práctica

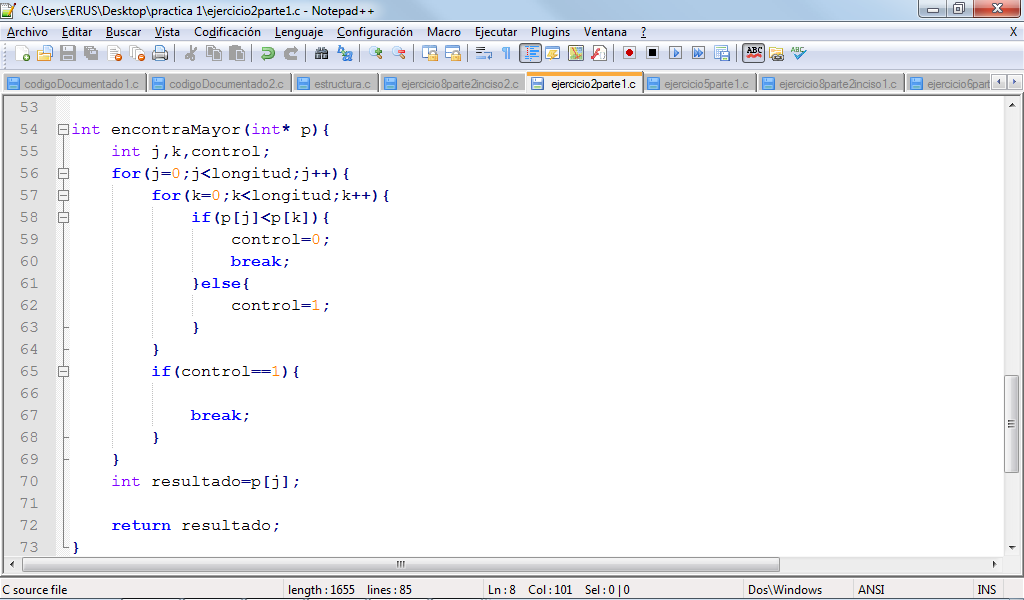
**Primera parte**

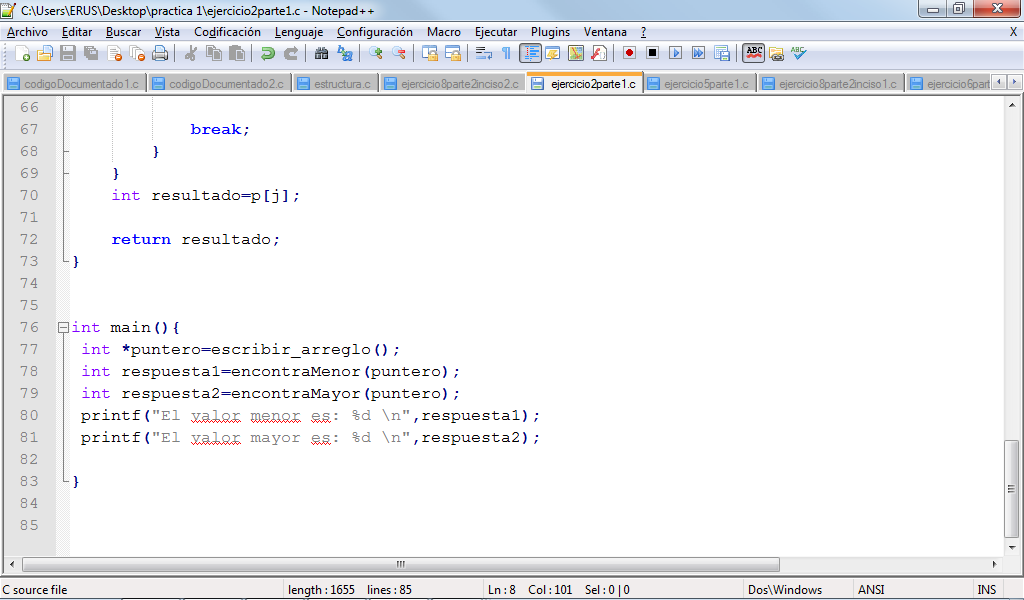
**Ejercicio1.**

**Ejercicio2.