1. ¿Están las variables locales en la misma zona de memoria que las variables globales?

Están en diferentes zonas. Por ejemplo, la variable global ‘a’ se ubica en la posición de memoria 0x403010, mientras que la variable local ‘x’ en 0x62fe1c.

2. ¿Y los parámetros argc y argv? ¿Están cerca de las variables globales y/o cerca de las locales?

Están cerca de las locales. La variable argc se ubica en 0x62fe30 y argv en 0x62fe38, y la variable local ‘x’, como se ha visto antes, en 0x62fe1c.

3. ¿Qué valor tienen las variables globales a las que el programa no asignó un valor inicial?

Las variables globales sin valor inicial están inicializadas a 0.

4. ¿Y las locales?

Las locales pueden tener contener cualquier valor (valores basura)

5. ¿En qué orden están las variables globales entre sí? ¿Están en el mismo orden que en el código fuente? ¿Tiene algo que ver el orden con el hecho de que unas están inicializadas y otras no?

Las variables globales están ordenadas según si están inicializadas o no. No siguen el mismo orden visto en el código fuente. Hay dos partes dividas en la memoria para las variables globales: datos inicializados y datos no inicializados.

6. ¿En qué direcciones de memoria se encuentran las funciones? ¿Hay algún otro elemento del programa en esa zona de la memoria?

Por ejemplo, main se encuentra en 0x100401092 o cadenas en 0x10040111d.

Sí, se encuentra las variables globales del programa en esa zona de la memoria como a en 0x100402010 y las cadenas de texto constantes como “hasta luego”

7. ¿En qué dirección está k (variable global perteneciente a mem_dinamica.c)? ¿Está junto a las variables globales de experimento_mem.c o está en una zona separada?

La variable global k está en 0x1004071ac. Sí, está junto a las variables globales de experimento_mem.c como a que está en 0x100402010.

8. Observe la dirección de los parámetros pasados a las funciones. ¿Guardan alguna relación con las direcciones de otros elementos del programa?

Sí, con los variables locales ya que están todas en la pila.

9. Observe las direcciones de las variables locales de las diferentes funciones. Hay algún caso en que dos variables de diferentes funciones ocupan la misma posición en la memoria. ¿Cómo se explica? ¿Puede esto ocasionar algún problema?

Sí, debido a que se guardan en la pila. Una vez terminada de llamar esa función se hace pop de la pila, y en futuras llamadas a otras funciones podrá reutilizar esa misma dirección. Por lo tanto, nunca va a ocasionar ningún problema porque simplemente reutiliza direcciones que ya no están en uso.

10. La función factorial es recursiva (en ocasiones se invoca a sí misma). Observe las direcciones y los valores del parámetro n y la variable local f en las sucesivas invocaciones. ¿Son siempre las mismas direcciones? ¿Qué sentido tiene esto?

No, no son siempre las mismas direcciones. Cada vez que se llama de forma recursiva se hace un push a la pila para las variables locales y argumentos de la función.

11. ¿Cambia la dirección de la propia función factorial en las sucesivas invocaciones anidadas? ¿Puede esto ocasionar algún problema?

No, no cambia la dirección. Esto no ocasiona problema, siempre se llama a la misma función que se ubica siempre en la misma dirección.

12. Observe la función cadenas. ¿Se encuentra la cadena literal "hasta luego" en la misma región de memoria que los arrays a y b? (Por cierto, reconsidere su respuesta a la pregunta 6) ¿Qué sentido tiene esto?

No, no se encuentra en la misma región de memoria. La cadena literal, al tratarse de una constante, se encuentran en su propia zona de memoria para no permitir su modificación. Se encuentran cerca de las variables globales y las funciones.

13. Observe la función mem_dinamica. ¿En qué direcciones se ubican los bloques de memoria dinámica reservados por malloc? ¿Se encuentran cerca de otros elementos del programa o en una región de memoria separada?

Se ubican en direcciones como 0x80003e810 o 0x80003e840. Se encuentra en una región de memoria separada llamada el heap aunque, según la teoría, el heap tiene cerca los datos no iniciados.

14. Observe las direcciones que resultan al evaluar p+1 y q+1 en relación a p y q respectivamente. ¿Cuál es la diferencia, en bytes, en un caso y en el otro? ¿Cómo incide el tipo del puntero en la operación de suma puntero+entero?

La diferencia de p+1 es de 4 bytes mientras que q+1 es de un byte.

Incide en el tipo de dato, ya que p es del tipo int * y como int son 4 bytes en está CPU (x64) va a tener una diferencia de 4 bytes. En el caso de q es lo mismo, pero al ser de tipo char * la diferencia es de 1 byte ya que char ocupa 1 byte.

15. Observe la ambivalencia del tipo char (equivalente a los tipos Char y Byte de Pascal). ¿Qué valor tiene *r cuando se muestra con %c? ¿Y cuando se muestra con %d? ¿Cuál es la correspondencia entre números y caracteres?

Cuando se muestra %c tiene el valor de ‘J’ y cuando es %d se muestra el valor numérico 74

La correspondencia se trata de los caracteres ASCII codificados en números de 8 bits.

16. ¿Coinciden las direcciones de memoria de las variables de un proceso con las del otro? Si coinciden, o si coincidieran... ¿cómo podrían estar en direcciones iguales, al mismo tiempo, y seguir siendo variables diferentes e independientes entre sí?

Sí, coinciden las direcciones de memoria de un proceso a otro.

Esto es debido a que está utilizando la memoria virtual que son espacios independientes, por lo tanto, a pesar de usar las mismas direcciones (que son virtuales) en realidad están mapeadas a direcciones físicas diferentes y así poder seguir siendo variables diferentes e independientes entre sí.