Seminario de Lenguajes (.NET)

Práctica 3

Ejercicios obligatorios para el coloquio: Los **ejercicios 12 y 14** de esta práctica forman parte del conjunto de ejercicios de programación obligatorios que el alumno debe resolver y exponer de manera oral sobre máquina el día del coloquio hacia final de la cursada.

1. Ejecutar y analizar cuidadosamente el siguiente programa:

```
using System;
class Programa
{
   static void Main()
   {
       Console.CursorVisible = false;
       ConsoleKeyInfo k = Console.ReadKey(true);
       while (k.Key != ConsoleKey.End)
       {
           Console.Clear();
           Console.Write($"{k.Modifiers}-{k.Key}-{k.KeyChar}");
           k = Console.ReadKey(true);
       }
   }
}
```

   Comprobar tipeando teclas y modificadores (**`shift`**, **`ctrl`**, **`alt`**) para apreciar de qué manera se puede acceder a esta información en el código del programa.

2. Implementar un método estático para imprimir por consola todos los elementos de una matriz (arreglo de dos dimensiones) pasada como parámetro. Debe imprimir todos los elementos de una fila en la misma línea en la consola.

```
static void ImprimirMatriz(double[,] matriz)
```

   Ayuda: Si **`A`** es un arreglo, **`A.GetLength(i)`** devuelve la longitud del arreglo en la dimensión **`i`**.

3. Idem al anterior pero ahora con un parámetro más que representa la plantilla de formato que debe aplicarse a los números al imprimirse. La plantilla de formato es un string de acuerdo a las convenciones de formato compuesto, por ejemplo **`"0.0"`** implica escribir los valores reales con un dígito para la parte decimal. Observar que no hay inconveniente para implementar dos métodos con el mismo nombre siempre que **NO** lleven la misma cantidad de parámetros con los mismos tipos y en el mismo orden (sobrecarga de métodos).

```
static void ImprimirMatriz(double[,] matriz, string formatString)
```

4. Implementar los métodos **`GetDiagonalPrincipal`** y **`GetDiagonalSecundaria`** que devuelven un vector con la diagonal correspondiente de la matriz pasada como parámetro. Si la matriz no es cuadrada generar una excepción **`ArgumentException`** con un mensaje explicativo.

```
static double[] GetDiagonalPrincipal(double[,] matriz)

static double[] GetDiagonalSecundaria(double[,] matriz)
```

5. Implementar un método estático que devuelva un arreglo de arreglos con los mismos elementos que la matriz pasada como parámetro:

```
static double[][] GetArregloDeArreglo(double [,] matriz)
```

6. Implementar los siguientes métodos estáticos que devuelvan la suma, resta y multiplicación de matrices pasadas como parámetros. Para el caso de la suma y la resta, las matrices deben ser del mismo tamaño, en caso de no serlo devolver **`null`**. Para el caso de la multiplicación la cantidad de columnas de A debe ser igual a la cantidad de filas de B, en caso contrario generar una excepción **`ArgumentException`** con un mensaje explicativo.

```
static double[,] Suma(double[,] A, double[,] B)

static double[,] Resta(double[,] A, double[,] B)

static double[,] Multiplicacion(double[,] A, double[,] B)
```

7. ¿De qué tipo quedan definidas las variables **`x`**, **`y`**, **`z`** en el siguiente código?

```
static void Main(string[] args)
{
   int i = 10;
   var x = i * 1.0;
   var y = 1f;
   var z = i * y;
}
```

8. ¿Qué líneas del siguiente método provocan error de compilación y por qué?

```
static void Main(string[] args)
{
   var a = 3L;
   dynamic b = 32;
   object obj = 3;
   a = a * 2.0;
   b = b * 2.0;
   b = "hola";
   obj = b;
   b = b + 11;
   obj = obj + 11;
   var c = new { Nombre = "Juan" };
   var d = new { Nombre = "María" };
   var e = new { Nombre = "Maria", Edad = 20 };
   var f = new { Edad = 20, Nombre = "Maria" };
   f.Edad = 22;
   d = c;
   e = d;
   f = e;
}
```

9. Señalar errores de compilación y/o ejecución en el siguiente código

```
static void Main(string[] args)
{
   object obj = new int[10];
   dynamic dyn = 13;
   Console.WriteLine(obj.Length);
   Console.WriteLine(dyn.Length);
}
```

10. Verificar con un par de ejemplos si la sección opcional [:*formatString*] de formatos compuestos redondea o trunca cuando se formatean números reales restringiendo la cantidad de decimales. Plantear los ejemplos con cadenas de formato compuesto y con cadenas interpoladas.

