

C# Y .NET 8

Descripción

IDE, variables, si condicional, ciclos, funciones, arreglos, programación orientada a objetos, estructuras de datos, LINQ, simulaciones, algoritmos evolutivos, redes neuronales y gráficos.

Rafael Alberto Moreno Parra
ramsoftware@gmail.com

Contenido

Tabla de ilustraciones.....	8
Historial de actualizaciones	14
Introducción.....	15
Parte 0: El entorno de desarrollo	16
Parte 1: Variables, si condicional, ciclos y funciones	25
Comentarios en el código.....	26
El tradicional "Hola Mundo"	27
Variables.....	29
Imprimiendo con formato	34
Operaciones matemáticas	37
Diferencias de precisión entre float, double y decimal.....	38
Función de potencia	39
Orden de evaluación de los operadores	40
Problemas al usar cast	41
De formato algebraico a C#	43
Diferencias entre usar el cast y la conversión	44
Conversión de números reales con el punto decimal.....	45
Fallos en la conversión	46
Constantes matemáticas.....	47
Funciones trigonométricas	48
Funciones hiperbólicas	49
Otras funciones matemáticas.....	50
Manejo del NaN (Not a Number) y el infinito	52
Leer un número por consola	53
Si condicional	54
Uso del if...else if... else	55
Uso del AND &&	56
Uso del OR 	57
Uso del switch.....	58
Operadores booleanos.....	59
Precedencia de los operadores	62
El operador "?", un si condicional	63
Captura de error con try...catch.....	64
Uso de TryParse para conversión	65
Ciclo for	66
Ciclo while.....	67
Ciclo do...while	69
Romper ciclo con break	71
Uso del continue en ciclos	72
Ciclos anidados	73
Rompiendo ciclos anidados.....	74
Uso del goto	75
Diferencias de precisión entre float y double	76
Funciones.....	77
Funciones con doble salida	78
Funciones iterativas y recursivas	80
Ámbito de las variables	81
Manejo de cifras	83
Parte 2: Cadenas	89
Declaración de cadenas	90
Uso de @ en cadenas	91
Constantes con cadenas	92

Copia de cadenas	93
Caracteres especiales.....	94
Acceder a un determinado carácter	95
Tamaño de la cadena y recorrerla.....	96
Subcadenas.....	97
Reemplazar caracteres	98
Encontrar subcadenas o caracteres.....	99
Convertir a mayúsculas y minúsculas.....	100
StringBuilder, más veloz y se puede modificar su contenido.....	101
StringBuilder, métricas de velocidad	102
Eliminar caracteres al inicio y al final	103
Comparar cadenas. Forma no recomendada.....	104
Comparar cadenas. Forma recomendada	105
Comparar cadenas. Ignorando las mayúsculas y minúsculas.....	106
Parte 3: Arreglos estáticos.....	107
Declaración y asignar valores	108
Otra forma de definir y asignar.....	109
Arreglo unidimensional de cadenas.....	110
Arreglo unidimensional. Tamaños	111
Recorrer arreglos de cadenas	112
Uso de la instrucción foreach para recorrer un arreglo	113
Ordenar un arreglo de cadenas.....	114
Ordenar un arreglo de cadenas. Las mayúsculas, minúsculas, tildes y diéresis.....	115
Ordenar un arreglo de tipo double.....	116
Funciones genéricas para arreglos unidimensionales	117
Algoritmos de ordenación	118
Métrica de velocidad: Algoritmos de ordenación.....	121
Arreglo bidimensional.....	124
Arreglo tridimensional	125
Arreglo de arreglos	126
Métrica de velocidad: arreglo bidimensional vs arreglo de arreglos	127
Métrica de velocidad: Arreglos multidimensionales	128
Tipo implícito de arreglo	129
Tipo implícito en arreglo de arreglos	131
Convertir una cadena en un arreglo de cadenas al dividirla.....	132
Parte 4: Programación Orientada a Objetos	134
Definir una clase	135
Errores al tratar de acceder a atributos o métodos privados	136
Los atributos deben ser privados. Accediendo a ellos.....	137
Forma reducida de los getters y setters	138
Uso de los getters/setters	139
Otro uso de los getters y setters	141
Forma de inicializar los objetos llamando los setters	142
Dos variables refiriendo al mismo objeto	144
Sobrecarga de métodos.....	146
Por número de parámetros.....	146
Por tipo de parámetros	147
Constructores	148
Constructor sin parámetros	148
Constructor con parámetros	149
Sobrecarga de constructores	150
Usando el constructor para copiar objetos.....	151
Un constructor puede llamar a otros métodos.....	153
Herencia	154
Clases abstractas y herencia.....	155
Nivel de protección en los métodos y atributos	156

Herencia y métodos iguales en clase madre e hija.....	159
Herencia y constructores.....	160
Llamando a métodos de clases madres.....	161
Evitar la herencia.....	162
Clases estáticas	163
Métodos estáticos	164
Constructor static	165
Cuidado con el constructor static.....	167
Interface.....	168
Interface múltiple	170
Herencia e Interface.....	172
Enums.....	173
Cambiando los valores de las constantes en enums.....	174
Ejemplo 1.....	174
Ejemplo 2.....	175
Structs	176
Un struct se puede copiar fácilmente	177
Métodos en un struct	178
Structs y constructores.....	179
Clases parciales	180
Destructores.....	181
Patrones de diseño	183
Factory Method.....	183
Abstract Factory	185
Singleton.....	188
Builder.....	189
Adapter.....	192
Composite	194
Facade	196
Modelo Vista Controlador	198
Parte 5: Estructuras de datos dinámicas	200
El ArrayList.....	201
Adicionar, tamaño, buscar e imprimir	201
Borrar elemento	203
Cambiar Elemento	204
Insertar Elemento.....	205
Referenciar con una variable, un rango de la lista.....	206
Tres formas de recorrer un ArrayList	207
Borrar completamente un ArrayList.....	208
Borrar un rango en un ArrayList	209
Guardar el ArrayList en un arreglo estático	210
Aregar un arreglo estático a un ArrayList	211
Inserta un arreglo estático en una determinada posición del ArrayList	212
Insertar varios ArrayList dentro de un ArrayList	213
Invertir un ArrayList	214
Invertir un rango de datos en el ArrayList	215
Ordena un ArrayList	216
Búsqueda Binaria.....	217
Capacidad del ArrayList	218
Detectar el tipo de dato del elemento del ArrayList.....	219
List	220
Métodos similares a los de ArrayList.....	220
Métricas: Comparativa de desempeño de ArrayList vs List vs Arreglo estático.....	223
Ordenando con tipo char.....	223
Ordenando con tipo int	229
Lista de objetos.....	232

