**GUÍA SEMANA 2**

P1) Una solución puede tener varios proyectos; el proyecto compila el ensamble, el ensamble es el ejecutable (.exe o .dll).

P2) Definirle al compilador que va a usar cosas de la biblioteca system.

P3) Son estructuras (Struct) y algunos de estos son int, short, bool, float, entre otros.

P4) Los unsigned no pueden tener valores negativos y los signed sí pueden.

P5) El valor de i no puede ser sbyte porque no está dentro de su rango (-128 a 127).

P6) La diferencia es el tamaño en bits, el int es de 32 bits y el short es de 16 bits.

P7) Sí se puede porque está en el rango.

P8) La f es float, L es long, UL es ulong y m es decimal.

P9) Porque está esperando una respuesta.

P10) Cambiar los tipos de datos.

P11) Console.WriteLine("C:\\Users\\juanfh\\Desktop\\MiArchivo.txt");

C2) No es el resultado esperado porque la parte de a/b se entiende como un entero, perdiéndose la parte decimal de esa división, así que hay que hacer un cast, para que en ese punto se entienda como float.

float result = (float)(a) /(float)(b) + c / d;

P12) El resultado que aparece no es el esperado porque hay un desbordamiento, pues el resultado es mayor al rango que puede ocupar el tipo de variable short; por lo tanto la variable del resultado debe ser de un tipo mayor en el cual quepa el resultado.

P13) La línea 1 es un vector que no tiene un tamaño asignado, mientras que la línea 2 es el mismo vector pero ya con un tamaño asignado.

P14) Es un vector de tipo int al cual se le asigna el tamaño y también, de una vez, se da el valor que tiene cada espacio del vector.

P15) Porque hay una instrucción que le indica al programa a que parte del código saltar cuando se decodifique (Memoria Stack).

P16) Stack.

P17) Se libera automáticamente mediante el garbage collector.

P18) Message queda en el stack y la cadena "Hello!" queda en el heap.

P19) Messages está en el stack, y las direcciones de los objetos y los objetos como tal quedan en el heap.

P20) Numbers queda en el stack y en el heap quedan los valores de la cadena.

P21) Al principio:

|  |  |
| --- | --- |
| **Stack** | **Heap** |
| numbers | 2, 4, 6, 8 |
| message (null) |  |

Al final:

|  |  |
| --- | --- |
| **Stack** | **Heap** |
| numbers (null) | 2, 4, 6, 8 |
| Message (null) |  |

P22) a y b están en el stack. Luego de ejecutar el programa a=3 y b=4.

P23) Imprime el 17 porque tanto a como b tienen la misma dirección de datos en el heap, al cambiar b[0]=17, así mismo está cambiando a[0]=17.

P24) ReturnValue retorna un 3 porque tanto x como y son variables diferentes, mientras que ReturnValue2 retorna un 4 porque tanto x como y están teniendo la dirección al mismo objeto.

P25) Es público para acceder a las variables desde otras clases; además no es necesario que tengan el mismo nombre, eso no importa.

P26) No funciona porque x=-5 está fuera del ámbito donde fue declarada la variable (Dentro del for).

P27) Para referirse a la variable de instancia title definida por fuera del método.

P28) Sirve para separar una oración o cadena en cada una de sus palabras, en diferentes renglones al momento de imprimir.

P29) El método Console.WriteLine() sirve para imprimir en pantalla tanto una variable como una cadena prestablecida que no necesariamente debe estar dentro de una variable, así como sucede en este caso de Console.WriteLine(“Hola mundo”), siempre y cuando se coloque la cadena que se quiere imprimir (Tal cual) entre comillas.

P30) La primera imagen es una estructura, así que es de tipo valor; mientras que la segunda imagen es una clase, por lo que es de tipo referencia.

P31) Cuando es de tipo valor la información de la variable queda almacenada en la memoria stack, por ejemplo:

int a=2;

Cuando es de tipo referencia, se guarda (en el stack) es una dirección de una información que se encuentra almacenada en el heap; por ejemplo:

int[] a =new int[3]  {2,3,4};

Este código crea la variable a que guarda en el stack la dirección del arreglo 2, 3, 4 almacenado en el heap.

C5) Aparece un 10 porque TimeStruct es una estructura, por lo tanto, cuando se llama a la función se crea una copia de la variable en el stack, diferente a la original, entonces cuando la función termina de ejecutarse, el 11 que había creado desaparece y al final, al llamar la variable igual queda la misma original que contenía el 10.

Al principio:

|  |
| --- |
| **Stack** |
|  |
| time=10 |

Durante la función:

|  |
| --- |
| **Stack** |
| time=10+1 |
| time=10 |

Al final:

|  |
| --- |
| **Stack** |
| time=11 (time generado por la función) |
| time=10 (time que se imprime) |

C6) Aparece un 200 porque inicialmente se declara en el stack un array wrapper que guarda la dirección de tres valores en el heap, cuando se llama la función, se está alterando mediante la misma dirección, el segundo valor que se encontraba en el heap (20) se cambia por el nuevo valor (200). Por lo tanto al momento de imprimir wrapper.numbers [1] se imprime el 200, en lugar del 20.

Al principio:

|  |  |
| --- | --- |
| **Stack** | **Heap** |
| wrapper | 10, 20, 30 |
|  |  |

Durante la función:

|  |  |
| --- | --- |
| **Stack** | **Heap** |
| wrapper | 10, 200, 30 |
| wrapper |  |

Al final:

|  |  |
| --- | --- |
| **Stack** | **Heap** |
| wrapper | 10, 200, 30 |
|  |  |