C-Sim

Introducción

C-Sim es un simulador de gestión de memoria de C. Su objetivo es ser un método didáctico de mostrar cómo se almacenan las variables y el funcionamiento de los punteros.

Con ella es posible interactuar y observar el efecto del empleo de los punteros, de realizar cambios en variables o en los mismos punteros, y finalmente, del efecto de los operadores & y *, de la forma más educativa posible.

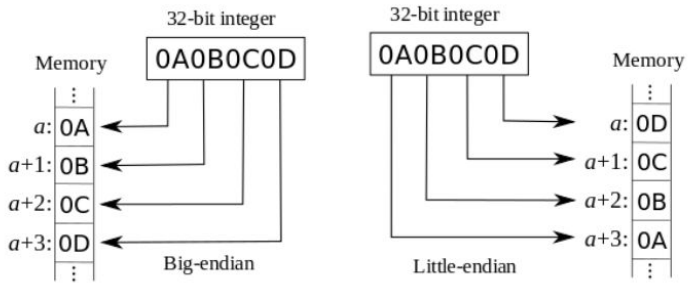
Conceptos previos necesarios.

Para sacarle el mayor provecho a este simulador es necesario conocer una serie de sencillos conceptos.

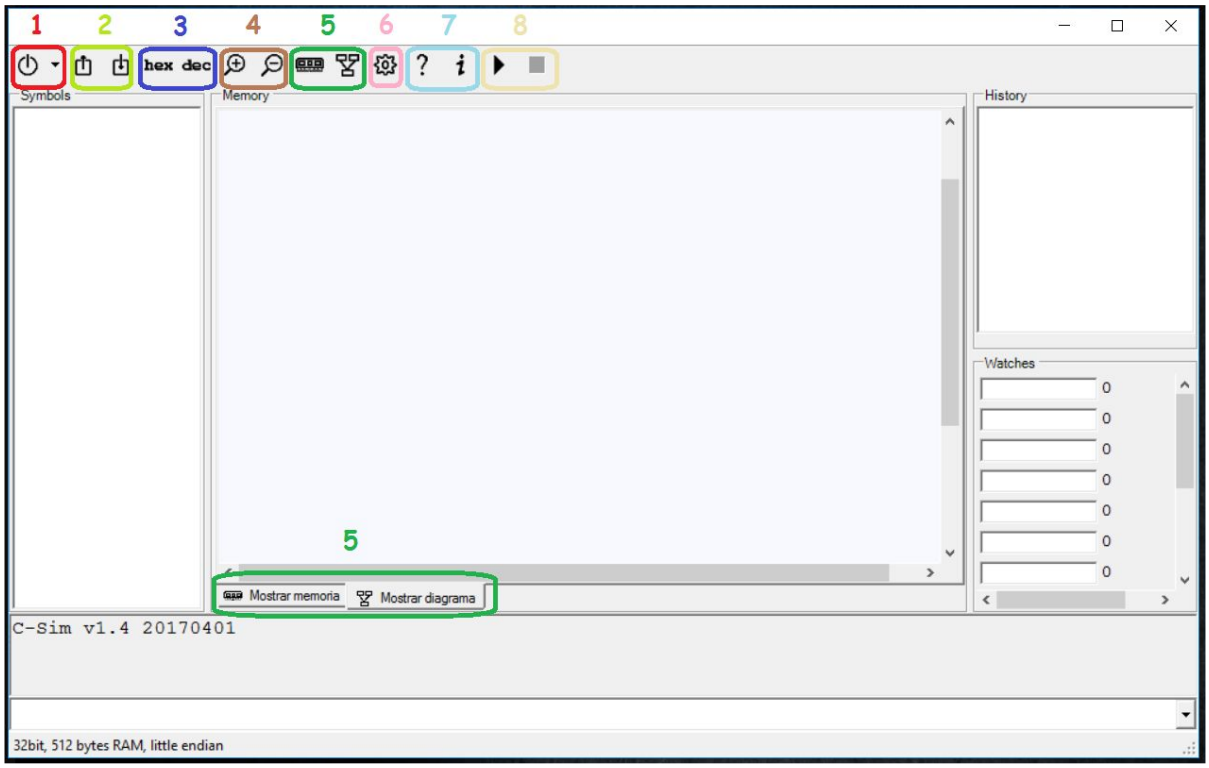
Puntero: un puntero es una variable que "apunta a" una dirección de memoria.
Un puntero puede apuntar a la dirección de memoria de un objeto de cualquier tipo desde una variable, a una función o una estructura.
Los punteros a datos mejoran significativamente el rendimiento de las operaciones repetitivas tales como cadenas de desplazamiento, tablas de búsqueda y estructuras árbol.

Palabra o word: Una palabras una cadena finita de bits que son manejados como un conjunto por la cpu. El tamaño o longitud de una palabra hace referencia al número de bits contenidos en ella.

Big endian - Little endian : Son los distintos formatos en los que se almacenan los datos de más de un byte en un ordenador.
En el sistema *Big-endian* consiste en representar los bytes en el orden "natural" ,así el valor hexadecimal 0x4A3B2C1D se codificaría en memoria en la secuencia {4A, 3B, 2C, 1D}.
En el sistema *little-endian* el mismo valor se codificaría como {1D, 2C, 3B, 4A}, de manera que de este modo se hace más intuitivo el acceso a datos, porque se efectúa fácilmente de manera incremental de menos relevante a más relevante.



