

**UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO**

(Ejercicio Apuntadores)

Calzada Martínez Jonathan Omar

Profesor

Gabriel Castillo Hernández

MÉXICO, D. F. (Septiembre) 2012

M en C Gabriel Castillo Hernández Ejercicios sobre apuntadores.

Para las siguientes preguntas considere que los números enteros utilizan 4 celdas de memoria y que el siguiente programa imprimió un 45621

int main(){

int a[] = {1,6,3,5};

int \*p; p = a;

printf("%d",&a); return 0;

}

Para cada pregunta suponga que se parte del programa arriba mostrado (Es decir los cambios introducidos por cada pregunta no son acumulados para responder la siguiente preguta)

1. ¿Qué imprimirá si ejecutamos la línea: printf("%d",a+3); antes del return?

R=45633

1. ¿Qué imprimirá si ejecutamos la línea: printf("%d",\*a+3); antes del return?

R= 4

1. ¿Qué imprimirá si ejecutamos la línea: printf("%d",\*(a+3)); antes del return?

R=5

1. ¿Qué imprimirá si ejecutamos la línea: printf("%d",p+3); antes del return?

R=45633

1. ¿Qué imprimirá si ejecutamos la línea: printf("%d",\*p+3)); antes del return?

R= 4

Sean las siguientes declaraciones, contenido de memoria y tabla de símbolos;

int a;

int \*p, \*q; int v[3]; int b;

|  |  |  |
| --- | --- | --- |
| símbolo | tipo | dirección |
| a | int | 102 |
| p | int \* | 101 |
| q | int \* | 110 |
| v | int [] | 100 |
| b | int | 106 |

considere que los enteros y las direcciones ocupan una celda para almacenarse en memoria es decir sizeof(int) es igual a 1

V[]100

\*P101

|  |
| --- |
| 345 |
| 102D=103 |
| A=5 |
| 88 |
| 25 |
| 34 |
| B=10 |
| 92 |
| 21 |
| 31 |
| 43 |
| 103 |

A102

103

104

105

B106

107

108

109

\*Q110

111

Con base en el esquema anterior, responda las siguientes preguntas. Cada pregunta siempre parte del estado de memoria y variables que se ha establecido originalmente (las modificaciones no se acumulan con cada pregunta).

1. Para el siguiente segmento de código se imprime el valor: R= 88

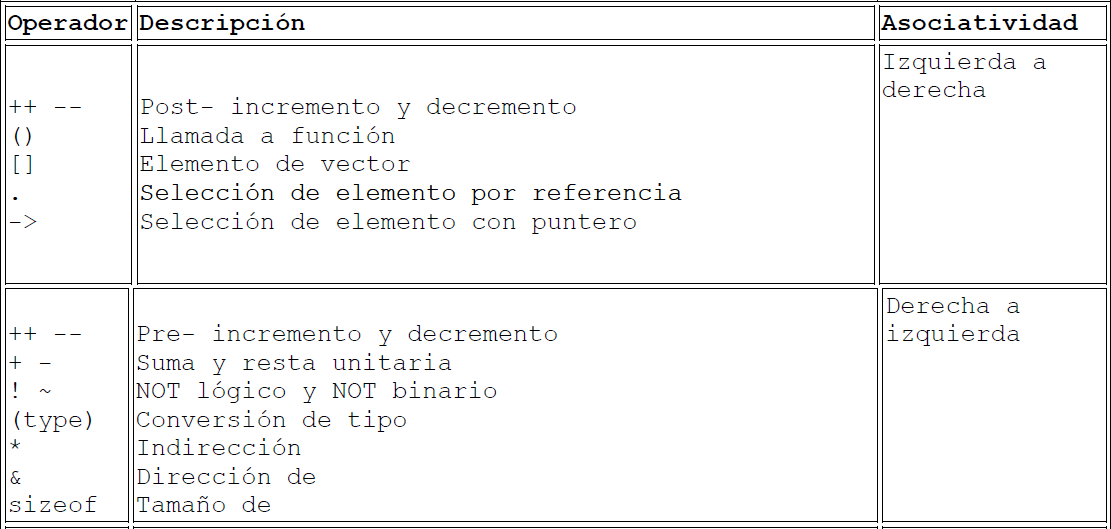
a =5;

b= 10;

p =&a; printf("%d",\*(p+1));