Listas en Listas	234
Dictionary	242
Uso de llaves	242
Llaves tipo string	243
Manejo de objetos en un Dictionary	244
Dato definido en la cola	246
Objetos en la cola	247
Stack (Pila)	249
Dato definido en la pila	250
Objetos en la pila	251
Hashtable.....	253
Manejo de objetos en un Hashtable	254
SortedList	256
LinkedList	257
Objetos en LinkedList	259
Parte 6: LINQ	261
Filtrar de un arreglo	262
Filtrar y poner en un List	263
Ordenación.....	264
Ordenación descendente.....	265
Consulta con salida a texto personalizado.....	266
Contar los registros	267
Máximo, mínimo y suma.....	268
Máximo, mínimo y suma con condiciones	269
Consulta con elementos tipo string	270
Consulta con objetos.....	271
Consulta con objetos y resultado en un List	272
Determinación de tipo de dato	273
Ordenación por un campo y luego por otro	274
Agrupación por un campo	275
Hacer un "join" entre listas	277
Un "join" con resultado personalizado	279
Extraer los datos de una lista que no están en otra	281
Intersección de dos listas.....	282
Unir dos listas sin repetir elementos	283
Consulta de texto por algún patrón	284
Ordenar internamente una cadena	285
Ordenar internamente una cadena con diversos caracteres alfanuméricos.....	286
Ordenamiento según tamaño de la palabra	287
Ordenamiento por la segunda letra de cada palabra	288
Invertir el ordenamiento	289
Parte 7. Un evaluador de expresiones algebraicas.....	290
El problema	291
El algoritmo	291
Paso 1: Convertirla a minúsculas y retirar cualquier carácter que no sea de una expresión algebraica.....	291
Paso 2: Verificando la sintaxis de la expresión algebraica	291
Paso 3: Transformando la expresión	291
Paso 4: Dividiendo la cadena en partes	292
Paso 5: Generando las Piezas desde las Partes	293
Paso 6: Evaluando a partir de las Piezas	293
Como usar el evaluador de expresiones.....	300
Con números reales	301
Con paréntesis	302
Con funciones	303
Con variables	304
Haciendo múltiples cálculos	305

Validando la sintaxis.....	306
Métricas: Comparativa de desempeño con el evaluador interno de C#	309
Parte 8. Estructuras de datos de bajo nivel	312
Lista simplemente enlazada	313
Enlazamiento continuo.....	316
Recorriendo lista simplemente enlazada	317
Recorriendo la lista usando un método	319
Tamaño de la lista.....	321
Traer un determinado nodo	322
Adicionar un nodo en determinada posición.....	323
Borrar un nodo de una determinada posición.....	326
La lista doblemente enlazada	329
Adicionar un nodo en determinada posición.....	331
Borrar un nodo de una determinada posición	335
Árbol binario.....	339
Primer ejemplo.....	339
Segundo ejemplo.....	341
Recorrido iterativo (no recursivo)	343
Generar árboles binarios al azar	345
Ordenamiento usando un árbol binario	347
Buscar en árbol binario ordenado, número de nodos y altura del árbol.....	349
Dibujar un árbol binario	351
Recorrer un árbol binario por niveles.....	353
Árbol N-ario.....	355
Recorriéndolo.....	357
Grafos	359
Generando un grafo aleatoriamente.....	360
Parte 9: Simulaciones	362
Números aleatorios.....	363
Generadores de números aleatorios.....	365
Generador congruencial lineal	365
Blum Blum Shub.....	366
Pruebas a generadores de números aleatorios.....	367
Variables aleatorias.....	371
Distribuciones	372
Distribución Normal.....	372
Distribución Triangular.....	373
Distribución Uniforme	375
Problema del viajero	376
Resolviendo un sudoku	379
El problema de las tres puertas.....	381
Área bajo la curva	383
Dos robots se encuentran	387
Parte 10: Algoritmos evolutivos	389
El problema	390
El operador Mutación	392
El operador Cruce.....	396
Combinando los operadores cruce y mutación	399
Consideraciones sobre los operadores	401
Buscar el máximo en una función.....	402
De genotipos y fenotipos	405
Genotipo: Operador mutación.....	407
Genotipo: Operador cruce	409
Genotipo: Operador cruce y mutación	411
Máximos y mínimos locales	413
Variando el algoritmo evolutivo.....	414