11. Señalar errores de ejecución en el siguiente código

```
static void Main(string[] args)
{
   ArrayList a = new ArrayList() {1,2,3,4};
   a.Remove(5);
   a.RemoveAt(5);
}
```

12. Utilizar la clase **Queue** (cola) para implementar un programa que realice el cifrado de un texto aplicando la técnica de clave repetitiva. La técnica de clave repetitiva consiste en desplazar un carácter una cantidad constante de acuerdo a la lista de valores que se encuentra en la clave. Por ejemplo: para la siguiente tabla de caracteres:

A	B	C	D	E	F	G	H	I	J	K	L	M	N	Ñ	O	P	Q	R	S	T	U	V	W	X	Y	Z	sp
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28

Si la clave es {5,3,9,7} y el mensaje a cifrar **"HOLA MUNDO"**
Se cifra de la siguiente manera:

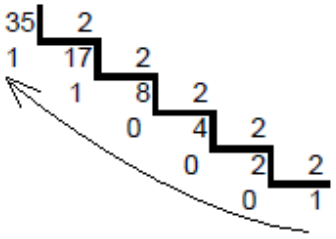
H	**O**	**L**	**A**	**sp**	**M**	**U**	**N**	**D**	**O**	← **Mensaje original**
8	16	12	1	28	13	22	14	4	16	← **Código sin cifrar**
5	3	9	7	5	3	9	7	5	3	← **Clave repetida**
13	19	21	8	5	16	3	21	9	19	← **Código cifrado**
M	**R**	**T**	**H**	**E**	**O**	**C**	**T**	**I**	**R**	← **Mensaje cifrado**

A cada carácter del mensaje original se le suma la cantidad indicada en la clave. En el caso que la suma fuese mayor que 28 se debe volver a contar desde el principio, Tenga en cuenta que para resolver este problema debe utilizar una cola que guarde la clave y que a medida que saque elementos de la cola los tiene que agregar nuevamente para poder utilizarla en forma repetitiva. Programe una rutina para cifrar y otra para descifrar.

13. Realizar un análisis sintáctico para determinar si los paréntesis en una expresión aritmética están bien balanceados. Verificar que por cada paréntesis de apertura exista uno de cierre en la cadena de entrada. Utilizar una pila para resolverlo. Los paréntesis de apertura de la entrada se almacenan en una pila hasta encontrar uno de cierre, realizándose entonces la extracción del último paréntesis de apertura almacenado. Si durante el proceso se lee un paréntesis de cierre y

la pila está vacía, entonces la cadena es incorrecta. Al finalizar el análisis, la pila debe quedar vacía para que la cadena leída sea aceptada, de lo contrario la misma no es válida.

14. Utilizar la clase **`Stack`** (pila) para implementar un programa que pase un número en base 10 a otra base realizando divisiones sucesivas. Por ejemplo para pasar 35 en base 10 a binario dividimos sucesivamente por dos hasta encontrar un cociente menor que el divisor, luego el resultado se obtiene leyendo de abajo hacia arriba el cociente de la última división seguida por todos los restos.



El resultado es 100011

15. ¿Qué salida por la consola produce el siguiente código?

```
static void Main()
{
   int x = 0;
   try
   {
      Console.WriteLine(1.0 / x);
      Console.WriteLine(1 / x);
      Console.WriteLine("todo OK");
   }
   catch (Exception e)
   {
      Console.WriteLine(e.Message);
   }
}
```

¿Qué se puede inferir respecto de la excepción división por cero en relación al tipo de los operandos?

16. Implementar un programa que permita al usuario ingresar números por la consola. Debe ingresarse un número por línea finalizado el proceso cuando el usuario ingresa una línea vacía. A medida que se van ingresando los valores el sistema debe mostrar por la consola la suma acumulada de los mismos. Utilizar la instrucción **`try/catch`** para validar que la entrada ingresada sea un número válido, en caso contrario advertir con un mensaje al usuario y proseguir con el ingreso de datos.

17. Implementar un programa calculadora que calcule una expresión ingresada por el usuario correspondiente a una operación binaria de las cuatro elementales (ejemplo “44.5/456”, “456*45”, “549-12”, “54+6”). Utilizar **`try/catch`** para validar los números y controlar cualquier tipo de excepción que pudiese ocurrir en la evaluación de la expresión.

18. Cuál es la salida por consola del siguiente programa:

```
using System;
class Ejercicio
{
    static void Main(string[] args) {
       try {
           Metodo1();
       } catch {
           Console.WriteLine("Método 1 propagó una excepción no tratada");
       }
       try {
           Metodo2();
       } catch {
           Console.WriteLine("Método 2 propagó una excepción no tratada");
       }
       try {
           Metodo3();
       } catch {
           Console.WriteLine("Método 3 propagó una excepción");
       }
    }
    static void Metodo1() {
       object obj = "hola";
       try {
          int i = (int)obj;
       } finally {
          Console.WriteLine("Bloque finally en Metodo1");
       }
    }
    static void Metodo2() {
       object obj = "hola";
       try {
           int i = (int)obj;
       }
       catch (OverflowException) {
           Console.WriteLine("Overflow");
       }
    }
    static void Metodo3() {
       object obj = "hola";
       try {
           int i = (int)obj;
       }
       catch (InvalidCastException) {
           Console.WriteLine("Excepción InvalidCast en Metodo3");
           throw;
       }
    }
}
```