|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Nombre de la práctica** | **APUNTADORES** | | | **No.** | **1** |
| **Asignatura:** | **METODOS NÚMERICOS** | **Carrera:** | **ING. SISTEMAS COMPUTACIONALES** | **Duración de la práctica (Hrs)** | **2** |

**BECERRA HERNÁNDEZ JESSICA LILIAN 342 ISIC**

**I. Competencia(s) específica(s):**

Utiliza algoritmos numéricos que proporcionen el mínimo de error para obtener soluciones aproximadas de modelos matemáticos de aplicación en ingeniería que no puedan resolverse por métodos analíticos utilizando un lenguaje de programación como herramienta.

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

Otro

**III. Material empleado:**

DEV C++

Computadora

**IV. Desarrollo de la práctica:**

**¿Qué es un apuntador?**

Un puntero es un objeto que apunta a otro objeto. Es decir, una variable cuyo valor es la dirección de memoria de otra variable.

En C no se debe indicar numéricamente la dirección de la memoria, sino que se usa una etiqueta que conocemos como variable.

Las direcciones de memoria dependen de la arquitectura del ordenador y de la gestión que el sistema operativo haga de ella.

**¿Cómo se declaran los apuntadores?**

Para declarar un apuntador se especifica el tipo de dato al que apunta, el operador ‘\*’, y el nombre del apuntador.

Un puntero tiene su propia dirección de memoria.

La sintaxis es la siguiente:

<tipo de dato apuntador> \*<identificador del apuntador>

int \* punt;

**¿Cómo se declaran los apuntadores?**

Al igual que el resto de las variables, los apuntadores se enlazan a tipos de datos específicos, de manera que a un apuntador sólo se le puede asignar direcciones de variables del tipo especificado en la declaración.

Tipos de apuntadores

Se puede también declarar apuntadores a estructuras más complejas.

Funciones

Struct

Ficheros

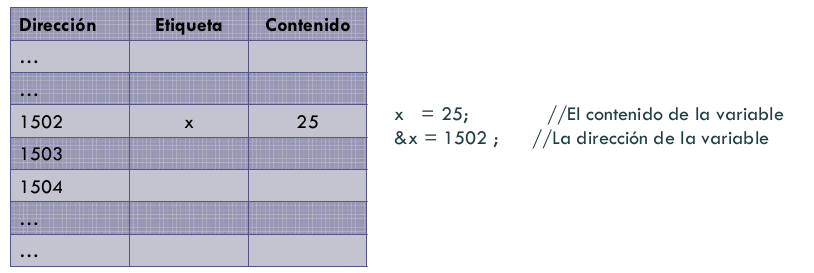
**¿Qué es la referenciación?**

La referenciación es obtener la dirección de una variable.

Se hace a través del operador ‘&’, aplicado a la variable a la cual se desea saber su dirección

&x ; //La dirección de la variable

No hay que confundir una dirección de memoria con el contenido de esa dirección de memoria.



**¿Qué es la desreferenciación?**

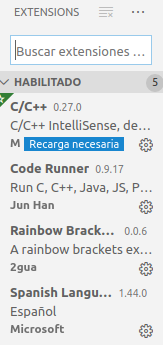
Es la obtención del valor almacenado en el espacio de memoria donde apunta un apuntador.

Se hace a través del operador ‘\*’, aplicado al apuntador que contiene la dirección del valor.

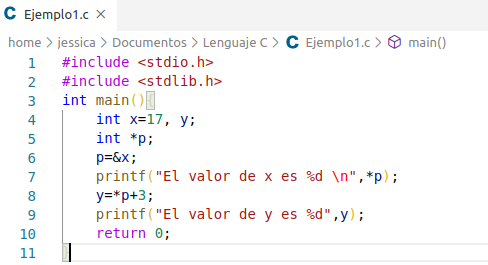
\*p ; //El contenido de p

Como ejemplo abrimos nuestro Visual Studio Code el cual estaremos usando para programar nuestros ejercicios.

