1) Sea el siguiente código:

```
object o1 = "A";
object o2 = o1;
o2 = "Z";
Console.WriteLine(o1 + " " + o2);
```

El tipo **object** es un tipo referencia, por lo tanto luego de la sentencia **o2 = o1** ambas variables están apuntando a la misma dirección. ¿Cómo explica entonces que el resultado en la consola no sea "**Z Z**"?

Cuando se le asigna a o2=”Z”; Genera un nuevo objeto Con una nueva dirección de memoria en la pila.

2) Qué líneas del siguiente código provocan conversiones *boxing* y *unboxing*.

```
char c1 = 'A';
string st1 = "A";
object o1 = c1;
object o2 = st1;
char c2 = (char)o1;
string st2 = (string)o2;
```

Boxing = Object o1 = c1; //

Unboxing = char c2 = (char)o1; //

6) Supongamos que Program.cs sólo tiene las siguientes dos líneas:

```
int i;
Console.WriteLine(i);
```

¿Por qué no compila?

No compila porque i no esta inicializada.

7) ¿Cuál es la salida por consola del siguiente fragmento de código? ¿Por qué la tercera y sexta línea producen resultados diferentes?

```
char c1 = 'A';
char c2 = 'A';
Console.WriteLine(c1 == c2);
object o1 = c1;
object o2 = c2;
Console.WriteLine(o1 == o2);
```

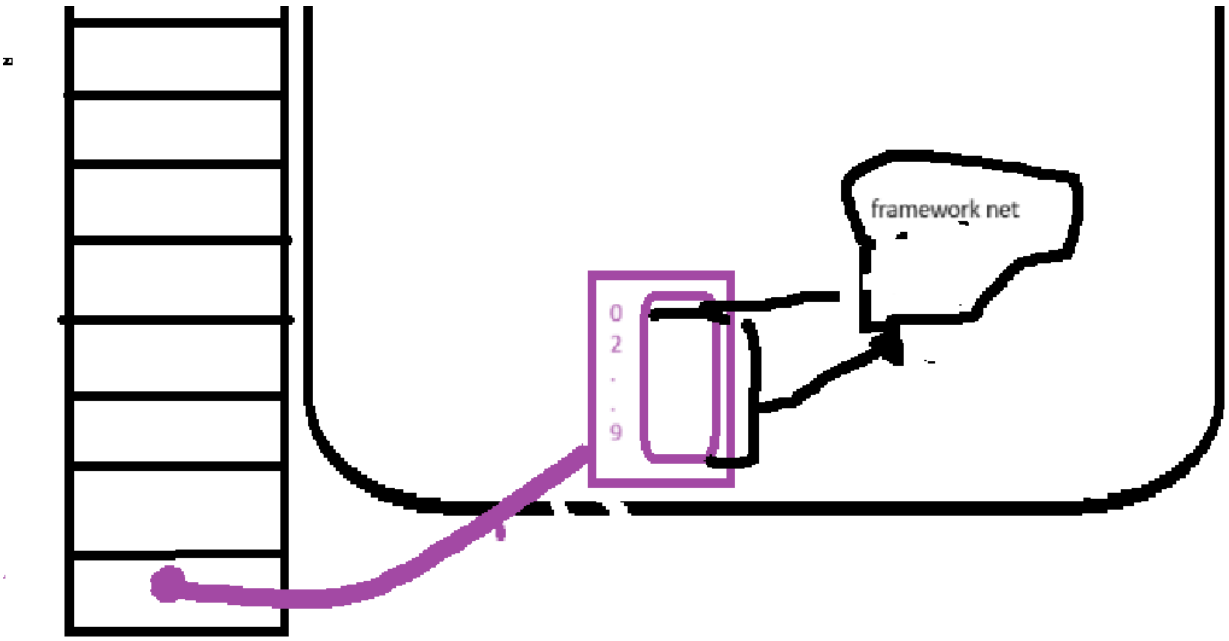
Producen resultados diferentes ya que en la tercera compara dos por tipo de valor por ende analiza el valor de esas variables y el otro analiza de dos variables referencia por ende analiza si apuntan a la misma dirección. (Si se quisiera analizar el contenido debería utilizar o1.Equals(o2).