Tamaño de la población	414
Selección de individuos.....	414
Representación de los individuos	414
Paralelizar el algoritmo	414
Buscar el mayor valor en una ecuación de múltiples variables.....	415
Parte 11: Redes Neuronales	418
Iniciando.....	419
Perceptrón simple.....	421
Fórmula de Frank Rosenblatt.....	426
Perceptrón simple: Aprendiendo la tabla del OR.....	428
Límites del Perceptrón Simple.....	429
Encontrando el mínimo en una ecuación.....	431
Descenso del gradiente	434
Mínimos locales y globales	436
Búsqueda de mínimos y redes neuronales	437
Perceptrón Multicapa.....	438
Las neuronas	439
Pesos y como nombrarlos	441
La función de activación de la neurona.....	443
Introducción al algoritmo “Backpropagation” (backward propagation of errors).....	447
Nombrando las entradas, pesos, umbrales, salidas y capas	448
Regla de la cadena	451
Derivadas parciales.....	452
Las derivadas en el algoritmo de propagación hacia atrás.....	453
Tratamiento del error en el algoritmo de propagación hacia atrás	467
Variando los pesos y umbrales con el algoritmo de propagación hacia atrás	473
Implementación en C# del perceptrón multicapa	474
Algoritmo de retro propagación en C#	482
Código completo del perceptrón en C#.....	489
Reconocimiento de números de un reloj digital.....	493
Detección de patrones en series de tiempo	501
Parte 12: Gráficos en C#	509
Línea.....	516
Arco	517
Rectángulo	518
Rectángulo relleno.....	519
Curva Bézier.....	520
Varias líneas rectas.....	521
Polígono.....	522
Polígono relleno.....	523
Curva cerrada	524
Curva cerrada rellena.....	525
Curva que une varios puntos	526
Elipse	527
Elipse rellena	528
Letras.....	529
Diagrama de pastel.....	530
Diagrama de pastel relleno	531
Líneas horizontales	532
Líneas verticales.....	533
Líneas más juntas en el centro	534
Líneas generan rombo.....	535
Líneas en esquina superior derecha	536
Líneas en esquina inferior derecha	537
Líneas en esquina inferior izquierda	538
Triángulos apuntan a la derecha.....	539

Triángulos apuntan a la izquierda	541
Triángulos apuntan a la izquierda alternando colores	543
Rectángulos concéntricos.....	545
Dibuja formas que semejan cuadrados a las que les falta el techo o la base	546
Dibuja ángulos rectos.....	547
Líneas diagonales cruzadas	548
Celdas	549
Líneas rectas simulan curvas	550
Algoritmo de Bresenham para dibujar líneas	551
Traslado de figuras en un plano	552
Giro de figuras en un plano	554
Giro de figuras en un plano calculando el centroide	556
Gráfico matemático en 2D	558
Gráfico polar.....	561
Bibliografía.....	564

Tabla de ilustraciones

Ilustración 1: Sitio oficial de Visual Studio 2022	17
Ilustración 2: Ediciones de Visual Studio 2022	18
Ilustración 3: Condiciones de uso de Visual Studio 2022	19
Ilustración 4: Pantalla de inicio de Visual Studio 2022. Se presiona "Crear un proyecto"	20
Ilustración 5: Se selecciona "Aplicación de consola"	21
Ilustración 6: Se le pone un nombre al proyecto.....	22
Ilustración 7: Se utiliza el .NET 8 y se marca la opción "No usar instrucciones de nivel superior".....	23
Ilustración 8: Se crea una aplicación por defecto. Se da clic en el botón de ejecutar.....	24
Ilustración 9: Ejecución del programa.....	24
Ilustración 10: Ejecución del programa "Hola Mundo". Impresión en consola.....	27
Ilustración 11: Diferencia entre Write y Writeln	28
Ilustración 12: Variables de tipo entero.....	29
Ilustración 13: Valores de tipo "double"	30
Ilustración 14: Valores de tipo "char"	31
Ilustración 15: Valores de tipo "string".....	32
Ilustración 16: Valores almacenados por variables de tipo "float"	33
Ilustración 17: Números impresos con determinados formatos	34
Ilustración 18: Impresión de números de tipo double con formato.....	35
Ilustración 19: Valores tipo "double" con formato	35
Ilustración 20: Valores tipo "double" con formato de redondeo	36
Ilustración 21: Resultados de operaciones.....	37
Ilustración 22: Diferencias entre float, double y decimal	38
Ilustración 23: Función de potencia	39
Ilustración 24: Orden de evaluación de los operadores	40
Ilustración 25: Problemas del uso del "cast"	41
Ilustración 26: Problemas del uso del "cast"	42
Ilustración 27: Formato algebraico a C#	43
Ilustración 28: Diferencias entre usar el cast y la conversión	44
Ilustración 29: Conversión de números reales con el punto decimal.....	45
Ilustración 30: Constantes matemáticas	47
Ilustración 31: Funciones trigonométricas	48
Ilustración 32: Funciones hiperbólicas	49
Ilustración 33: Otras funciones matemáticas	51
Ilustración 34: Manejo del NaN (Not a Number) y el infinito	52
Ilustración 35: Leer un número por consola.....	53
Ilustración 36: Si condicional	54
Ilustración 37: Uso de if...else	55
Ilustración 38: Uso de if... else if ... else	55
Ilustración 39: Uso del AND &&	56
Ilustración 40: Uso del OR 	57
Ilustración 41: Uso del switch	58
Ilustración 42: Operadores booleanos.....	59
Ilustración 43: Tablas de verdad	60
Ilustración 44: Tablas de verdad usando && y 	61
Ilustración 45: Expresión booleana	61
Ilustración 46: Precedencia de los operadores.....	62
Ilustración 47: El operador "?", un si condicional	63
Ilustración 48: Captura de error con try...catch.....	64
Ilustración 49: Uso de TryParse para conversión.....	65
Ilustración 50: Ciclo for.....	66
Ilustración 51: Ciclo while.....	68
Ilustración 52: Ciclo do...while	70
Ilustración 53: Romper ciclo con break	71
Ilustración 54: Uso del continue en ciclos	72
Ilustración 55: Ciclos anidados	73
Ilustración 56: Rompiendo ciclos anidados	74
Ilustración 57: Uso del goto	75
Ilustración 58: Diferencias de precisión entre float y double	76
Ilustración 60: Funciones	77
Ilustración 60: Función que retorna dato de tipo bool.....	77
Ilustración 61: Funciones con doble salida.....	78
Ilustración 62: Funciones con doble salida.....	79
Ilustración 63: Funciones iterativas y recursivas.....	80
Ilustración 64: Ámbito de las variables	81
Ilustración 65: Ámbito de las variables	82
Ilustración 66: Manejo de cifras.....	88
Ilustración 67: Declaración de cadenas	90
Ilustración 68: Uso de @	91
Ilustración 69: No se puede cambiar una constante	92
Ilustración 70: Copia de cadenas	93