Barra de herramientas



1. Botón de Reset: Este botón permite reiniciar la aplicación a su estado inicial. (Esta acción no afecta a los parámetros de "opciones").

2. Botones de Guardado/Carga: Estos botones permiten guardar y cargar todos los comandos introducidos en la aplicación.

-La opción Guardado generará un archivo con extensión .csim.

-La opción Carga borrara todos los comandos que hayas introducido hasta el momento, por ello, si son de relevancia, asegurate de guardarlos antes de cargar algún fichero.

3. Opciones de formato (hexadecimal - decimal): Estas opciones modifican la codificación los de los datos que se muestran.
(OJO: Esto no afecta a los comandos que introduces, solo al formato de los datos que se muestran, por ello para introducir un número en hexadecimal debe estar precedido de "0x". Por ejemplo el número "18" en decimal se expresara como "0x12" en hexadecimal.)

4. Zoom: Permite aumentar o reducir el tamaño de la fuente del texto.

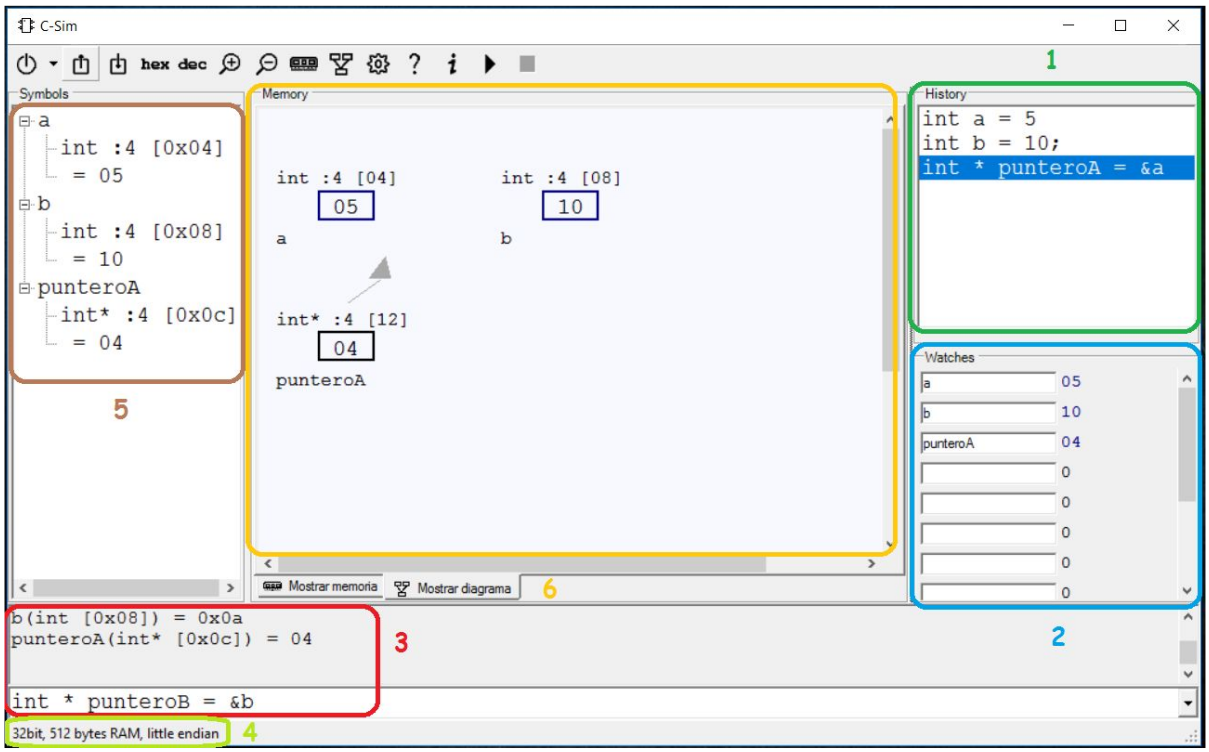
5. Mostrar Diagrama/Memoria: Muestra las ventanas principales de diagrama y memoria (Ver sección “Ventanas principales”).

6. Preferencias/Opciones: Muestra la ventana de “opciones” (Ver sección “Ventana de opciones”).

7. Opciones de información y ayuda: Muestra información de la aplicación y su desarrollador.

8. Opciones Ejecutar y Parar: Ejecuta los comandos introducidos uno a uno, y la opción pausa permite detener su ejecución en cualquier momento.(Más información en la sección “Ventanas principales/Ventana de Diagrama”)

Ventana de Diagrama



1. Historia: Muestra los comandos introducidos en la aplicación.(El botón “Ejecutar” ejecutará estos comandos uno a uno, de arriba a abajo.)

2. Watches: Permite controlar el valor una variable.

-Para controlar el valor de una variable basta con introducir el nombre de está en una de las casillas

-Ejemplo. En la imagen superior puede observarse que la variable “b” posee el valor “10”.