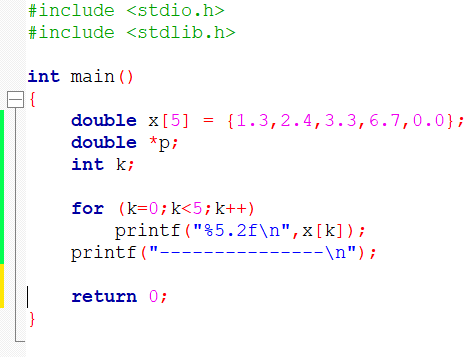
|  |
| --- |
| 7. Para el siguiente segmento de código se imprime el valor: 5 |
| a =5;  b= 10;  p =&a; printf("%d",\*p+1); |
| 8. Para el siguiente segmento de código se imprime el valor: 5  a =5;  b= 10;  p =&a;  \*(p+3)=50;  printf("%d",v[3]); |
| 9. En este segmento de código se imprime el valor: 10 |
| a =5;  b= 10;  v[3] = 0  p =&a; printf("%d",b); |
| 10. En este segmento de código se imprime el valor: 102 D |
| printf("%d",\*v+1); |
| 11. En este segmento de código se imprime el valor: BASURA |
| printf("%d",v[1]); |

|  |
| --- |
| 12. En este segmento de código se imprime el valor: 102 |
| printf("%d",v+1);  PRECEDENCIA DE LOS OPERADORES: En ausencia de paréntesis, los operadores del primer renglón se ejecutan siempre antes que los operadores del segundo renglón. |



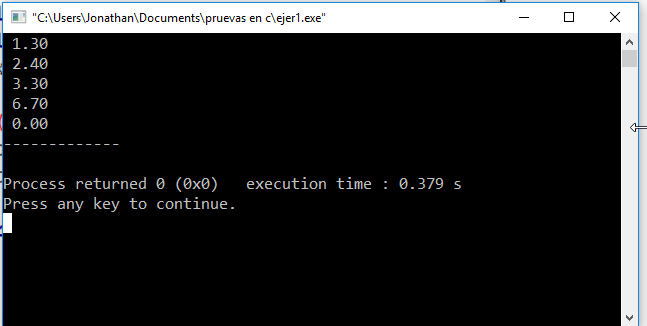
Para cada uno de los siguientes programas, escríbalo en CodeBlocks, ejecútelo, capture su salida, haga un esquema de cómo se vería la memoria y qué cambios irían sufriendo las variables y justifique su respuesta.

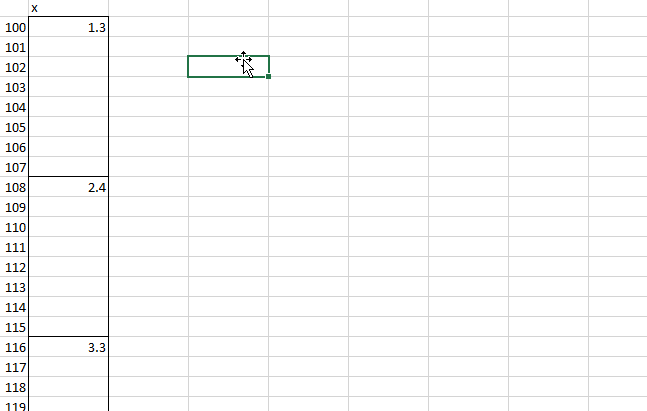
1. Programa:



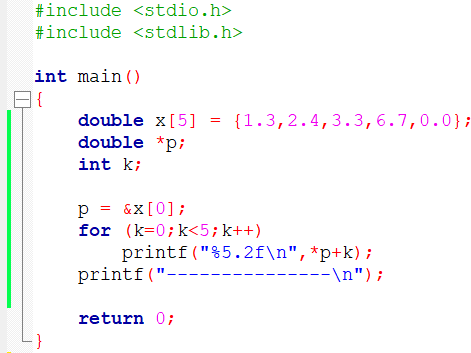
/\* Se inicializo un Array de 5 elementos,

Y por medio de un ciclo for se mostro cada uno de los elementos\*/



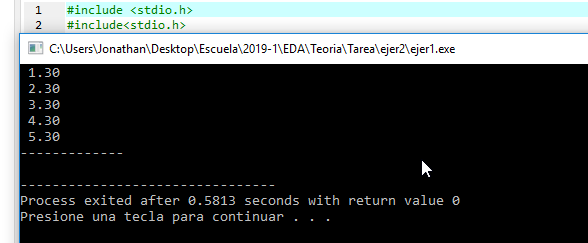


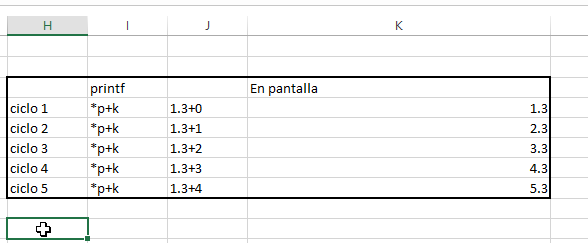
1. Programa:

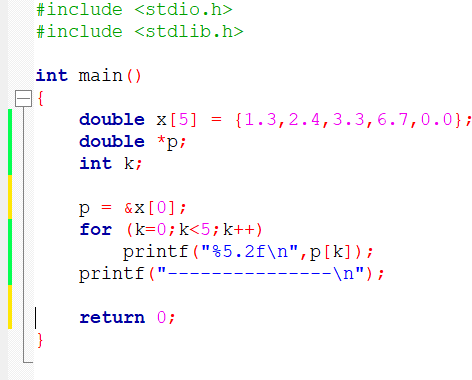


/\*Al valor dentro de \*p que es 1.3 se le suma "k" en cada ciclo, entonces en el cliclo uno

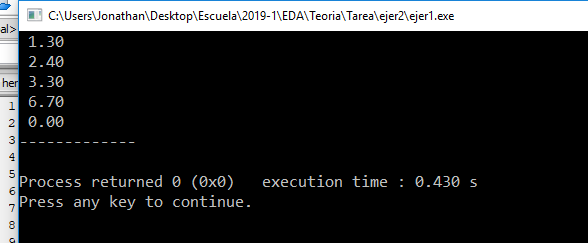
tendra 1.3, en el dos 2.3, en el 3, 3.3 y así sucesivamente \*/

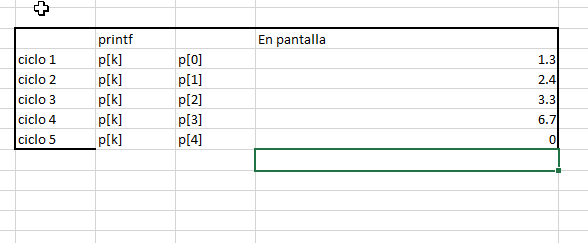




1. Programa:

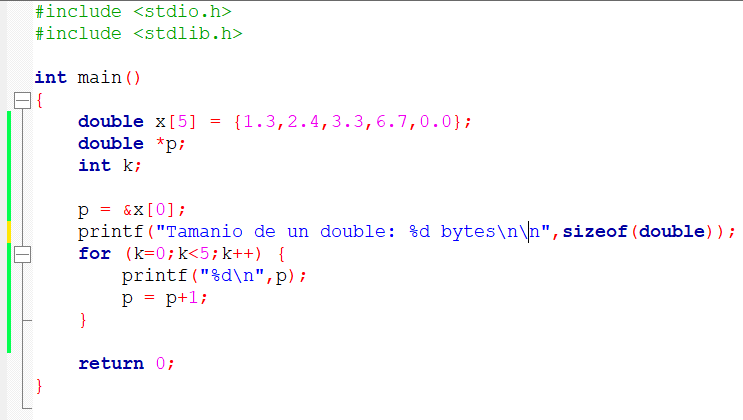
/\*SE LE ASIGNO LA DIRECCIÓN DEL PRIMER VALOR DEL ARREGLO A LA VARIABLE \*P Y CON ELLA SE OPERO EL ARREGLO EN EL CICLO FOR\*/