Ilustración 71: Caracteres especiales.....	94
Ilustración 72: Acceder a un determinado carácter	95
Ilustración 73: Tamaño de la cadena y recorrerla.....	96
Ilustración 74: Subcadenas.....	97
Ilustración 75: Reemplazar caracteres	98
Ilustración 76: Encontrar subcadenas o caracteres	99
Ilustración 77: Convertir a mayúsculas y minúsculas.....	100
Ilustración 78: StringBuilder, más veloz y se puede modificar su contenido	101
Ilustración 79: StringBuilder, métricas de velocidad	102
Ilustración 80: Eliminar caracteres al inicio y al final	103
Ilustración 81: Comparar cadenas. Forma no recomendada.....	104
Ilustración 82: Comparar cadenas. Forma recomendada	105
Ilustración 83: Comparar cadenas. Ignorando las mayúsculas y minúsculas.....	106
Ilustración 84: Declaración y asignar valores	108
Ilustración 85: Otra forma de definir y asignar.....	109
Ilustración 86: Arreglo unidimensional de cadenas.....	110
Ilustración 87: Arreglo unidimensional. Tamaños	111
Ilustración 88: Recorrer arreglos de cadenas	112
Ilustración 89: Uso de la instrucción foreach para recorrer un arreglo	113
Ilustración 90: Ordenar un arreglo de cadenas.....	114
Ilustración 91: Ordenar un arreglo de cadenas. Las mayúsculas, minúsculas, tildes y diéresis.....	115
Ilustración 92: Ordenar un arreglo de tipo double.....	116
Ilustración 93: Funciones genéricas para arreglos unidimensionales	117
Ilustración 94: Algoritmos de ordenación	120
Ilustración 95: Métrica de velocidad: Algoritmos de ordenación	123
Ilustración 96: Arreglo bidimensional.....	124
Ilustración 97: Arreglo tridimensional	125
Ilustración 98: Arreglo de arreglos	126
Ilustración 99: Métrica de velocidad: arreglo bidimensional vs arreglo de arreglos	127
Ilustración 100: Métrica de velocidad: arreglos multidimensionales	129
Ilustración 101: Métrica de velocidad: arreglos multidimensionales. Límite = 30.....	129
Ilustración 102: Tipo implícito de arreglo	130
Ilustración 103: Tipo implícito en arreglo de arreglos	131
Ilustración 104: Convertir una cadena en un arreglo de cadenas al dividirla	132
Ilustración 105: Cuando hay más de un espacio intermedio	133
Ilustración 106: Atributos/Métodos privados y públicos.....	135
Ilustración 107: Intento de acceder a un atributo privado	136
Ilustración 108: Los atributos deben ser privados. Accediendo a ellos.	137
Ilustración 109: Forma reducida de los getters y setters	138
Ilustración 110: Uso de los getters/setters	140
Ilustración 111: Otro uso de los getters y setters	141
Ilustración 112: Forma de inicializar los objetos llamando los setters	143
Ilustración 113: Ambas variables apuntan al mismo objeto instanciado	144
Ilustración 114: Dos variables refiriendo al mismo objeto	145
Ilustración 115: Sobrecarga de métodos	146
Ilustración 116: Sobrecarga de métodos	147
Ilustración 117: Constructor sin parámetros	148
Ilustración 118: Constructor con parámetros	149
Ilustración 119: Sobrecarga de constructores	150
Ilustración 120: Usando el constructor para copiar objetos	152
Ilustración 121: Un constructor puede llamar a otros métodos.....	153
Ilustración 122: Los atributos o métodos con el atributo private no pueden ser usados por las clases hijas	156
Ilustración 123: Atributos "protected" pueden ser accedidos por las clases hijas, pero no pueden ser accedidos por instancias.....	157
Ilustración 124: Herencia y métodos iguales en clase madre e hija.....	159
Ilustración 125: Constructores y herencia	160
Ilustración 126: Llamando a métodos de clases madres	161
Ilustración 127: Evitar la herencia.....	162
Ilustración 128: Clases estáticas.....	163
Ilustración 129: Métodos estáticos	164
Ilustración 130: Constructor static	166
Ilustración 131: Cuidado con el constructor static.....	167
Ilustración 132: Interface	169
Ilustración 133: Interface múltiple	171
Ilustración 134: Herencia e Interface.....	172
Ilustración 135: Enums	173
Ilustración 136: Cambiando los valores de las constantes en enums	174
Ilustración 137: Cambiando los valores de las constantes en enums	175
Ilustración 138: Structs.....	176
Ilustración 139: Un struct se puede copiar fácilmente	177
Ilustración 140: Métodos en un struct.....	178
Ilustración 141: Structs y constructores	179
Ilustración 142: Clases parciales.....	180
Ilustración 143: Destructores	182
Ilustración 144: Factory Method	184
Ilustración 145: Abstract Factory	187