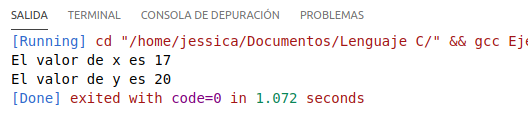
Debemos instalar la extensión C/C++ y Code Runner, Code Runner nos va a permitir ejecutar de manera más sencilla nuestros programas en VSC.

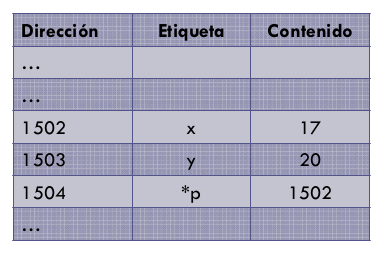


Nuestro ejemplo hace uso de la referenciación:



Damos clic en Run Code

Se muestra por la Salida nuestro programa:

Su funcionamiento en la memoria RAM es de la siguiente manera:

Asignación de apuntadores

A un apuntador se puede asignar direcciones de variables a través del operador de referenciación (“&”) o direcciones almacenadas en otros apuntadores.

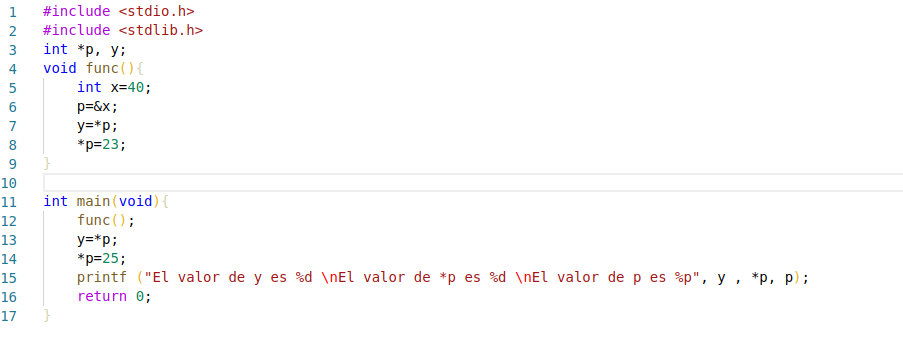
Direcciones inválidas

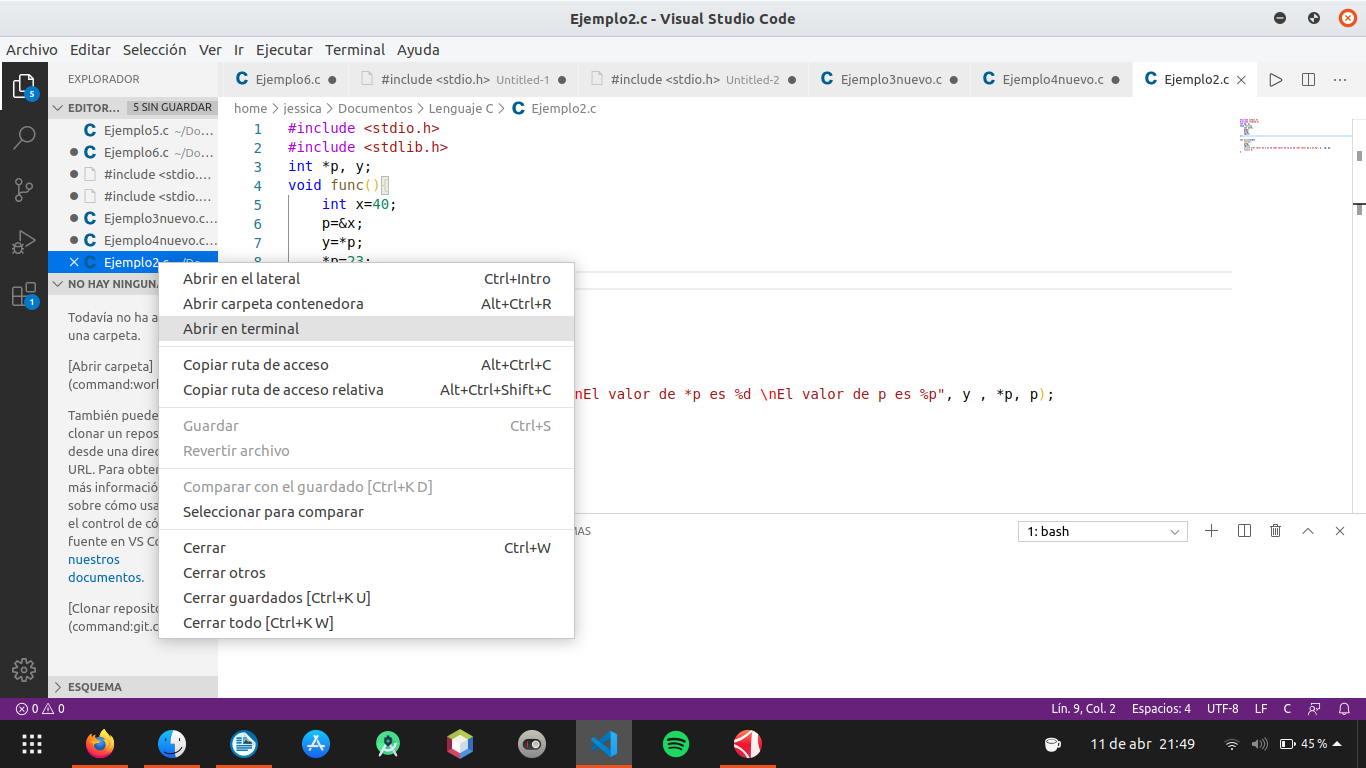
Un apuntador puede contener una dirección inválida por:

Cuando se declara un apuntador, posee un valor cualquiera que no se puede conocer con antelación.

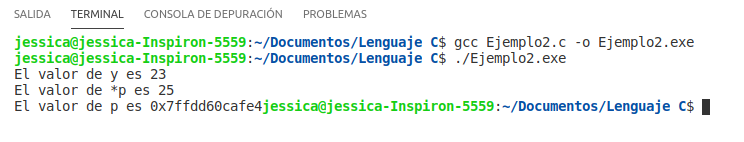
Después de que ha sido inicializado, la dirección que posee puede dejar de ser válida por que la variable asociada termina su ámbito o porque ese espacio de memoria fue reservado dinámicamente.

Ejemplo:



Para poder hacer nuestra compilación y ejecución de nuestro programa vamos a abrir una terminal dentro de Visual Studio Code.

Procedemos a hacer la compilación por medio de gcc el nombre de nuestro archivo guardado previamente en una carpeta, y creamos él .exe que nos va a permitir ejecutarlo.



**La dirección NULL**

Cuando no se desea que el apuntador apunte a algo, se le suele asignar el valor de NULL, en cuyo caso se dice que el apuntador es nulo (no apunta a nada).

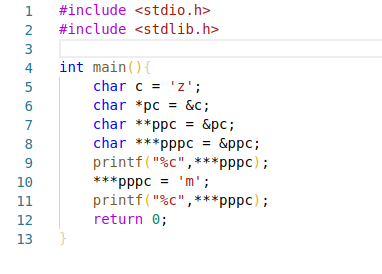
NULL es una macro típicamente definida en archivos de cabecera como stdef.h y stdlib.h.

Se utiliza para proporcionar a un programa un medio de conocer cuándo un apuntador contiene una dirección inválida.

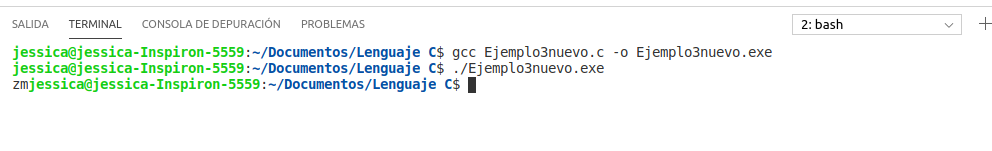
**Apuntadores a apuntadores**

Dado que un apuntador es una variable que apunta a otra, fácilmente se puede deducir que pueden existir apuntadores a apuntadores, y a su vez los segundos pueden apuntar a apuntadores.