**



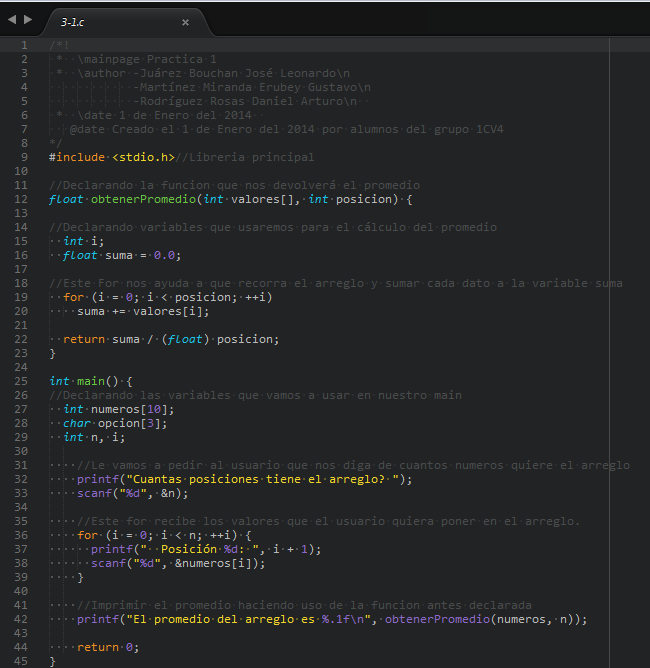






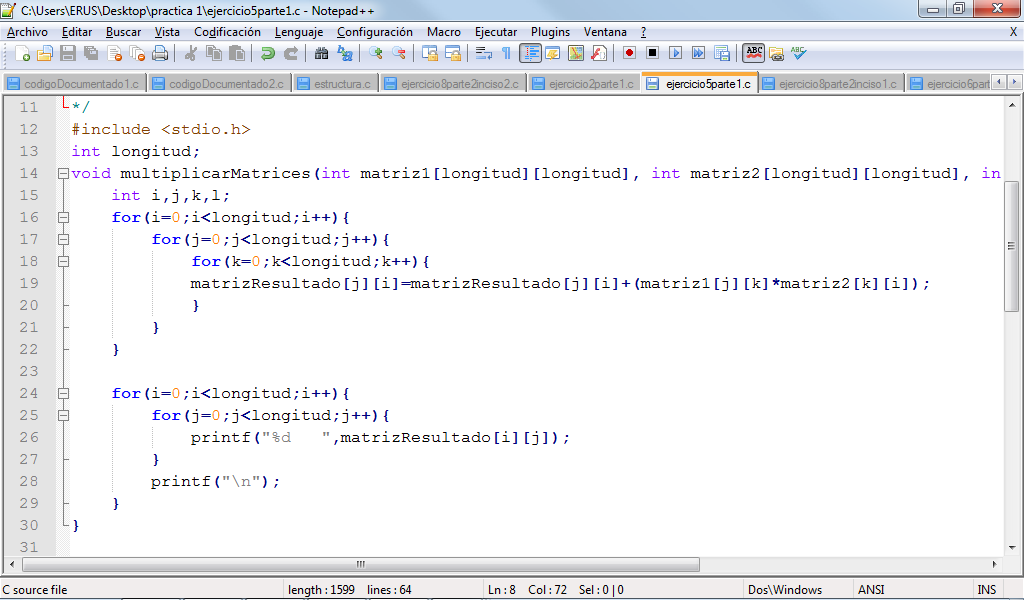
**Ejercicio3.**

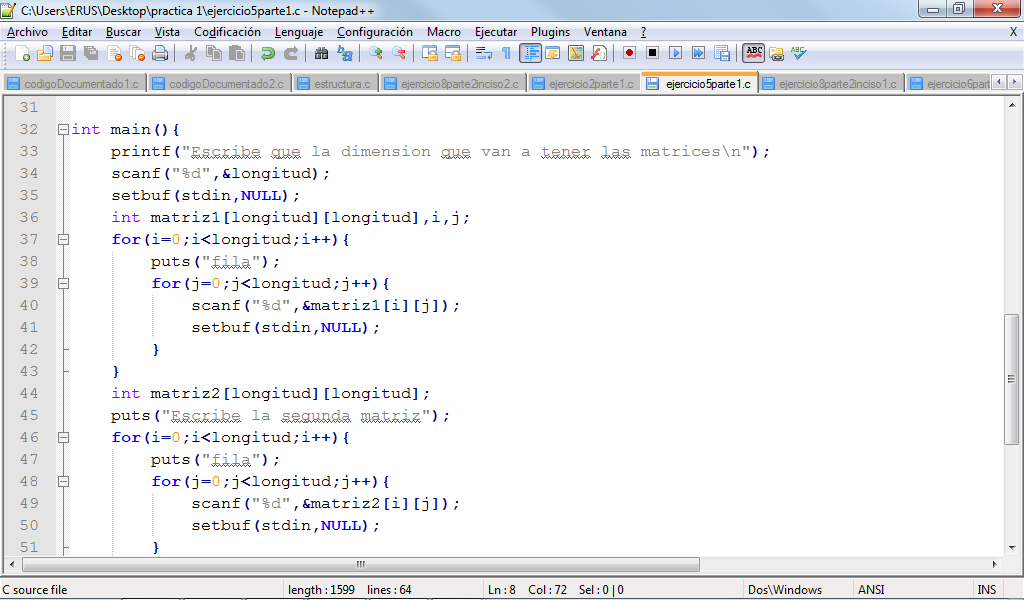
El siguiente código muestra cómo obtener el promedio de los elementos de un arreglo. El promedio de los elementos.