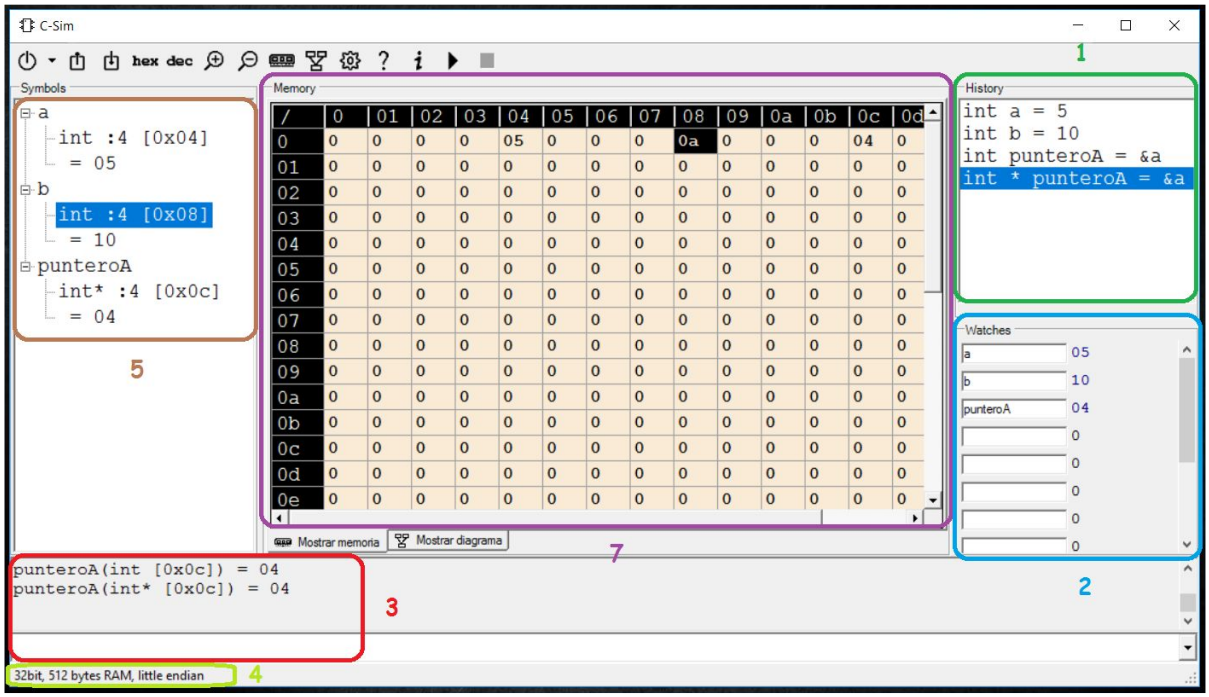
3. Barra de comandos: Permite introducir comandos en el sistema. La sección superior mostrará los últimos comandos introducidos.

4. Barra de información: Muestra información del sistema. Esta informacion es configurable desde la sección opciones (Ver sección “Ventana de opciones”).

5. Symbols: Muestra un árbol con la diferente información de las variables creadas en el sistema (La información de las variables será detallada más adelante).

6. Mapa del sistema: Muestra información gráfica del sistema (Detallado más adelante).

Ventana de Memoria



Los apartados del 1-4 son exactamente iguales que en la sección anterior.

5. Symbols: A pesar de mostrar la misma información que se indicaba en la sección anterior (información de las variables del sistema), además permite seleccionar, con el cursor, una variable, y está, será marcada en negro en el mapa de memoria (como se puede observar en la imagen superior).

7. Mapa de Memoria: En esta sección se muestra el mapa físico de la memoria del sistema.

Menú de Opciones

1. Selector de idioma: Permite seleccionar el idioma de la interfaz entre un gran número de idiomas.

2. Selector de formato: Está opción permite seleccionar el formato de codificación, entre Big endian y Little endian (Para más información, consultar sección “conceptos previos necesarios”)

3. Align variables in memory: Está opción de estar seleccionada, colocara las variables en zonas adyacentes en memoria.

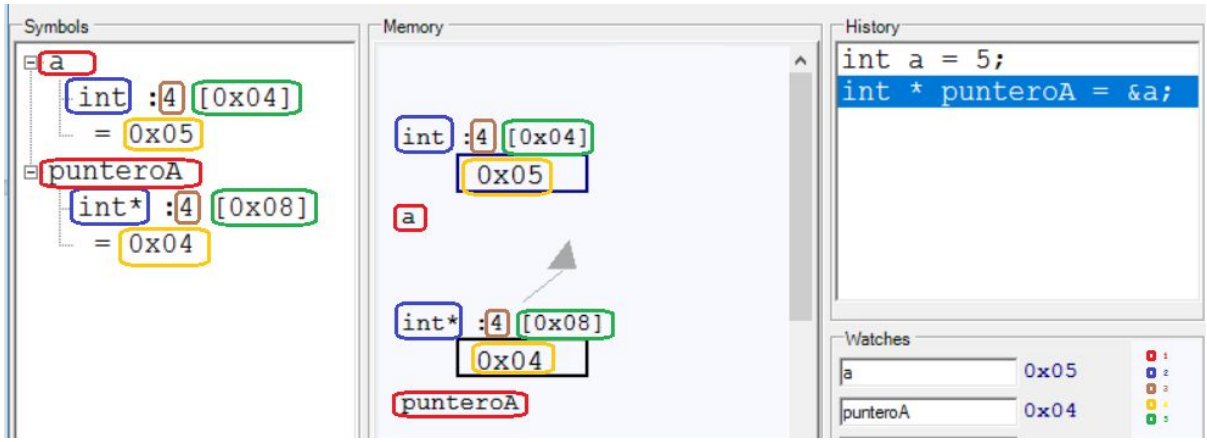
4. Word sice: Permite seleccionar el tamaño de palabra entre 16, 32 o 64 bits (Para más información, consultar sección “conceptos previos necesarios/palabra”)

