RESPUESTAS GUIA 2 -calentamiento

DANIEL MONTOYA

Los retos se encuentran en el github, en forma de .cs.

P1) Cuando se crea un proyecto, visual studio crea automáticamente una **solución** que lo contiene, la cual puede agrupar varios proyectos y contiene todos los archivos necesarios para crear la aplicación. El **proyecto** administra de forma lógica, compilan y depuran los elementos que conforman parte de la solución. Un **ensamble** se genera cuando el programa es compilado y genera un archivo **.exe** o **.dll,** consiste en una especie de paquete de código precompilado

Dentro de una solución puede haber muchos proyectos.

P2) **Using system** se utiliza cuando programa accederá a las diferentes librerías del computador, por ejemplo, para hacer uso de la parte grafica en los programas se implementan librerías adicionales.

P3) Son estructuras, variables, los diferentes tipos de datos que utiliza el lenguaje de programación, un ejemplo es **int**, que incluye los números enteros en un rango determinado.

P4) Un tipo de dato **unsigned** no incluye números negativos, de esta forma duplicando su rango en los positivos, mientras que los **signed** si pueden tener valores negativos.

P5) **sbyte** es un tipo de dato **unsigned**, lo que significa que no incluye números negativos, por lo que el resultado de la resta de esos valores (siendo un numero negativo) no podrá ser incluido en el tipo de dato unsigned, debido a la restricción del rango.

P6) La diferencia radica en el rango, siendo el **short** una versión de rango menor en comparación con el **int**, esto se utiliza, entre otras razones, para aumentar la **eficiencia** del programa, al no tener que utilizar la misma cantidad de memoria.

P7) Si, porque el rango del tipo de dato **ushort** va de 0 a 65535, por lo cual no hay un incumplimiento del rango.

P8) Esto es para rectificar el tipo de dato utilizado, **f es float**, mientras que **UL es ULong**, **double** es el valor asignado por defecto, por esto carece de terminación, con esta denominación el compilador puede entender mejor el dato.

P9) Porque **el programa espera una respuesta** por parte del usuario, en este caso, espera que el usuario escriba una línea para leerla.

P10) Sirve para **convertir** un tipo de dato a otro, por ejemplo de string a int;

C1)

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

C2) El resultado no es lo que esperaba.

P12) El programa declara dos variables tipo short a y b, a las cuales asigna un valor inicial de 30000, luego declara otra variable tipo shor llamada sum, que esta determinada por la suma de a y b. Finalmente el programa imprime el resultado de la suma. En este programa hay un desbordamiento de la variable, por el sobrepaso del rango de short en el valor de la suma, generando una respuesta incorrecta, se puede solucionar declarando que de la suma resultara un tipo de variable con mayor rango (ej. ushort) que incluya el valor del resultado (En la línea 12, antes de la declaración de la suma, se reemplace el short por el nuevo tipo de dato).

P13) Ambas declara una variable inicial de tipo array de int, la diferencia esta en que la segunda línea define un valor inicial para la variable, mientras que la primera no

P14) Esta línea define un array de variables tipo int, de tamaño 10 y define estos valores.

C3)

C4)

Los programas tienen procesos, ,los cuales utilizan memoria RAM

De tipo STACK y HEAP, que almacenan datos, como variables.

El STACK se usa para seguir el estado actual de programa y almacenar variables locales

El stack se llena y vacia constantemente

HEAP: zona de almacenamiento dinamico, para almacenar info que se requerirá en tiempo de ejecución(genralmente)

Garbage collector busca los objetos no utilizados y los borra automáticamente

-----------------------------------------------------------------------------------------------------------------------------------

P15) La instrucción de la línea 12 indica que el programa salte a la línea 18 a la vez que anota la posición retorno (13), y la 19, con el retorno lo devuelve a la línea anotada (13)

En el stack se almacena la dirección de retorno

P16) En el stack, intercambio de información, almacenamiento temporal.

P17) El espacio permanece allí hasta que es liberado manualmente o por el garbage collector (si se pierde su dirección y se vuelve inaccesible)

value no es el objeto pero tiene la dirección de el

STACK

Result = Dir-------------------dirreccion a My Int

pValue

Dir Retorno

Value--------------------------- dirreccion a My Int

Result es la referencia, pointer

P18) Message, por ser variable local de tipo referencia se almacena en el stack, la cadena, por ser un objeto tipo string permanece en el heap.