8) Investigar acerca de la clase **StringBuilder** del espacio de nombre **System.Text** ¿En qué circunstancias es preferible utilizar **StringBuilder** en lugar de utilizar **string**? Implementar un caso de ejemplo en el que el rendimiento sea claramente superior utilizando **StringBuilder** en lugar de **string** y otro en el que no.

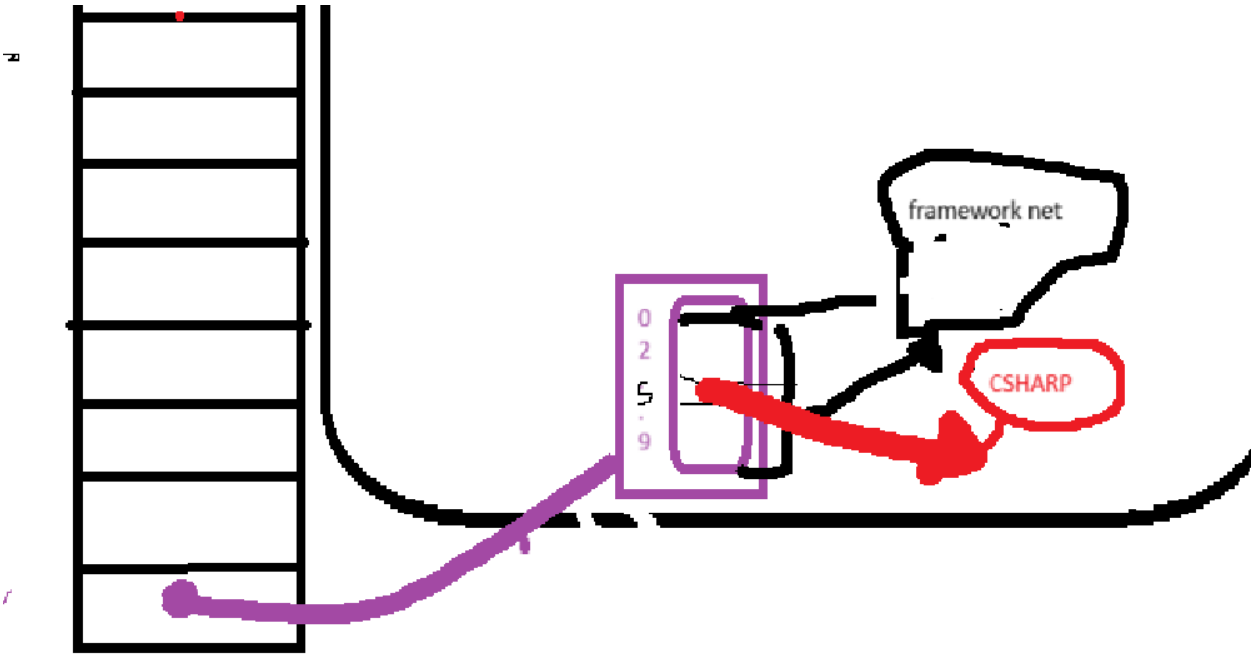
Cuando se quiere se quiere optimizar la memoria conviene utilizar un stringBuilder para no ir creando objetos y ocupando mas memoria.

10) Comprobar el funcionamiento del siguiente programa y dibujar el estado de la pila y la memoria *heap* cuando la ejecución alcanza los puntos indicados (comentarios en el código)

```
using System.Text;

object[] v = new object[10];
v[0] = new StringBuilder("Net");
for (int i = 1; i < 10; i++)
{
   v[i] = v[i - 1];
}
(v[5] as StringBuilder).Insert(0, "Framework .");
foreach (StringBuilder s in v)
   Console.WriteLine(s);

//dibujar el estado de la pila y la mem. heap
//en este punto de la ejecución

v[5] = new StringBuilder("CSharp");
foreach (StringBuilder s in v)
   Console.WriteLine(s);

//dibujar el estado de la pila y la mem. heap
//en este punto de la ejecución
```



En el primer caso.