Ilustración 146: Singleton	188
Ilustración 147: Builder.....	191
Ilustración 148: Adapter.....	193
Ilustración 149: Composite.....	195
Ilustración 150: Facade	197
Ilustración 151: Modelo Vista Controlador	199
Ilustración 152: ArrayList Adicionar, tamaño, buscar e imprimir	201
Ilustración 153: ArrayList, borrar elemento	203
Ilustración 154: ArrayList, cambiar elemento	204
Ilustración 155: ArrayList, insertar elemento	205
Ilustración 156: ArrayList, referenciar con una variable, un rango de la lista	206
Ilustración 157: ArrayList, tres formas de recorrerlo	207
Ilustración 158: Borrar completamente un ArrayList	208
Ilustración 159: Borrar un rango en un ArrayList	209
Ilustración 160: Guardar el ArrayList en un arreglo estático	210
Ilustración 161: Agregar un arreglo estático a un ArrayList	211
Ilustración 162: Inserta un arreglo estático en una determinada posición del ArrayList	212
Ilustración 163: Insertar varios ArrayList dentro de un ArrayList	213
Ilustración 164: Invertir un ArrayList.....	214
Ilustración 165: Invertir un rango de datos en el ArrayList	215
Ilustración 166: Ordena un ArrayList	216
Ilustración 167: Búsqueda Binaria.....	217
Ilustración 168: Capacidad del ArrayList	218
Ilustración 169: Detectar el tipo de dato del elemento del ArrayList.....	219
Ilustración 170: List, métodos	222
Ilustración 171: List, métodos	222
Ilustración 172: Métrica: Ordenando con tipo char.....	225
Ilustración 173: Métrica: Ordenando con tipo double	228
Ilustración 174: Métrica: Ordenando con tipo int	231
Ilustración 175: Lista de objetos.....	233
Ilustración 176: Uso de listas para simular un sistema de información	239
Ilustración 177: Uso de listas para simular un sistema de información	239
Ilustración 178: Uso de listas para simular un sistema de información	239
Ilustración 179: Uso de listas para simular un sistema de información	240
Ilustración 180: Uso de listas para simular un sistema de información	240
Ilustración 181: Uso de listas para simular un sistema de información	240
Ilustración 182: Uso de listas para simular un sistema de información	241
Ilustración 183: Uso de listas para simular un sistema de información	241
Ilustración 184: Uso de listas para simular un sistema de información	241
Ilustración 185: Dictionary	242
Ilustración 186: Dictionary	243
Ilustración 187: Dictionary, manejo de objetos	244
Ilustración 188: Queue	245
Ilustración 189: Dato definido en la cola	246
Ilustración 190: Objetos en la cola	248
Ilustración 191: Stack.....	249
Ilustración 192: Dato definido en la pila	250
Ilustración 193: Objetos en la pila	252
Ilustración 194: Hashtable	253
Ilustración 195: Manejo de objetos en un Hashtable	255
Ilustración 196: SortedList	256
Ilustración 197: LinkedList.....	258
Ilustración 198: Objetos en LinkedList	260
Ilustración 199: LINQ: Filtrar de un arreglo	262
Ilustración 200: LINQ: Filtrar y poner en un List.....	263
Ilustración 201: LINQ: Ordenación	264
Ilustración 202: LINQ: Ordenación descendente	265
Ilustración 203: LINQ: Consulta con salida a texto personalizado	266
Ilustración 204: LINQ: Contar los registros.....	267
Ilustración 205: LINQ: Máximo, mínimo y suma	268
Ilustración 206: LINQ: Máximo, mínimo y suma con condiciones	269
Ilustración 207: LINQ: Consulta con elementos tipo string	270
Ilustración 208: LINQ: Consulta con objetos	271
Ilustración 209: LINQ: Consulta con objetos y resultado en un List	272
Ilustración 210: LINQ: Determinación de tipo de dato	273
Ilustración 211: LINQ: Ordenación por un campo y luego por otro	274
Ilustración 212: LINQ: Agrupación por un campo	276
Ilustración 213: LINQ: Hacer un "join" entre listas.....	278
Ilustración 214: LINQ: Un "join" con resultado personalizado	280
Ilustración 215: LINQ: Extraer los datos de una lista que no están en otra	281
Ilustración 216: LINQ: Intersección de dos listas.....	282
Ilustración 217: LINQ: Unir dos listas sin repetir elementos	283
Ilustración 218: LINQ: Consulta de texto por algún patrón.....	284
Ilustración 219: LINQ: Ordenar internamente una cadena	285
Ilustración 220: LINQ: Ordenar internamente una cadena con diversos caracteres alfanuméricos	286