P19) La variable Message queda en el stack, que apunta al arreglo (dentro del heap), el cual tiene las tres direcciones de los tres objetos (hello, world, ¡) que están en el heap, dentro de un objeto de arreglo sin nombre.

P20) Numbers queda almacenado en stack, la new int crea un objeto de arrays que se encuentra en el heap, el cual tiene los int dentro de sí mismo, sin necesidad de contener las direcciones de cada uno.

P21) null es para declarar que no hay direcciones en una variable de referencia

|  |
| --- |
|  |
|  |
| numbers(null) |
| message(null) |
| **STACK** |

|  |
| --- |
|  |
|  |
|  |
| 2 4 6 8 |
| **HEAP** |

P22) a=3 y b=4, a y b están en el stack

P23) imprime 17, porque a contenia la dirección del arreglo, y al pasarle a b esa misma dirección se afecta dicha posición del heap, de modo que si b la afecta, al acceder a, ya abra cambiado.

P24) 3 y 4, esto es porque en el primer returnValue se modifican directamente y por separado los valores almacenados en las variables, mientras que en el segundo se modifican a través de variables de tipo referencia que contienen las direcciones donde están almacenados los valores, cuando se realiza y = x se almacena la dirección que almacena x en la de y, de modo que ambas apuntan a la misma posición en el heap y cuando se realiza y.MyValue se afecta el valor almacenado en aquella dirección, de modo que afecta también a x, porque apunta al mismo lugar, my int es tipo referencia

P25) Para que pueda ser invocado desde cualquier parte por fuera de la clase, como en el main, si debe tener el mismo nombre porque así fue diseñado.

P26) No en ese estado sacara un error porque la última línea, en la que se cambia el valor de x por -5 está fuera del for, por lo que no existe en el contexto actual.

P27) Como la variable propia de la clase book, definida al inicio de la clase también se llama title al igual que la definida en este método, this.title se usa para referirse a la que se encuentra por fuera del método, pero dentro de la clase.

P28) Devuelve la longitud de un array formado por las sub-strings que conforman la frase original, determinadas cada una por la separación de un espacio, de esa forma partiendo la frase en palabras.

P29) Que llama un método estático “WriteLine” que pertenece a la clase “Console”, el cual imprime en pantalla la frase entre comillas, sin necesidad de acceder a un objeto en específico de tipo Console.

P30) En la primera imagen se declara una estructura mientras que en la segunda se declara una clase, struct es tipo valor, class es tipo referencial, el contenido de las estructuras se guardan en el stack, mientras que el de las clases en el heap

P31) La variable de tipo estructura contendrá todos los datos directamente, mientras que la clase solo contendrá las direcciones a estos datos.

struct Point

{

public int x, y;

public Point(int x, int y) {

this.x = x;

this.y = y;

}

}

Point a = new Point(10, 10);

Point b = a;

a.x = 100;

System.Console.WriteLine(b.x);

De esta forma el programa imprime 10, porque a y B contienen el valor en si mismo, mientras que si point fuese una clase, a y b contendrían las direcciones de forma que al igualar b a a, ambos contienen la misma dirección y afectar un valor dentro de este lo afecta para ambos.(el programa imprimiría 100)

|  |
| --- |
|  |
|  |
|  |
|  |
| **HEAP** |

C5) 11, pero el resultado es 10, porque TimeStruct es una estructura y no una clase, esto es porque la dirección de retorno ignora el cambio anterior y retorna a cuando seconds era igual a 10, debido a que el time fue copiado una segunda vez en el stack, ese fue alterado no el de retorno, con la clase si funciona porque la copia no contiene un valor diferente sino la dirección es la que es alterada, de modo que con el retorno si imprime lo correcto.

|  |
| --- |
|  |
|  |
| seconds=10 |
| time |
| **STACK** |

C6) se esperaría un 200 y es lo correcto, el programa altera directamente la variable numbers sin hacer uso de geters o setters, además aquí puede ser accedida la variables numbers porque es publica y no privada

|  |
| --- |
|  |
|  |
| numbers=200 |
| wrapper |
| **STACK** |

|  |
| --- |
|  |
|  |
|  |
|  |
| **HEAP** |