Ejemplo:

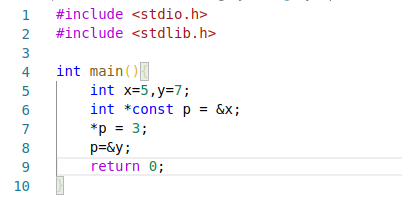


Se nos muestra nuestra salida dos caracteres, que es la “z” y la “m”

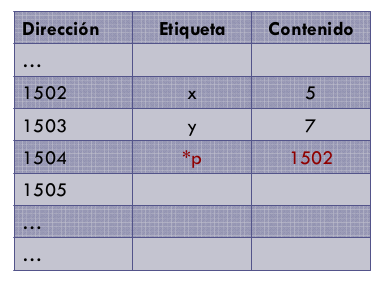


**Apuntadores constantes**

Es posible declarar apuntadores a constantes. De esta manera, no se permite la modificación de la dirección almacenada en el apuntador, pero si se permite la modificación del valor al que apunta.

Ejemplo:

Este es su comportamiento en la memoria RAM



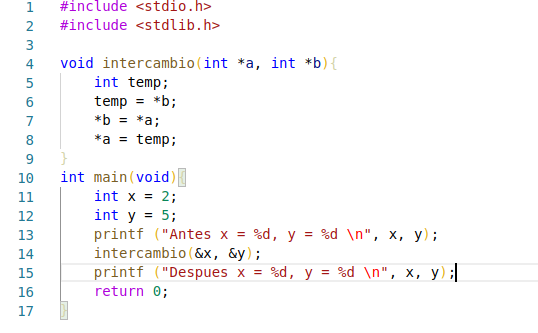
**Paso de parámetros por referencia**

En este tipo de llamadas los argumentos contienen direcciones de variables.

Dentro de la función la dirección se utiliza para acceder al argumento real.

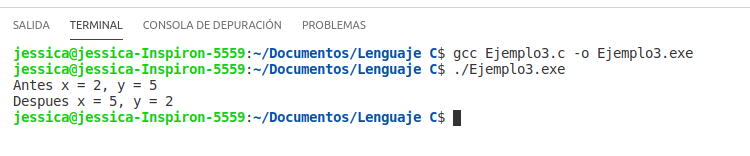
En las llamadas por referencia cualquier cambio en la función tiene efecto sobre la variable cuya dirección se pasó como argumento.

Ejemplo:



Se nos muestra por terminal nuestro resultado, el propósito de este código es cambiar el orden de salida de cada elemento mediante la función intercambio, nos cambia las posiciones de el apuntador a y b.

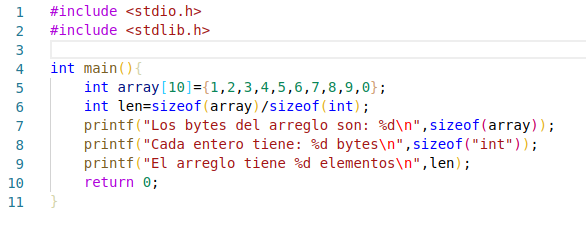
En el lenguaje Java se llama a este tipo de intercambios métodos de ordenamiento.

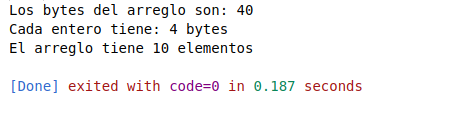


**La función sizeof()**

Devuelve el tamaño en bytes que ocupa un tipo o variable en memoria.

Ejemplo:



Ejecutamos nuestro programa y manda a imprimir los bytes del arreglo, la cantidad de bytes que tiene cada entero y el número de elementos del arreglo.

**Asignación dinámica de memoria**

Los programas pueden crear variables globales o locales.