Manejo básico del simulador

Una vez comprendidos los conceptos previos necesarios y comprendido la interfaz, es hora de comenzar a realizar unos ejercicios básicos.

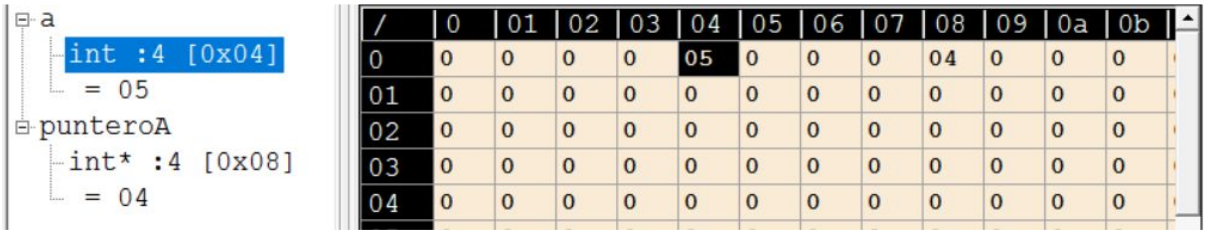
Ejercicio de inicio con punteros
Comenzamos declarando una variable y un puntero que apunte a está.
Codigo:

- int a = 5;
- int * punteroA = &a;



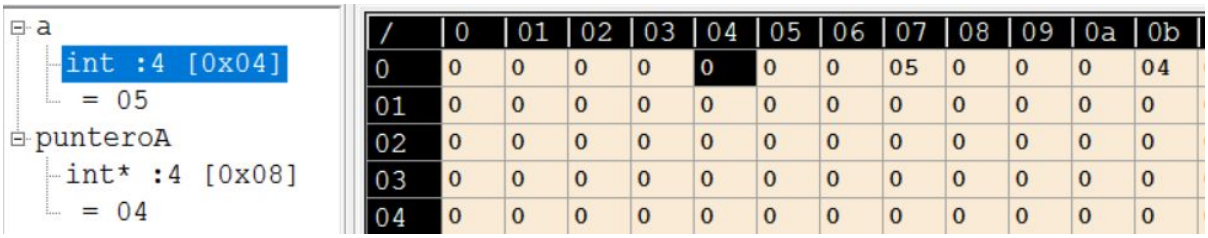
1. Nombre.
2. Tipo.
3. Espacio que ocupa (En está imagen solo hay enteros con un tamaño de palabra de 32 bits, pero os animo a probar a introducir caracteres y a cambiar el tamaño de palabra).
4. Valor de la variable
5. Primera celda de memoria que ocupa la variable. (Es importante entender que la celda a la que está apuntando puede contener el byte más o menos significativo según la codificación)

Ejemplo:



/	0	01	02	03	04	05	06	07	08	09	0a	0b
0	0	0	0	0	05	0	0	0	04	0	0	0
01	0	0	0	0	0	0	0	0	0	0	0	0
02	0	0	0	0	0	0	0	0	0	0	0	0
03	0	0	0	0	0	0	0	0	0	0	0	0
04	0	0	0	0	0	0	0	0	0	0	0	0

En está imagen se puede observar que, con un tamaño de palabra de 32 bits y una codificación little endian, el byte menos significativo de la variable “a” está situado en la celda número 04 (y la variable ocupa desde la celda número 04 a la celda número 07).



/	0	01	02	03	04	05	06	07	08	09	0a	0b
0	0	0	0	0	0	0	0	05	0	0	0	04
01	0	0	0	0	0	0	0	0	0	0	0	0
02	0	0	0	0	0	0	0	0	0	0	0	0
03	0	0	0	0	0	0	0	0	0	0	0	0
04	0	0	0	0	0	0	0	0	0	0	0	0

En cambio en está imagen se puede observar que, con un tamaño de palabra de 32 bits y una codificación big endian, el byte menos significativo de la variable “a” está situado en la celda número 07 (y la variable ocupa desde la celda número 04 a la celda número 07).

Ejercicio printf() y sizeof()

En este ejercicio comenzaremos declarando una variable de tipo entero , una variable de tipo carácter y un puntero de tipo entero.