**Ejercicio4.**

**Ejercicio5.**

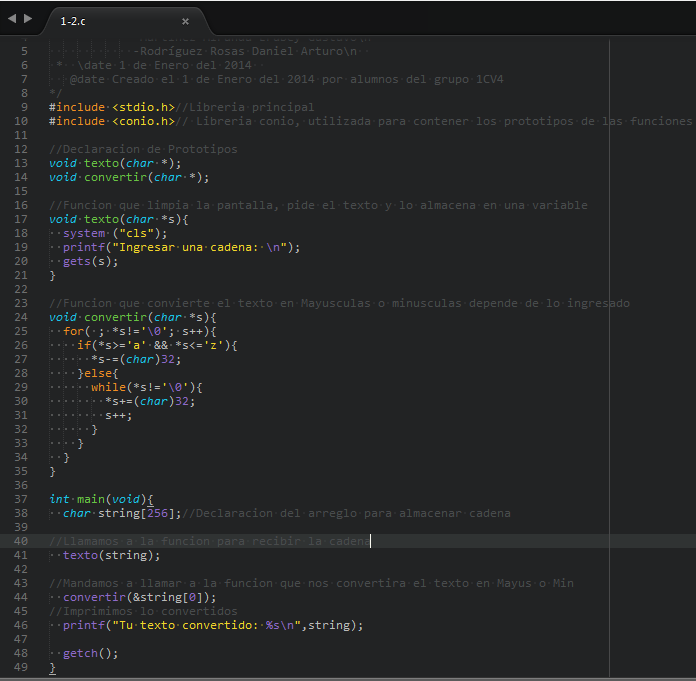




**Segunda parte**

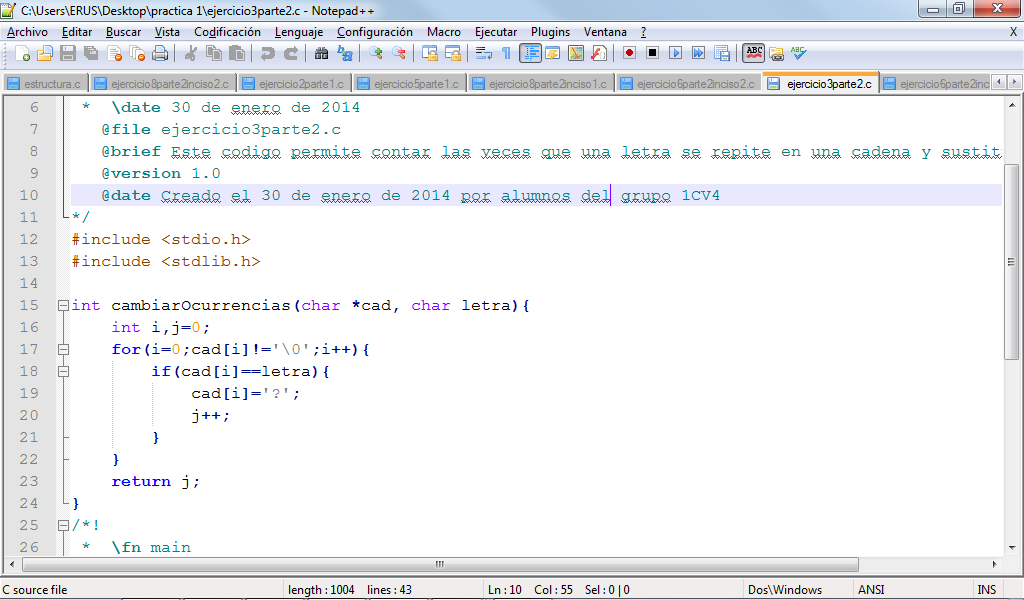
**Ejercicio1.**

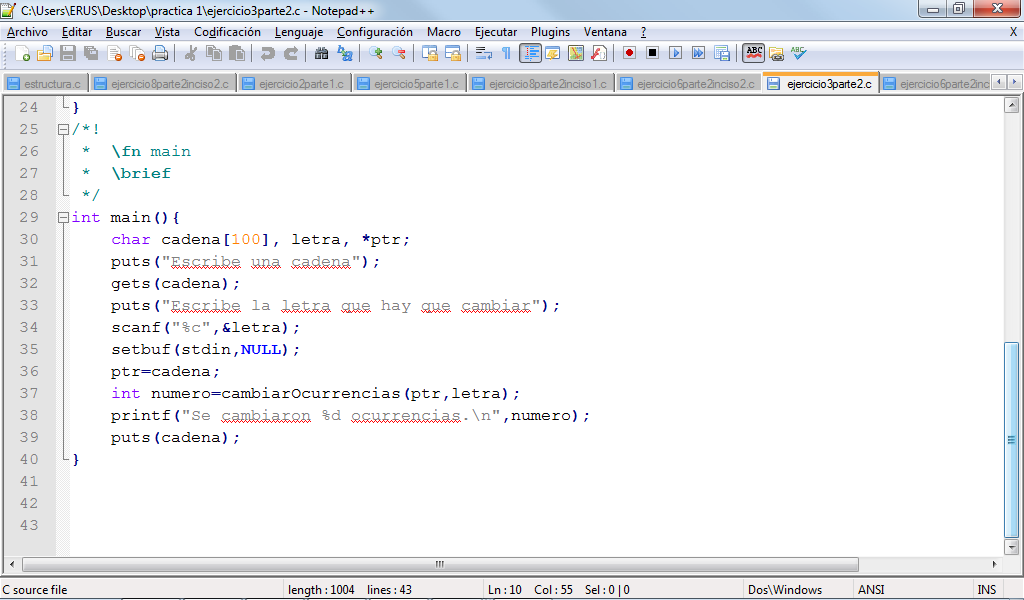
Este código muestra cómo convertir de mayúsculas a minúsculas y viceversa en una cadena.