Ilustración 221: LINQ: Ordenamiento según tamaño de la palabra	287
Ilustración 222: LINQ: Ordenamiento por la segunda letra de cada palabra	288
Ilustración 223: LINQ: Invertir el ordenamiento	289
Ilustración 224: Uso del evaluador de expresiones.....	300
Ilustración 225: Resultado obtenido con el evaluador	300
Ilustración 226: Evaluador 4 operando números reales.....	301
Ilustración 227: Evaluador 4 operando con paréntesis.....	302
Ilustración 228: Evaluador 4 operando con funciones	303
Ilustración 229: Evaluador 4 operando con uso de variables.....	304
Ilustración 230: Evaluador 4 usado para generar múltiples valores de una ecuación al cambiar los valores.	305
Ilustración 231: Evaluando la sintaxis.....	307
Ilustración 232: Evaluando la sintaxis.....	308
Ilustración 233: Evaluando la sintaxis.....	308
Ilustración 234: Métrica compara ambos evaluadores	311
Ilustración 235: Métrica compara ambos evaluadores	311
Ilustración 236: Métrica compara ambos evaluadores	311
Ilustración 224: Representación gráfica de una lista simplemente enlazada	313
Ilustración 225: El nodo es un objeto con sus atributos, métodos y un apuntador.....	313
Ilustración 239: Lista simplemente enlazada.....	314
Ilustración 226: Se crean tres nodos	314
Ilustración 227: Conectar el nodo primero con el nodo segundo.....	315
Ilustración 242: Enlazamiento continuo	316
Ilustración 228: Variable "lista" sostiene la lista simplemente enlazada	317
Ilustración 244: Recorriendo lista simplemente enlazada	318
Ilustración 229: La variable "pasea" va de nodo en nodo. Con esta variable se muestra el contenido de cada nodo.	318
Ilustración 246: Recorriendo la lista usando un método	320
Ilustración 247: Tamaño de la lista.....	321
Ilustración 248: Traer un determinado nodo	322
Ilustración 249: Adicionar un nodo en determinada posición	325
Ilustración 250: Borrar un nodo de una determinada posición.....	328
Ilustración 251: La lista doblemente enlazada.....	330
Ilustración 252: Adicionar un nodo en determinada posición	334
Ilustración 253: Borrar un nodo de una determinada posición.....	337
Ilustración 230: Ejemplo de un árbol binario.....	339
Ilustración 255: Recorrido de un árbol binario.....	339
Ilustración 231: Ejemplo de árbol binario	341
Ilustración 257: Recorrido de un árbol binario.....	341
Ilustración 232: Árbol binario	343
Ilustración 259: Recorrido iterativo (no recursivo)	343
Ilustración 260: Generar árboles binarios al azar.....	346
Ilustración 261: Ordenamiento usando un árbol binario	348
Ilustración 262: Buscar en árbol binario ordenado, número de nodos y altura del árbol	350
Ilustración 263: Dibujar un árbol binario	351
Ilustración 264: Dibujar un árbol binario	352
Ilustración 233: Árbol binario de ejemplo	354
Ilustración 266: Recorrer un árbol binario por niveles.....	354
Ilustración 267: Árbol N-ario	356
Ilustración 268: Recorriendo árbol N-ario	358
Ilustración 234: Grafo generado	359
Ilustración 270: Grafos	359
Ilustración 271: Grafo generado	361
Ilustración 272: Grafo dibujado con https://viz-js.com/	361
Ilustración 273: Generando números aleatorios	363
Ilustración 274: Números aleatorios	365
Ilustración 275: Prueba del generador de números aleatorios	369
Ilustración 276: Prueba del generador de números aleatorios	370
Ilustración 277: Distribución Normal	372
Ilustración 278: Distribución Normal	372
Ilustración 279: Distribución Triangular.....	373
Ilustración 280: Distribución Triangular.....	374
Ilustración 281: Distribución Uniforme.....	375
Ilustración 282: Resuelve el Sudoku.....	380
Ilustración 283: Problema de las tres puertas	382
Ilustración 284: Problema de las tres puertas	382
Ilustración 285: Curva Y = F(X).....	383
Ilustración 286: Los límites Xmin y Xmax para hallar el área interna	383
Ilustración 287:Se dibuja un rectángulo que cubra la curva.....	384
Ilustración 288: Se lanzan puntos al azar al interior del rectángulo	384
Ilustración 289: Dos robots moviéndose	388
Ilustración 290: Generar cadenas al azar	390
Ilustración 291: Algoritmo evolutivo con el operador mutación	395
Ilustración 292: Algoritmo evolutivo con el operador cruce.....	398
Ilustración 293: Algoritmo evolutivo: Combinando operador cruce y mutación.....	400
Ilustración 294:Polinomio de grado 6.....	402
Ilustración 295: El mayor valor de Y en un rango dado	404