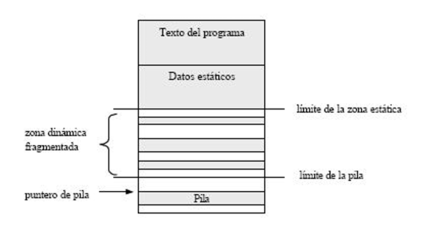
Las variables declaradas globales en sus programas se almacenan en posiciones fijas de memoria (segmento de datos) y todas las funciones pueden utilizar estas variables.

Las variables locales se almacenan en la pila (stack) y existen solo mientras están activas las funciones donde están declaradas.

En ambos casos el espacio de almacenamiento se reserva en el momento de compilación del programa.

**Asignación dinámica de memoria**

Para asignar memoria dinámicamente se utilizan las funciones malloc() y free(), definidas típicamente en el archivo stdlib.h.



**free ()**

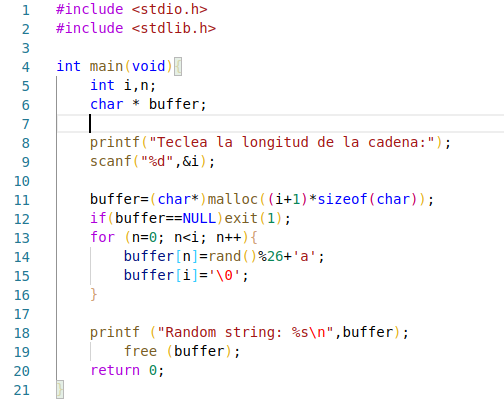
La función free () permite liberar la memoria reservada a través de un apuntador.

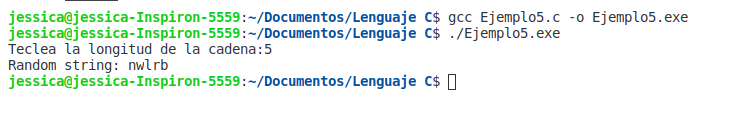
**malloc ()**

La función malloc () reserva memoria y retorna su dirección, o retorna NULL en caso de no haber conseguido suficiente memoria.

malloc () reserva memoria sin importar el tipo de datos que almacenará en ella.

Ejemplo:

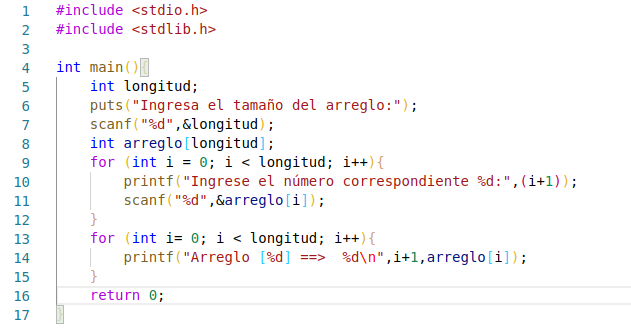


Se nos muestra nuestra salida desde terminal, el ejercicio nos regresa una cadena de la longitud que nosotros queramos.

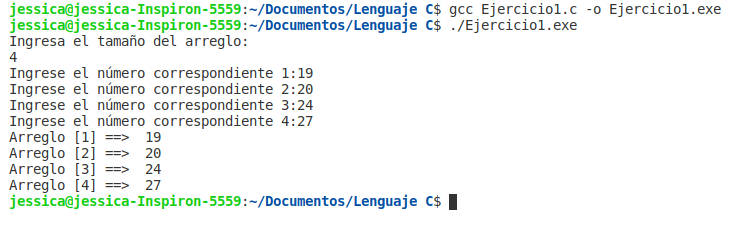
Crea un arreglo entero de tamaño x, en donde x es ingresado por teclado.

Llena todos los elementos del arreglo con datos ingresados por el usuario.

Muestra los valores



Por terminal se muestra la resolución del ejercicio.