**Ejercicio2.**

**Ejercicio3.**





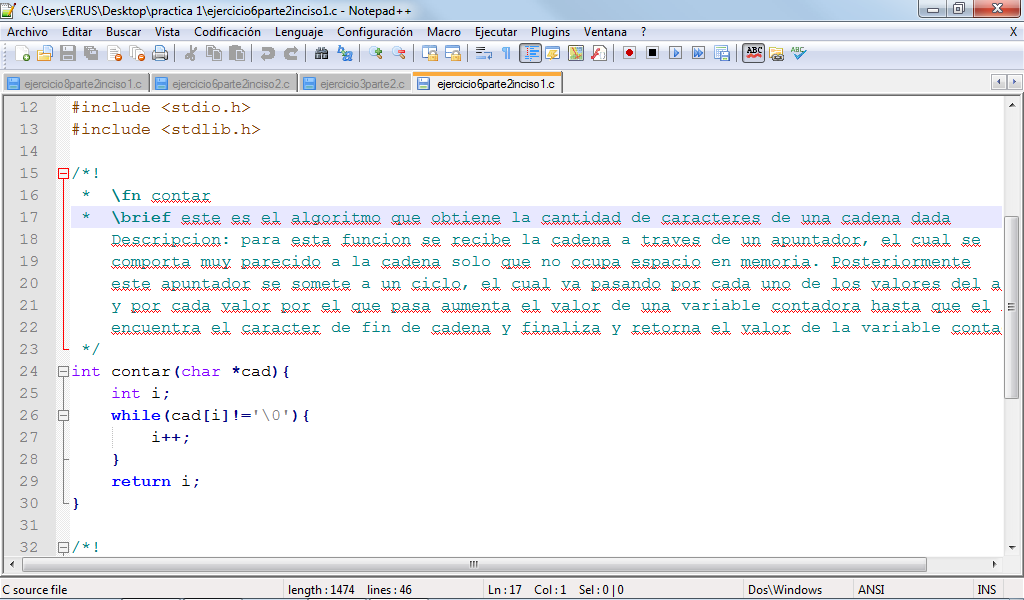
**Ejercicio4.**

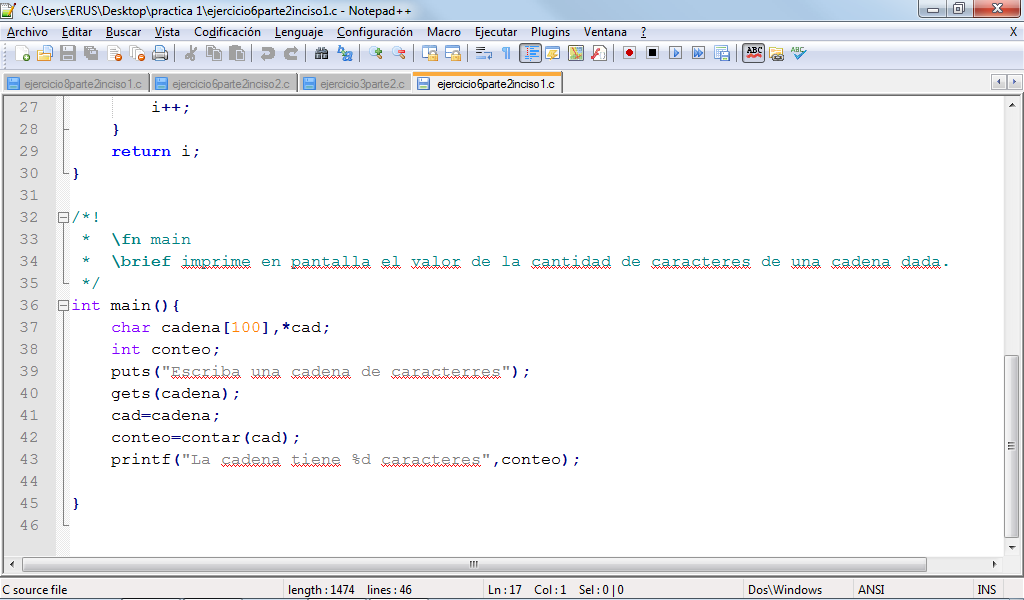
El siguiente código muestra como Validar si una cadena es una palíndroma. La validación de la cadena se deberá realizar en una función que regrese un 1 si la cadena es palíndroma y 0 en caso contrario.