En el segundo.

11) ¿Para qué sirve el método **Split** de la clase **string**? Usarlo para escribir en la consola todas las palabras (una por línea) de una frase ingresada por consola por el usuario.

Divide un String es partes mas pequeñas.

Ejercicio.cs

```
1   String st = Console.ReadLine();
2   String [] st2 = st.Split ();  //  se utliza un arreglo para almacenar las palabras.
3   foreach (String s in st2)  //foreach para recorrer todo un vector, s para mostrar el vector
4   {
5       Console.WriteLine(s);
6   }
```

12) Definir el tipo de datos enumerativo llamado **Meses** y utilizarlo para:

a) Imprimir en la consola el nombre de cada uno de los meses en orden inverso (diciembre, noviembre, octubre …, enero)

b) Solicitar al usuario que ingrese un texto y responder si el texto tipeado corresponde al nombre de un mes

Nota: en todos los casos utilizar un **for** iterando sobre una variable de tipo **Meses**

13) ¿Cuál es la salida por consola si no se pasan argumentos por la línea de comandos?

```
Console.WriteLine(args == null);
Console.WriteLine(args.Length);
```

False,0

14) ¿Qué hace la instrucción? ¿Asigna a la variable vector el valor **null**?

```
int[]? vector = new int[0];
```

Si bien el vector puede ser null , se crea el vector vacio ósea sin elementos,Por lo que no es null

15) Determinar qué hace el siguiente programa y explicar qué sucede si no se pasan parámetros cuando se invoca desde la línea de comandos.

```
Console.WriteLine("¡Hola {0}!", args[0]);
```

Se quiere acceder a una posicion fuera de rango. Ya que no tienen parámetros el arreglo

16) Escribir un programa que reciba una lista de nombres como parámetro por la línea de comandos e imprima por consola un saludo personalizado para cada uno de ellos.

a) Utilizando la sentencia **for**

b) Utilizando la sentencia **foreach**

a)

```
for (Meses m = Meses.Diciembre;m >=Meses.Enero;m--)
    Console.WriteLine("El mes es : " + m);


3 references
enum Meses
{
    1 reference | 0 references | 0 references | 0 references | 0 references | 0 references | 0 references | 0 references | 0 references | 0 refer
    Enero,Febrero,Marzo,Abril,Mayo,Junio,Julio,Agosto,Septiembre,Octubre,Noviembre,Diciembre
}
```

b)

```
Console.WriteLine("INGRESE UN MES DEL ANIO");
String st = Console.ReadLine();
for (Meses m = Meses.Enero;m <=Meses.Diciembre;m++){
    if (m.ToString()== st)  // el to string para el contenido de m compararlo con un string.  PARA MI ESTA MAL.
    {
        Console.WriteLine("Coincide");
        break;
    }
}
3 references
enum Meses
{
    1 reference | 0 references | 0 references | 0 references | 0 references | 0 references | 0 references | 0 references | 0 references | 0 references | 0 references | 1 reference
    Enero,Febrero,Marzo,Abril,Mayo,Junio,Julio,Agosto,Septiembre,Octubre,Noviembre,Diciembre
}
```

16) Escribir un programa que reciba una lista de nombres como parámetro por la línea de comandos e imprima por consola un saludo personalizado para cada uno de ellos. a) Utilizando la sentencia for b) Utilizando la sentencia foreach

a)
```
for (int i=0; i<args.Length;i++){
    Console.WriteLine("Hola " + args[i]);
}
```