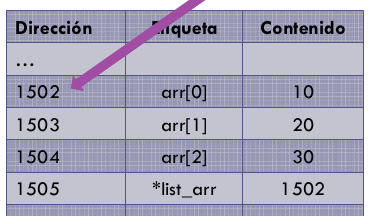
**Apuntadores a arreglos**

El nombre de un arreglo es simplemente un apuntador constante al inicio del arreglo

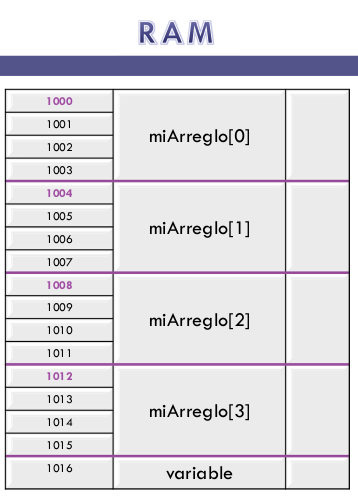
Ejemplo:

int lista\_arr[3]={10,20,30};

int \*lista\_ptr;

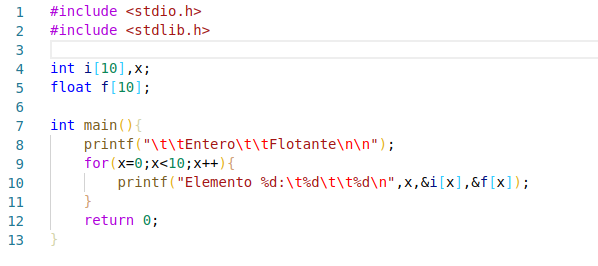
lista\_ptr = lista\_arr;

Se crea el apuntador lista\_ptr para poder ir modificando la dirección a donde apunta.

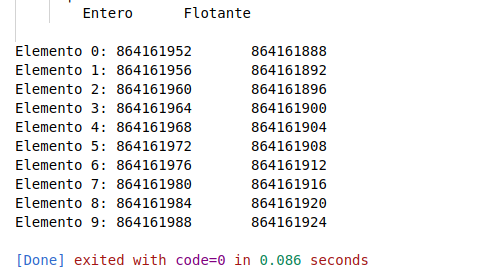


Direcciones del arreglo

Ejemplo:



Se nos muestra en nuestra consola de la siguiente manera:



**ARITMÉTICA DE OPERADORES**

Incremento de operadores

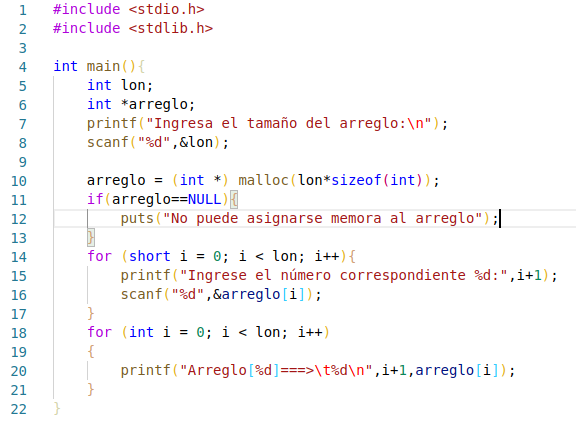
Cuando se incrementa un apuntador se está incrementando su valor.

Si incrementamos en 1 el valor del apuntador, C sabe el tipo de dato al que apunta e incrementa la dirección guardada en el apuntador en el tamaño del tipo de dato.

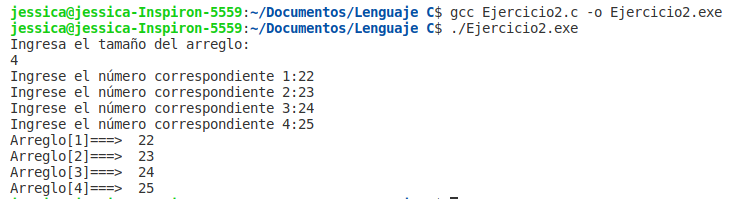
Ejercicio

Crea un arreglo entero de tamaño x, en donde x es ingresado por teclado.