- int a = 5;
- char c = ‘b’
- int * ptr= &a;

La función sizeof() devuelve exactamente el número de bytes que ocupa cualquier estructura de datos.Esto es útil para saber cuánto espacio en memoria es conveniente reservar.

La función printf() muestra el valor de la variable que se le pasa por parámetro.

- printf(a) //muestra el valor de la variable “a” (5)
- printf(sizeof(int)) // muestra el valor que ocupa un int (en este caso 4 bytes)
- printf(sizeof(a)) // muestra el valor que ocupa la variable “a” (4 bytes)

- printf(sizeof(c)) // muestra el valor que ocupa la variable “c” (1 bytes)
- printf(*ptr) // muestra el valor de la variable a la que apunta (“a” = 5)
- printf(ptr) // muestra el valor de la posición de memoria a la que apunta //(primera celda de la variable “a”)

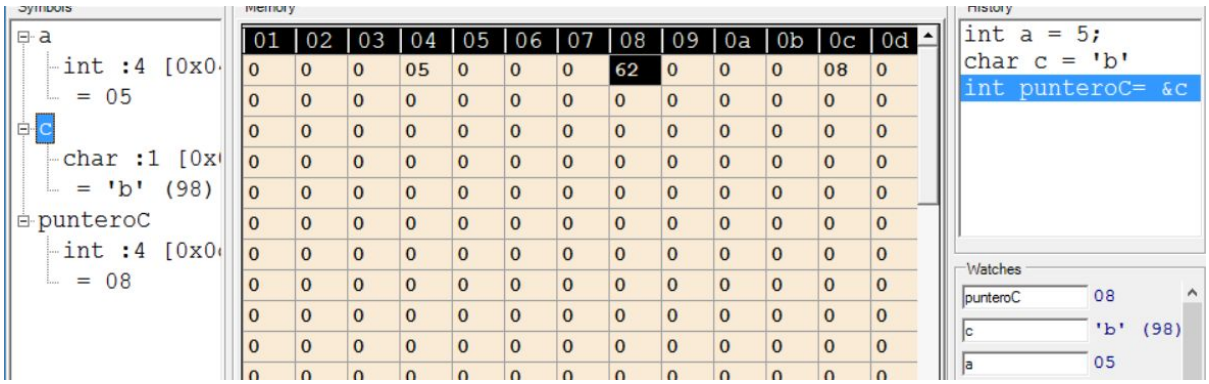
Se recomienda probar diferentes sentencias con diferentes tipos de variables y punteros.

Ejercicio punteros de diferente tipo

En este ejercicio comenzaremos iniciando dos variables (una entera y otra de tipo carácter) y un puntero de tipo entero que apunte a la posición de memoria del carácter.

Codigo:

- int a = 5;
- char c = ‘b’
- int * punteroC = &c;



En la imagen se puede observar que, con un tamaño de palabra de 32 bits y una codificación little endian, podemos observar que la variable de tipo caracter ‘c’ está alojada en la celda 08 y posee el valor 0x62 (es decir, 98 en decimal, que corresponde con el carácter ‘b’).
El puntero de tipo entero (alojado en la celda 0c) apunta a la posición de memoria 08.
Por lo cual, aparentemente, podría parecer que todo está correcto, ya que si preguntamos el valor de *punteroC, el valor devuelto sería 0x62, es decir el valor correcto.

Pero añadiendo el código:

- char d = 'z'

04	05	06	07	08	09	0a	0b	0c
08	0	0	0	62	7a	0	0	08
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0

Podemos observar como el valor de el *punteroC pasa a ser 0x7A62, esto es debido a que el puntero no solo almacena la posición de memoria a la que apunta, si no también el tamaño del tipo de dato a la que está apuntando, por lo cual como el *punteroC es de tipo entero (y el tipo de palabra es de 32 bits) está referenciando al valor guardado en las celdas 08 a 0B.