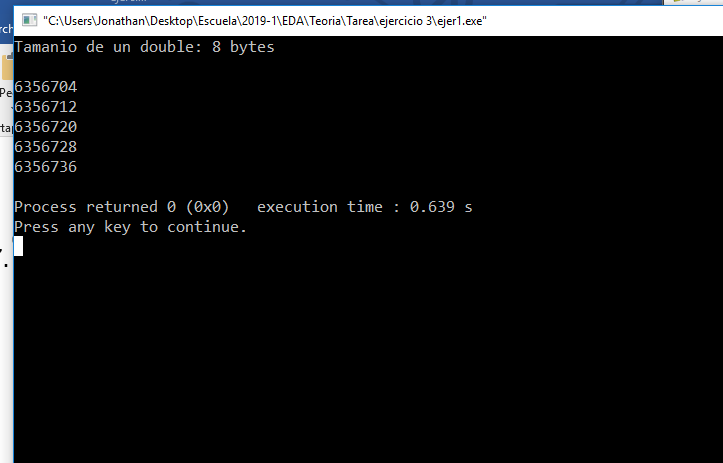


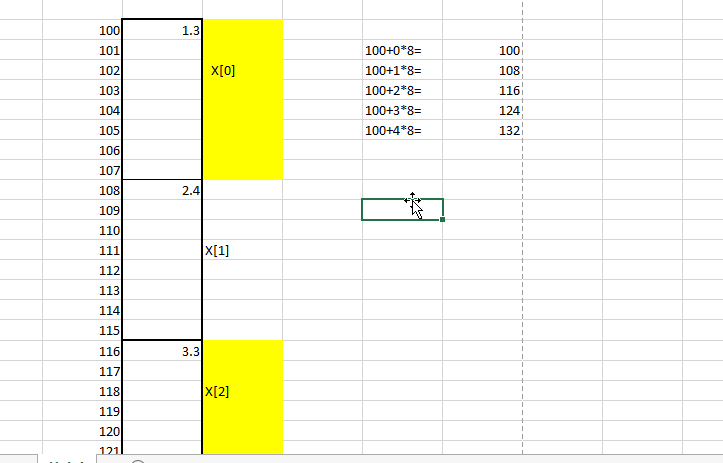


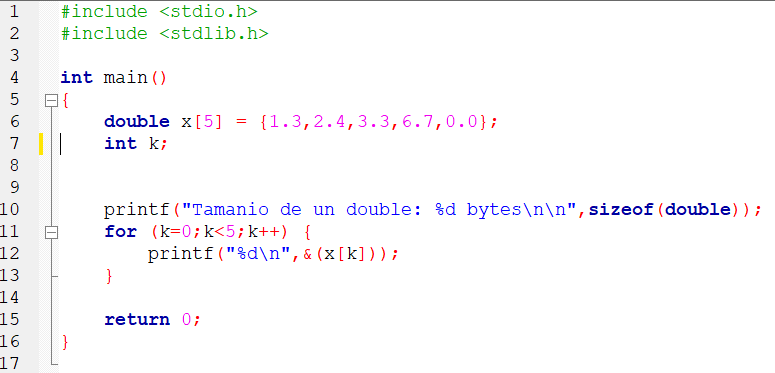
1. Programa:

/\*Muestra en pantalla las direcciones de memoria de los nuemeros inicializados \*/