b)
```
foreach(String st in args){
    Console.WriteLine("Hola "+ st);
}
```

17) Implementar un programa que muestre todos los números primos entre 1 y un número natural dado (pasado al programa como argumento por la línea de comandos). Definir el método **`bool EsPrimo(int n)`** que devuelve **`true`** sólo si *n* es primo. Esta función debe comprobar si *n* es divisible por algún número entero entre 2 y la raíz cuadrada de *n*. (Nota: **`Math.Sqrt(d)`** devuelve la raíz cuadrada de **`d`**)

```
int dim = int.Parse(args[0]);
for (int i=0;i<=dim;i++)
{
    if (EsPrimo(i)){   //para cada numero imprimo sus valores primos
        Console.WriteLine("EL NUMERO " + i + " ES PRIMO");
    }
}

bool EsPrimo(int n )
{
    if (n<2)
        return false;
    else
```

```
            for (int i=2;i<=Math.Sqrt(n);i++)
            {
                if (n % i == 0){
                    return false;
                }
        }
    return true;
}
```

18) Escribir una función (método **`int Fac(int n)`**) que calcule el factorial de un número *n* pasado al programa como parámetro por la línea de comando

a) Definiendo una función no recursiva

b) Definiendo una función recursiva

c) idem a b) pero con *expression-bodied methods* (**Tip**: utilizar el operador condicional ternario)

```
int num = int.Parse(args[0]);
Console.WriteLine(factorial(num));

int factorial (int n)
{
    int suma = 1;
    if (n == 0 | n ==1 )
        return 1;
    else
        for (int i=n;i>=2;i--)
        {                                           //Normal.
            suma = suma * i;
        }
    return suma;
}
```

```
int num = int.Parse(args[0]);
Console.WriteLine(factorial(num));

int factorial (int n)
{                                       //factorial
    int suma = n;
    if (n==1 || n==0)
        return 1;
    else
         return n * factorial(n-1);
}
```

```
int num = int.Parse(args[0]);
Console.WriteLine(factorial(num));


int factorial (int n) => (n==1) ? 1 :n * (factorial(n-1));     //operador
ternario
```

19) Idem. al ejercicio 18.a) y 18.b) pero devolviendo el resultado en un parámetro de salida **void Fac(int n, out int_f)**

```
int result = 0;
int num = int.Parse(args[0]);
Factorial(num,out result);
Console.WriteLine(result);

void Factorial (int n,out int result)
{
    int suma = 1;
    if ((n == 1)||(n==0))                                   /A
        suma = 1;
    else
        for (int i=n;i>=2;i--)
            suma = suma*i;
    result = suma;
}
```

```
int num = int.Parse(args[0]);
int result;
factorial(num,out result);
Console.WriteLine(result);
void factorial (int n, out int result)
{
    int suma = 1 ;
    if (n<=1){
        result = 1;
    }                                                       //B
    else
    {
        factorial(n-1,out suma);
        result = n*suma;
    }                       //factorial
```

20) Codificar el método **Swap** que recibe 2 parámetros enteros e intercambia sus valores. El cambio debe apreciarse en el método invocador.


```
Console.WriteLine("Ingrese dos valores");
int a = int.Parse(Console.ReadLine());
int b = int.Parse(Console.ReadLine());
Swap(a,b);

void Swap (int a,int b)
{
    Console.WriteLine("El numero A es : " +a);
    Console.WriteLine("El numero b es : " +b);
    int item=a;
    a=b;
    b=item;
    Console.WriteLine("El numero A es : " +a);
    Console.WriteLine("El numero b es : " +b);
}
```

21) Codificar el método **Imprimir** para que el siguiente código produzca la salida por consola que se observa. Considerar que el usuario del método **Imprimir** podría querer más adelante imprimir otros datos, posiblemente de otros tipos pasando una cantidad distinta de parámetros cada vez que invoque el método. **Tip**: usar **params**

```
Imprimir(1, "casa", 'A', 3.4, DayOfWeek.Saturday);
Imprimir(1, 2, "tres");
Imprimir();
Imprimir("------------");
```

```
Imprimir(1,3,5,"HOLA MUNDO",Tamaño.chico);

void  Imprimir(params object[]vec)
{
    foreach(object o in vec)
    {
       Console.WriteLine(o.ToString());
    }
}
```