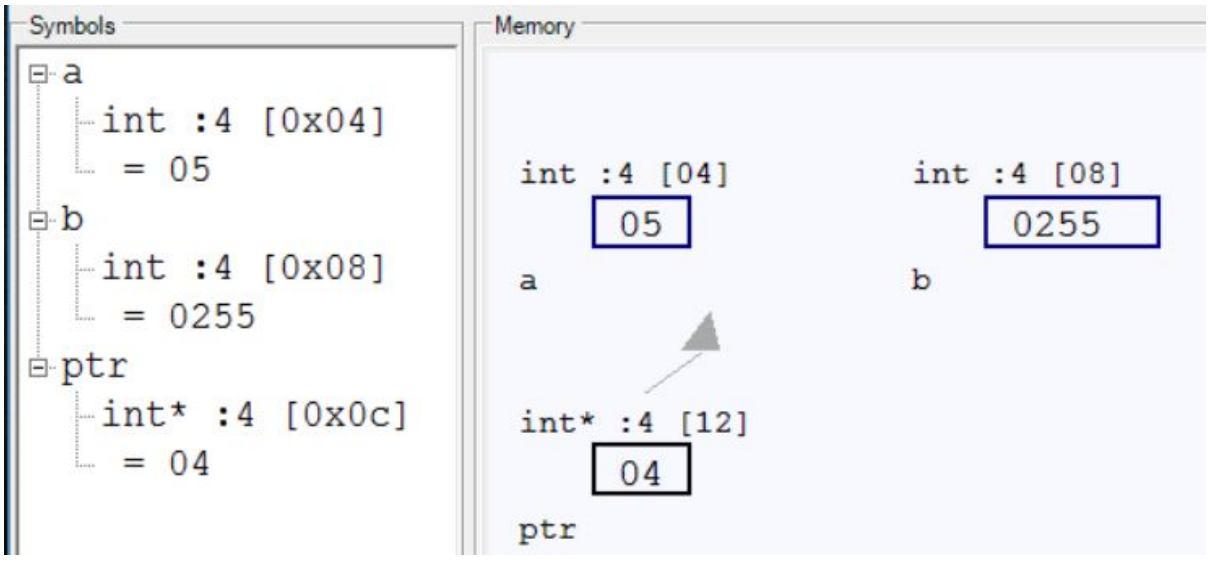
Ejercicio mover punteros

Una de las mayores utilidades de los punteros, es que pueden moverse por diferentes zonas de memoria.

En este ejercicio comenzaremos iniciando dos variables enteras y un puntero que apunte a una de ellas.

Codigo:

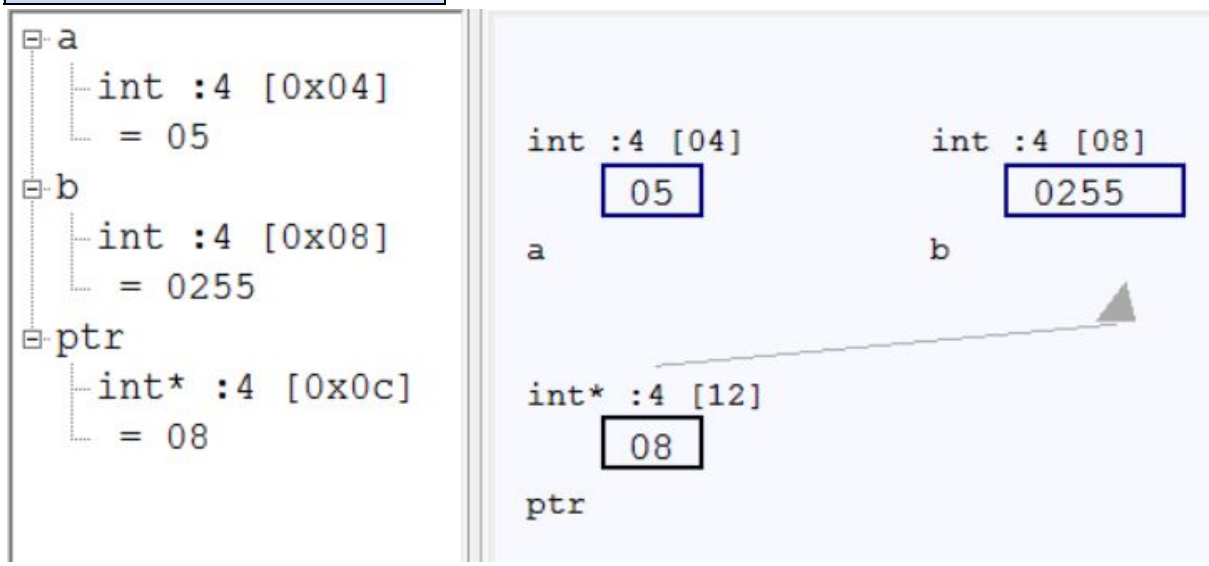
- int a = 5;
- int b = 0xff
- int * ptr= &a;



En esta imagen puede observarse cómo el puntero *ptr apunta a la posición de memoria de la variable “a” (posición de memoria 0x04), por lo cual el comando printf(*ptr) devolverá el valor

05.
Pero añadiendo el código:

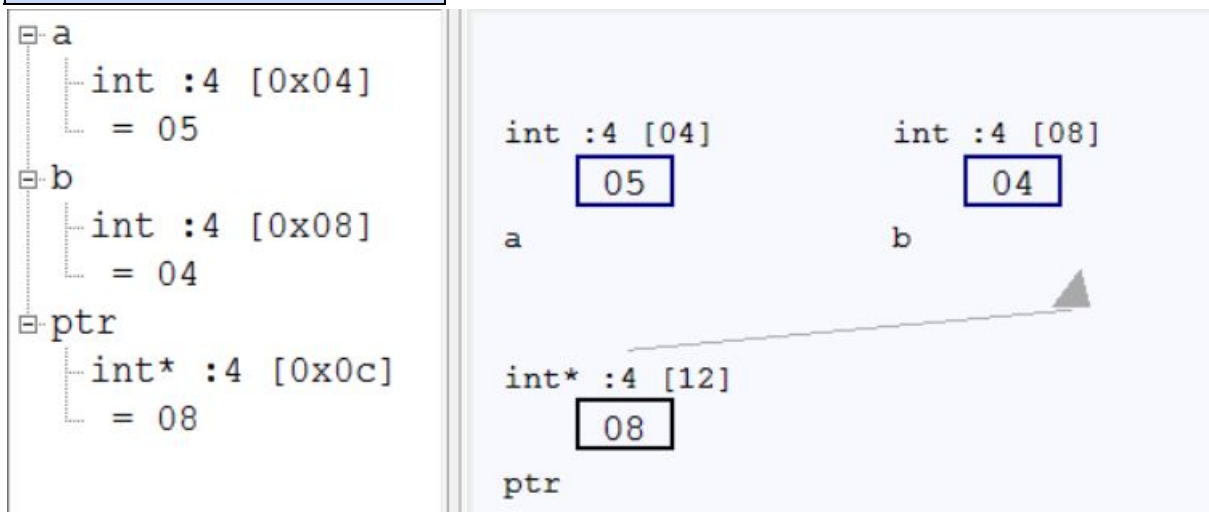
- ptr= &b;