Ilustración 296: Gráfico de una ecuación.....	405
Ilustración 297: Búsqueda del mayor valor usando genotipo de cadenas de bits	408
Ilustración 298: Genotipo: Operador cruce	410
Ilustración 299: Operador cruce y mutación	412
Ilustración 300: Gráfico matemático.....	413
Ilustración 301: Buscar el mayor valor de Y con múltiples variables independientes	417
Ilustración 302: Caja negra, entradas y salidas	419
Ilustración 303: Pesos al interior de la caja	419
Ilustración 304: Esos pesos ya operan con el ejemplo A	419
Ilustración 305: Los mismos pesos funcionan para el ejemplo B.....	420
Ilustración 306: Y esos mismos pesos funcionan para el ejemplo C	420
Ilustración 307: Perceptrón simple	421
Ilustración 308: Funcionamiento del perceptrón simple	421
Ilustración 309: Los pesos funcionan para esa regla.....	422
Ilustración 310: Esos pesos fallan con la segunda regla	423
Ilustración 311: Dar con los pesos con sólo azar	424
Ilustración 312: Dar con los pesos con sólo azar	424
Ilustración 313: Dar con los pesos con sólo azar	424
Ilustración 314: Encontrando los pesos más rápido	427
Ilustración 315: Encontrando los pesos más rápido	427
Ilustración 316: Encontrando los pesos más rápido.....	427
Ilustración 317: Tabla del AND	429
Ilustración 318: Tabla del XOR	429
Ilustración 319: Red neuronal con 3 neuronas	430
Ilustración 320: Red neuronal con otro tipo de conexiones	430
Ilustración 321: Tabla y gráfico de la ecuación hechos con Microsoft Excel.....	431
Ilustración 322: Derivada usando en WolframAlpha	431
Ilustración 323: Buscando el mínimo	433
Ilustración 324: Pendientes	434
Ilustración 325: Gráfico de un polinomio de quinto grado.....	436
Ilustración 326: Red neuronal con varias conexiones distintas	437
Ilustración 327: Perceptrón multicapa	438
Ilustración 328: Esquema de una neurona.....	439
Ilustración 329: Esquema de un perceptrón multicapa	441
Ilustración 330: Nombrando los pesos con una letra	442
Ilustración 331: Gráfico de una función sigmoide.....	443
Ilustración 332: Derivada de la función sigmoide.....	444
Ilustración 333: Comparativa de derivadas	445
Ilustración 334: Esquema de un perceptrón multicapa	447
Ilustración 335: Partes de un perceptrón multicapa	448
Ilustración 336: Derivada parcial	452
Ilustración 337: Nombramiento de pesos, umbrales y salidas	453
Ilustración 338: Esquema de un perceptrón multicapa	461
Ilustración 339: Primer camino para ese peso	461
Ilustración 340: Segundo camino para ese peso	462
Ilustración 341: Primer camino	463
Ilustración 342: Segundo camino	464
Ilustración 343: Tercer camino	464
Ilustración 344: Cuarto camino	465
Ilustración 345: Dos entradas y dos salidas	467
Ilustración 346: Representación del error	468
Ilustración 347: Modelo del perceptrón	474
Ilustración 348: Modelo del perceptrón	474
Ilustración 349: Un perceptrón multicapa	481
Ilustración 350: Perceptrón aprendiendo la tabla del XOR	492
Ilustración 351: La red neuronal "va aprendiendo" los patrones para identificar el número digital.....	500
Ilustración 352: La red neuronal "aprendió" los patrones para identificar el número digital.....	500
Ilustración 353: Gráfico generado por los puntos	501
Ilustración 354: Gráfico uniendo los puntos	502
Ilustración 355: Gráfico al normalizar los datos	503
Ilustración 356: Red neuronal adaptándose a una serie temporal	508
Ilustración 357: Red neuronal adaptándose a una serie temporal	508
Ilustración 358: Inicia proyecto en Visual Studio 2022	510
Ilustración 359: Selecciona "Aplicación de Windows Forms"	511
Ilustración 360: Pone nombre al proyecto, para este libro es "Graficos" (sin tilde).....	512
Ilustración 361: Selecciona .NET 8.0	512
Ilustración 362: Una aplicación de escritorio típica	513
Ilustración 363: Clic botón derecho sobre la ventana y selecciona "Propiedades"	513
Ilustración 364: En la ventana de "Propiedades", selecciona "Eventos"	514
Ilustración 365: En "Eventos" se da doble clic al frente del evento "Paint"	514
Ilustración 366: Esta es el código que se genera automáticamente	515
Ilustración 367: Línea	516
Ilustración 368: Arco	517
Ilustración 369: Rectángulo	518
Ilustración 370: Rectángulo relleno	519

Ilustración 371: Curva Bézier	520
Ilustración 372: Varías líneas rectas	521
Ilustración 373: Polígono	522
Ilustración 374: Polígono relleno	523
Ilustración 375: Curva cerrada	524
Ilustración 376: Curva cerrada rellena	525
Ilustración 377: Curva que une varios puntos	526
Ilustración 378: Elipse	527
Ilustración 379: Elipse rellena	528
Ilustración 380: Letras	529
Ilustración 381: Diagrama de pastel	530
Ilustración 382: Diagrama de pastel relleno	531
Ilustración 383: Líneas horizontales	532
Ilustración 384: Líneas verticales	533
Ilustración 385: Líneas más juntas en el centro	534
Ilustración 386: Líneas generan rombo	535
Ilustración 387: Líneas en esquina superior derecha	536
Ilustración 388: Líneas en esquina inferior derecha	537
Ilustración 389: Líneas en esquina inferior izquierda	538
Ilustración 390: Triángulos apuntan a la derecha	540
Ilustración 391: Triángulos apuntan a la izquierda	542
Ilustración 392: Triángulos apuntan a la izquierda alternando colores	544
Ilustración 393: Rectángulos concéntricos	545
Ilustración 394: Dibuja formas que semejan cuadrados a las que les falta el techo o la base	546
Ilustración 395: Dibuja ángulos rectos	547
Ilustración 396: Líneas diagonales cruzadas	548
Ilustración 397: Celdas	549
Ilustración 398: Líneas rectas simulan curvas	550
Ilustración 399: Algoritmo de Bresenham para dibujar líneas	551
Ilustración 400: Traslado de figuras en un plano	552
Ilustración 401: Traslado de figuras en un plano $mueveX = 70$	552
Ilustración 402: Traslado de figuras en un plano $mueveY = 70$	553
Ilustración 403: Traslado de figuras en un plano $mueveX = 70$ y $mueveY = 70$	553
Ilustración 404: Giro de figuras en un plano	555
Ilustración 405: Giro de figuras en un plano calculando el centroide	557
Ilustración 406: Gráfico matemático en 2D	560
Ilustración 407: Gráfico polar	563

Historial de actualizaciones

Fecha	Qué se hizo
17 de enero de 2024	Lanzamiento del libro con 455 páginas.
04 de febrero de 2024	Agregado el capítulo de redes neuronales. Mejoras en la redacción y presentación de las partes. 564 páginas.

Introducción

¿Por qué C#? Se enumeran algunas razones:

Lenguaje fuertemente tipado [1] por lo que obliga a estar atento en la escritura de operaciones matemáticas y validar que los resultados sean correctos minimizando el riesgo de cálculos incorrectos que pueden aparecer al mezclar datos de diferente tipo.

Los entornos de desarrollo son bastante maduros [2] [3] y completos.

Puede ser usado para hacer aplicaciones de escritorio [4], Web [5] e inclusive videojuegos usando motores como Godot [6] o Unity [7].

Es multiplataforma [8] [9].

Es compilado por lo que su velocidad de ejecución es muy rápida [10] [11] [12].

Tiene protección de memoria [13], luego las operaciones con arreglos son mucho más confiables, ante un error de desbordamiento, C# se detiene y muestra en qué instrucción hubo el problema.