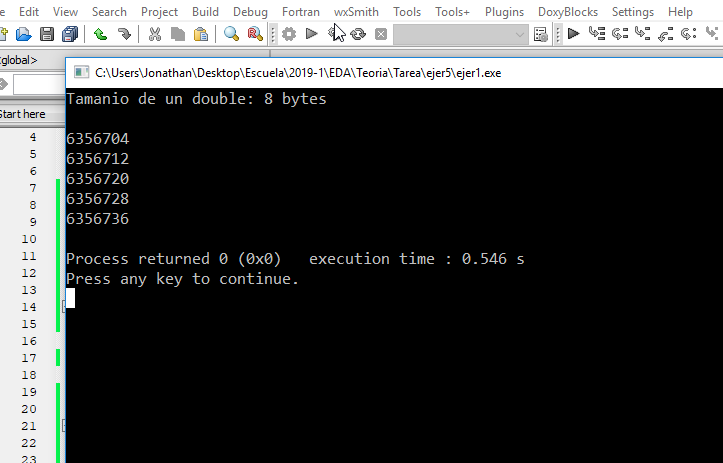




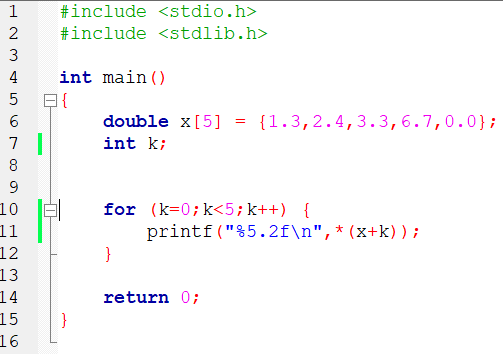


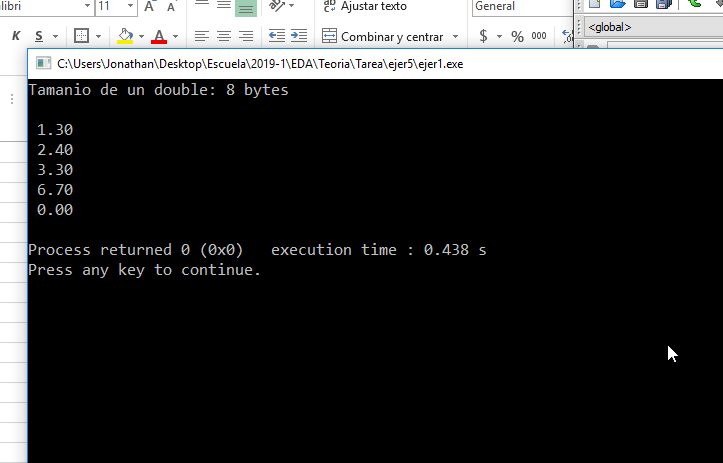
1. Programa:

/\*Muestra las direcciones de memoria en donde se encuentran los datos inicializados de con el indice del for y el ampersand\*/

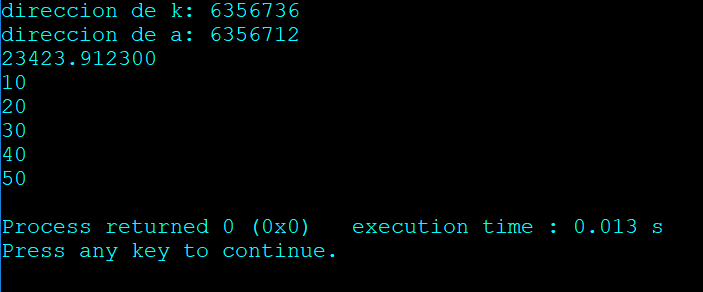
5

Programa:

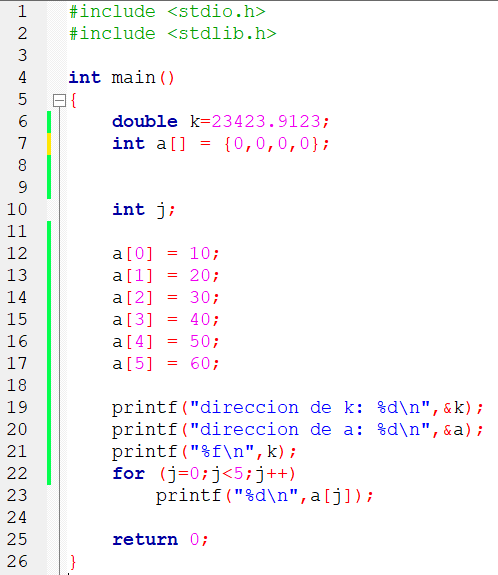




/\*Nos muestra el los valorez inicializados en el arreglo accediendo a la dirección de memoria\*/

1. En el siguiente programa el usuario espera que se tenga la siguiente salida:

Sin embargo, eso no sucede, indique qué salida se obtiene. Para explicarlo, haga un diagrama de memoria donde se muestren las variables y explique a que se debe esto.



1. A partir de estos ejercicios escriba un ensayo donde exponga el concepto de apuntadores, la aritmética de apuntadores y la relación que existe entre apuntadores y arreglos.

En el famoso libro de Kernighan y Ritchie “El lenguaje de programación C”, se define un puntero como «una variable que contiene la dirección de una variable». Tras esta definición clara y precisa, podemos remarcar que un puntero es un tipo especial de variable que almacena el valor de una dirección de memoria.

Los punteros nos permiten manipular la memoria del ordenador, y con ello manipular la con eficiencia las variables que estamos utilizando.

Un apuntador apunta a otra variable (valla la redundancia), utilizando su dirección de memoria.   
  
Los apuntadores pueden declararse para cada tipo de datos e incluso para estructuras complejas como funciones, structs, archivos, e incluso punteros vacíos (void) y punteros nulos.

Un apuntador recién declarado apunta a una dirección indefinida.

A un apuntador se le asigna la dirección de una variable

Int \*p; para declarar   
&p para ver la dirección de memoria de esa variable tipo puntero  
p vara ver la dirección que almacena   
\*p para ir a la dirección que esta dentro de la variable tipo puntero

Se relaciona bastante con los arreglos ya que los índices de los arreglos son punteros ya que podemos alterar, trabajar o modificar el dato teniendo su dirección y en ese caso teniendo el índice de esa posición del arreglo.