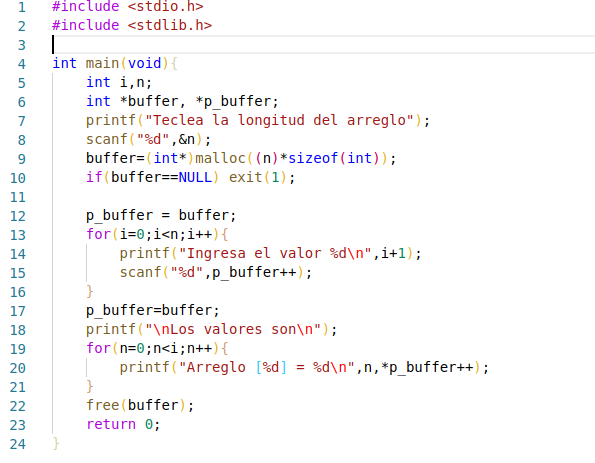
Llena todos los elementos del arreglo con datos ingresados por el usuario usando apuntadores.



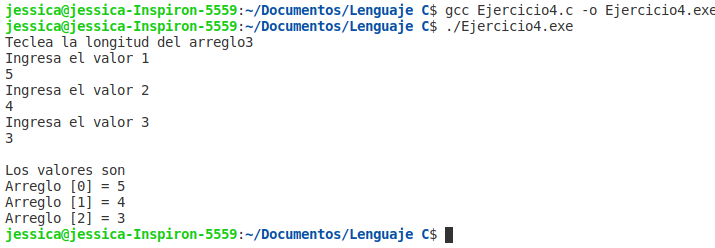
Se nos muestra por la terminal cada elemento del arreglo ingresado por el teclado.



Ejemplo:



Se nos muestra nuestro programa:



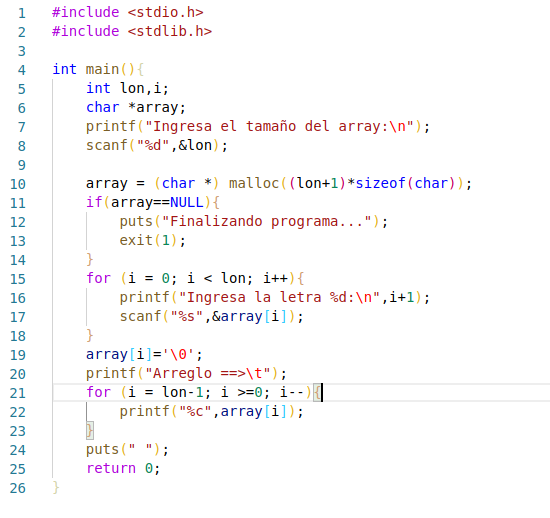
Ejercicio

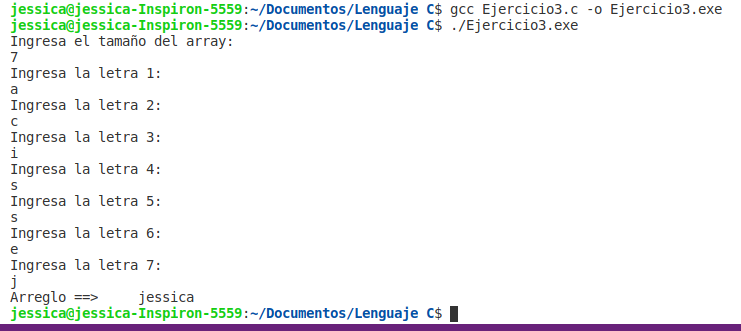
Crea un arreglo de tipo char de tamaño x, en donde x es ingresado por teclado.

Llena elemento por elemento del arreglo con letras ingresados por el usuario.

Muestra el arreglo impreso en forma inversa.

Todo debe ser manejado con apuntadores.





Se muestra nuestro programa por terminal y vemos que queda de manera invertida los caracteres ingresados.

**V. Conclusiones:**

Aprendí el uso de los apuntadores en C, la referenciación y la desreferenciación, las direcciones de memoria, algunos métodos como free (), que nos sirve para liberar la memoria reservada a través de un apuntador o malloc () que reserva memoria y retorna su dirección, o retorna NULL en caso de no haber conseguido suficiente memoria.

También aprendí cómo se comporta un arreglo en apuntadores.