En esta imagen puede observarse cómo el puntero *ptr pasa a apuntar a la posición de memoria de la variable “b” (posición de memoria 0x08), por lo cual el comando printf(*ptr) devolverá el valor 255.

Y añadiendo el código:

- * ptr= &a;



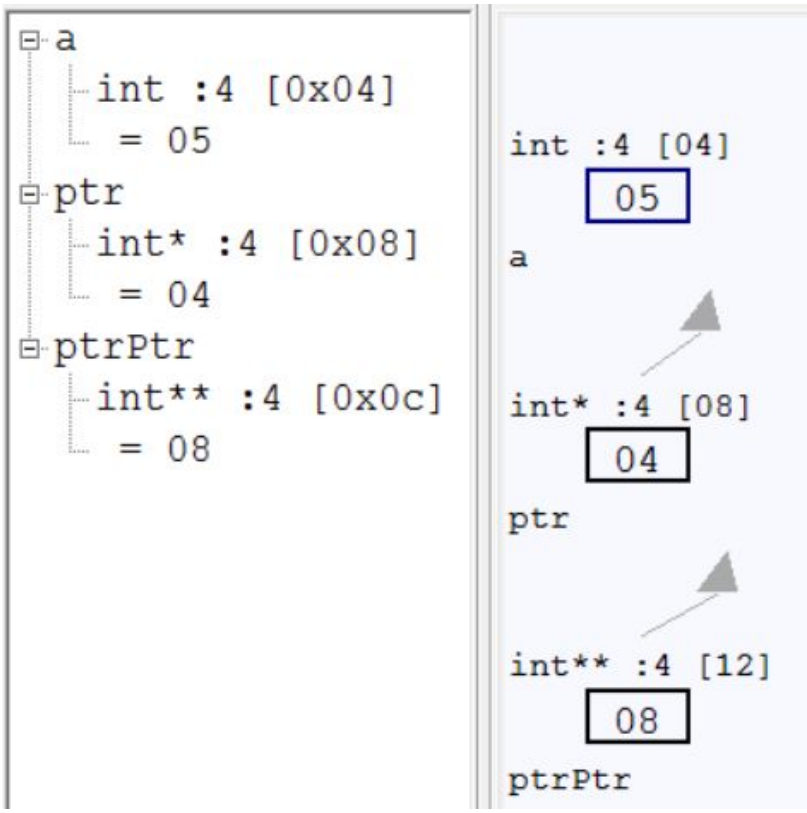
Es importante entender que en esta operación, la posición de memoria de el puntero no varia. Está operación asigna el valor numérico de la primera celda de memoria de la variable “a” (es decir 0x04) a la variable “b”.

Ejercicio Complex pointer creation

En este ejercicio comenzaremos iniciando una variable entera, un puntero que apunte a esta y un puntero que apunte al primer puntero.

Codigo:

- int a = 5;
- int * ptr= &a;
- int **ptr



En esta imagen puede observarse cómo el puntero *ptr apunta a la posición de memoria de la variable “a” (posición de memoria 0x04), y el puntero *ptrPtr apunta a la dirección de memoria del puntero ptr (posición de memoria 0x08).

Ejemplo:

- printf(ptrPtr) //muestra el valor de la celda de memoria donde está el puntero ptr
- printf(*ptrPtr) //muestra el valor de la celda de memoria de la variable “a”
- printf(**ptrPtr)//muestra el valor de la variable “a”