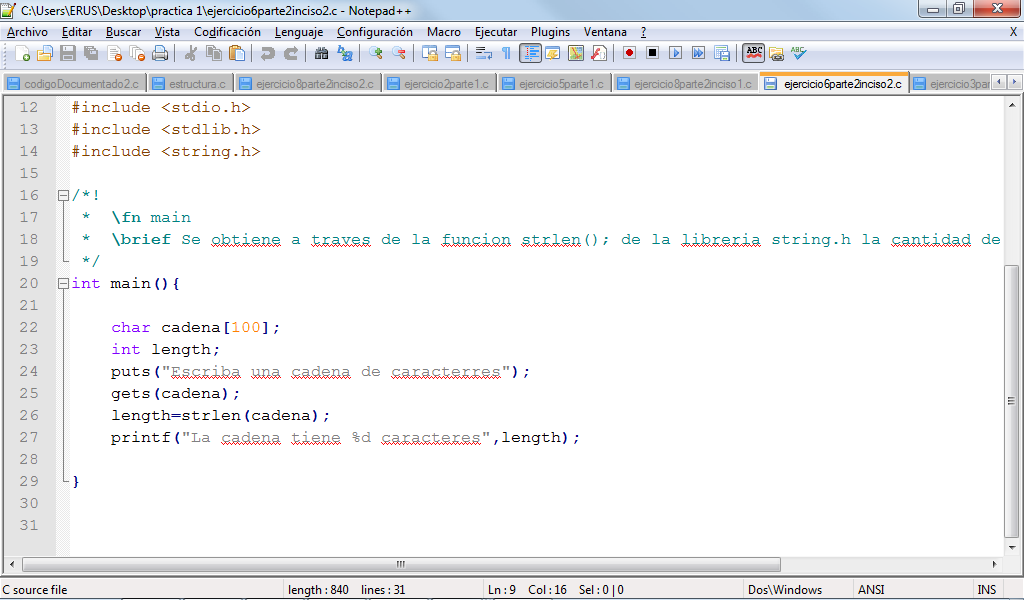


**Ejercicio5.**

**Ejercicio6.**





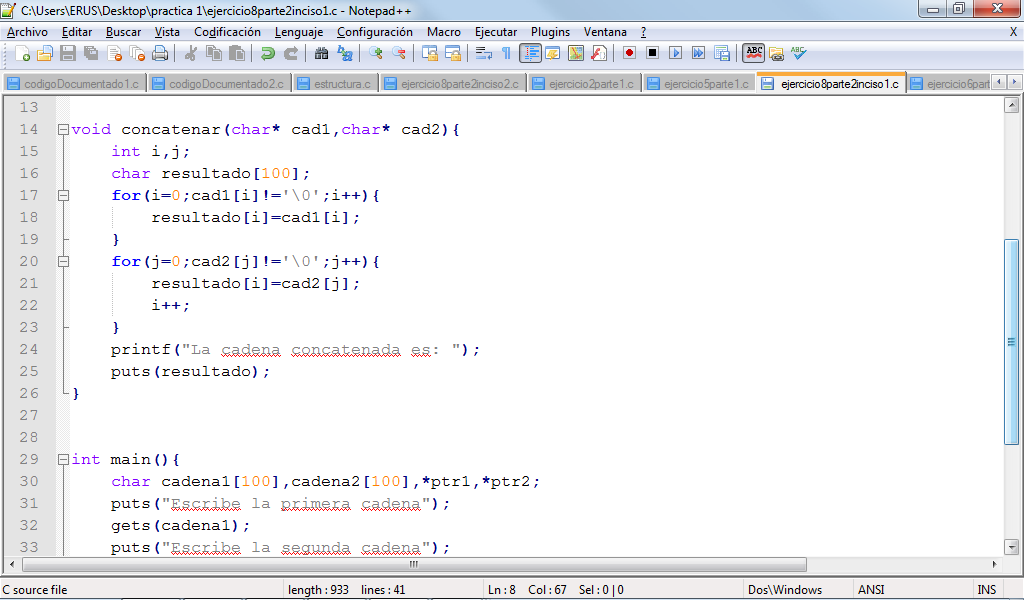
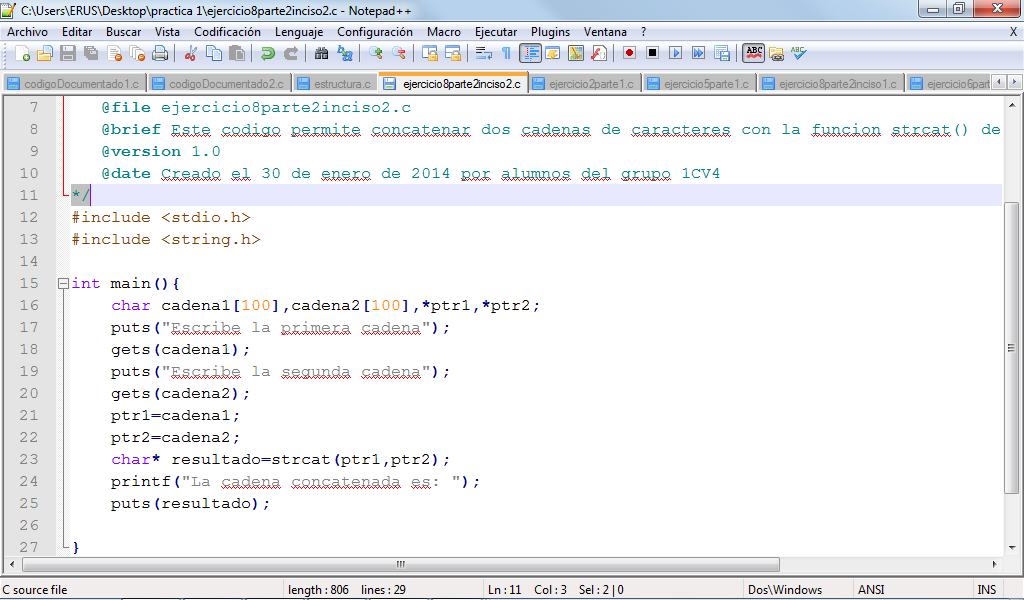


**Ejercicio7.**

1) Utilizando aritmética de apuntadores y 2) Utilizando funciones de la biblioteca copiar una cadena a otra.