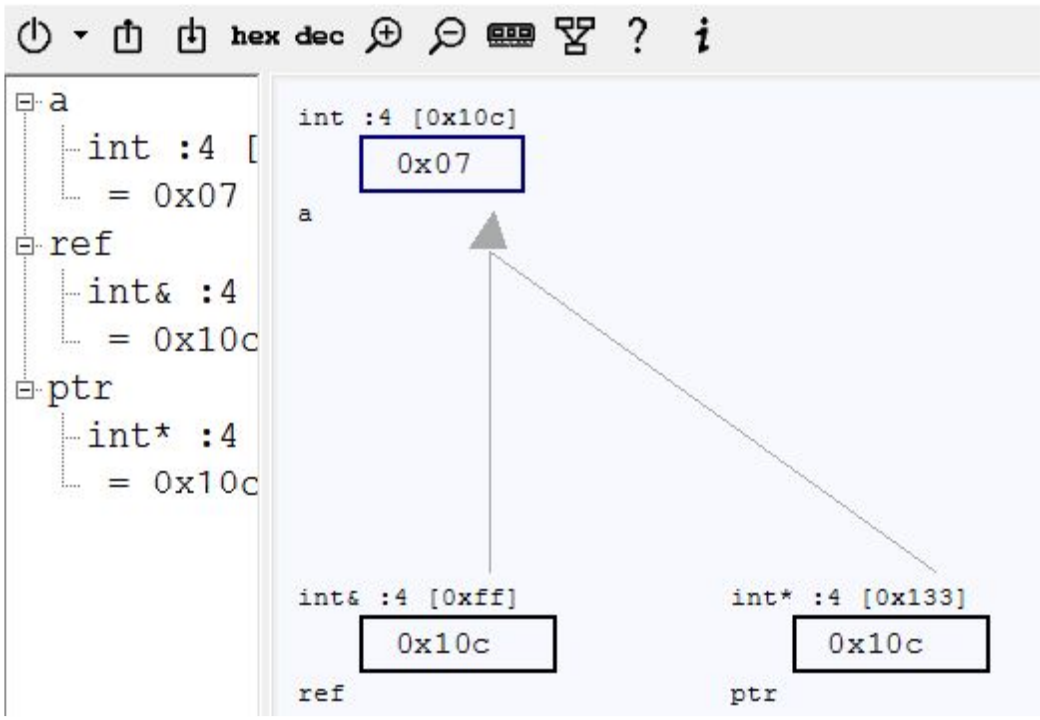
Ejercicio referencia

Una *referencia* (&) es como un puntero constante que se de-referencia automáticamente. Aunque no son propias de C, sino de C++. Estos punteros no precisan de la sintaxis habitual (asterisco) para acceder al valor apuntado.

En este ejercicio comenzaremos iniciando una variable entera y, un puntero y una referencia que apunten a esta.

Codigo:

- int a = 5;
- int &ref = a;
- int * ptr = &ref



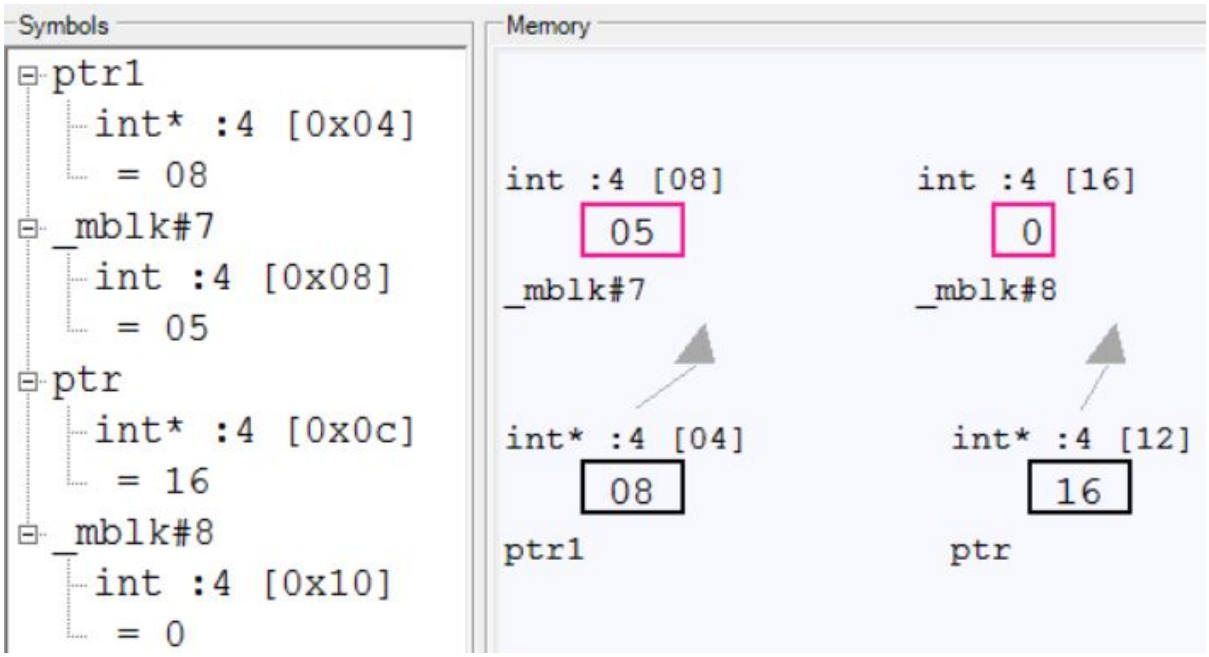
En esta imagen puede observarse cómo el puntero *ptr y la referencia ref apuntan a la misma posición de memoria de la variable “a” (posición de memoria 0x10c).

Ejercicio Definition of variable in heap

En este ejercicio comenzaremos iniciando un dos punteros que apunten a dos variables enteras.

Codigo:

- int * ptr1 = new int(5);
- int * ptr2 = new int;



En esta imagen puede observarse cómo los punteros *ptr1 y *ptr2 apuntan a dos variables enteras.Pero a estas variables solo se tiene acceso a través de los punteros.