**Ejercicio8.**

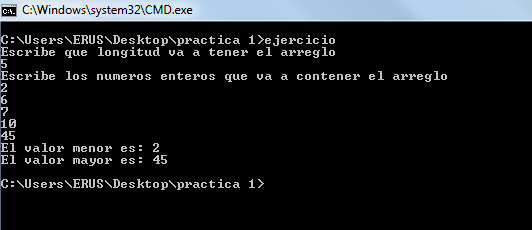


Pruebas y Resultados

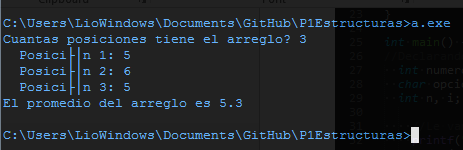
**Primera parte**

**Ejercicio1.**

**Ejercicio2.**

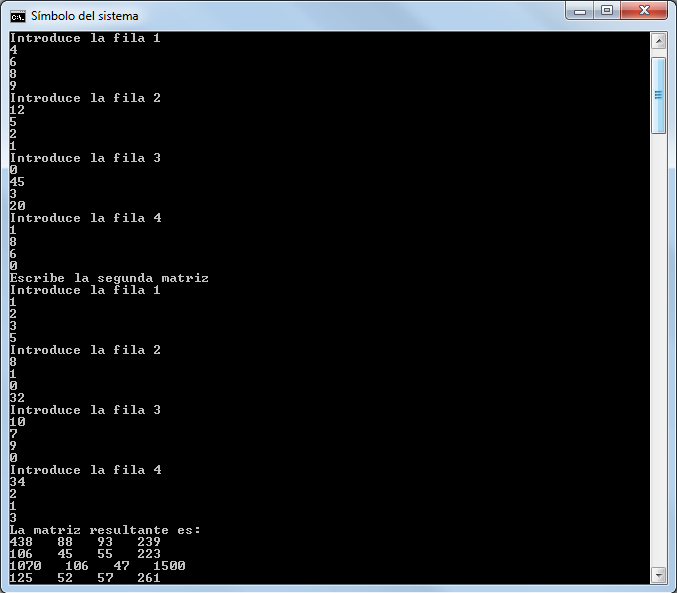
****

**Ejercicio3.**



**Ejercicio4.**

**Ejercicio5.**

****

**Segunda parte**

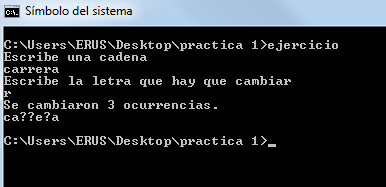
**Ejercicio1.**



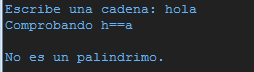


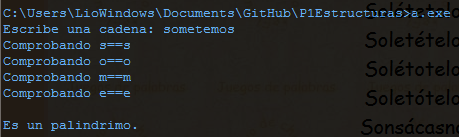
**Ejercicio2.**

**Ejercicio3.**

****

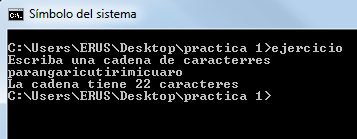
**Ejercicio4.**

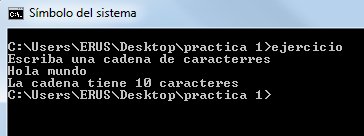




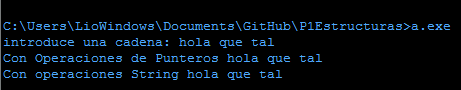
**Ejercicio5.**

**Ejercicio6.**

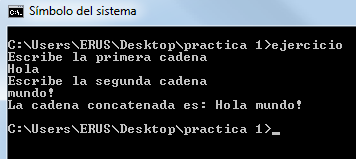
****

****

**Ejercicio7.**



**Ejercicio8.**

****

Conclusiones

**Conclusión de Leonardo:**

Las prácticas me parecieron muy divertidas y para lograr hacerlas tuve que repasar mucho sobre punteros ya que los había olvidado por completo, sin duda creo que es una buenísima práctica por que con esto nos ayuda a agarrar el hilo otra vez, me quedé con ganas de programar más espero que en clase nos de tips y cosas por el estilo ya que estoy muy interesado en seguir aprendiendo y me gusta su manera de enseñanza.

**Conclusión de Erubey:**

Para mi esta práctica fue de mucha utilidad ya que me ayudo a recordar varias cosas que ya no me acordaba. Fue una buena práctica para empezar este semestre, y para mí los ejercicios estuvieron muy interesantes e ilustrativos.