Tiene recolector de basura [14], es decir, no hay que preocuparse por liberar la memoria de los objetos que ya no se usan, C# se encarga automáticamente de esto.

Es libre [15]. C# está regulado por ECMA International [16].

El entorno de desarrollo será en principio Microsoft Visual Studio 2022 Community Edition, en sucesivas actualizaciones, se documentará en los anexos otras herramientas de desarrollo.

Parte 0: El entorno de desarrollo

El entorno de desarrollo que ofrece Microsoft para C# es Visual Studio 2022, el sitio oficial es <https://visualstudio.microsoft.com/es/vs/> :

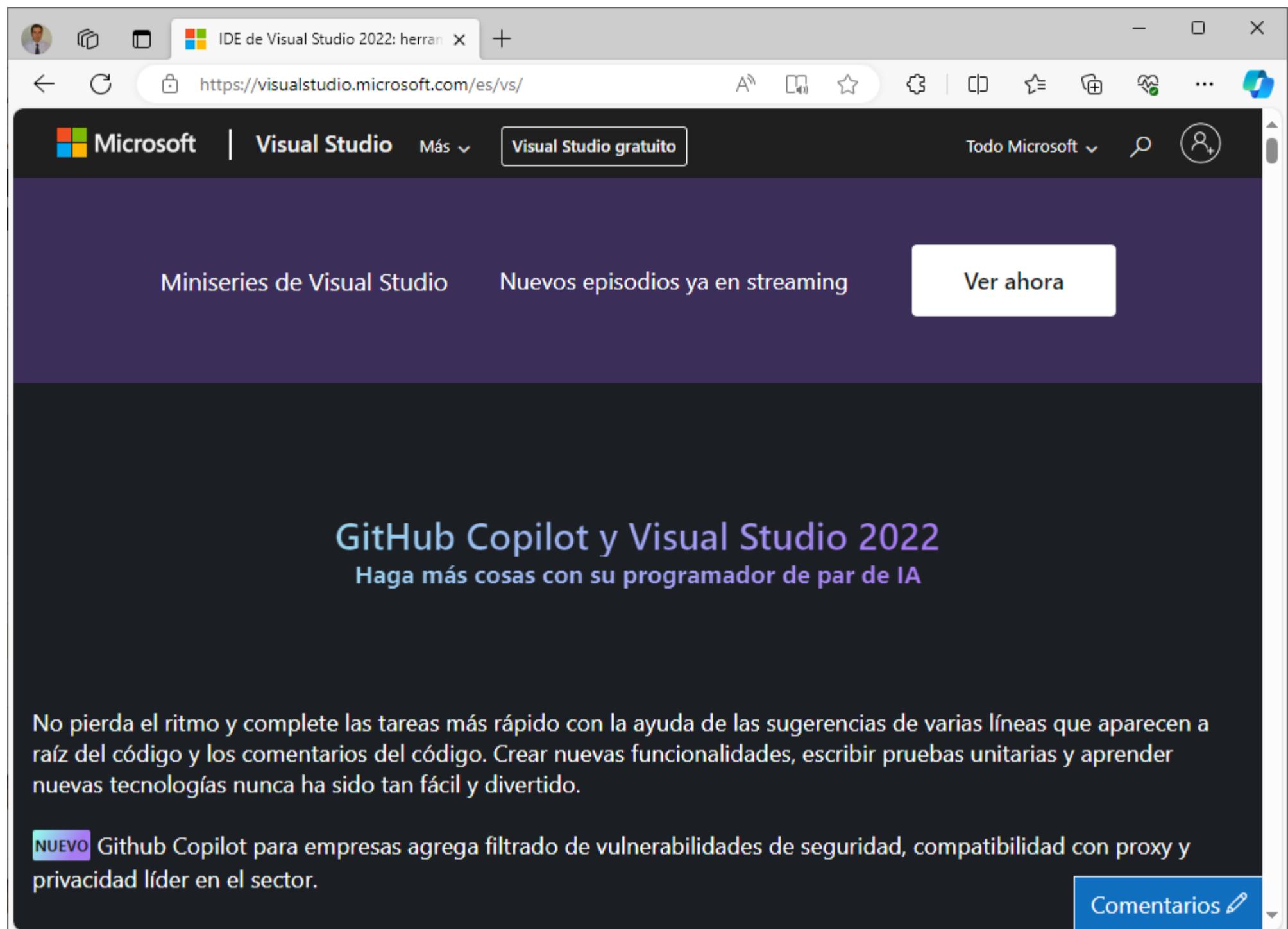


Ilustración 1: Sitio oficial de Visual Studio 2022

Hay tres ediciones de esta herramienta:

The screenshot shows a Microsoft Edge browser window comparing three editions of Visual Studio 2022. The top navigation bar includes the Microsoft logo, 'Visual Studio' menu, a 'Visual Studio gratuito' button, and a search bar. A woman with curly hair is shown in profile on the right side of the page.

Características admitidas	Visual Studio Community	Visual Studio Professional	Visual Studio Enterprise
Escenarios de uso admitidos	● ● ● ○	● ● ● ●	● ● ● ●
Compatibilidad con la plataforma de desarrollo ²	● ● ● ●	● ● ● ●	● ● ● ●
Entorno de desarrollo integrado	● ● ● ○	● ● ● ○	Comentarios

Ilustración 2: Ediciones de Visual Studio 2022

Para iniciar en el desarrollo de C#, se recomienda descargar y usar la edición Visual Studio Community. Al desplegar en "Escenarios de uso admitidos" muestra las condiciones de uso:

Comparar ofertas de productos <https://visualstudio.microsoft.com/es/vs/compare/>

The screenshot shows a comparison chart for Microsoft Visual Studio editions. The columns represent Visual Studio Community, Professional, and Enterprise. The rows list various usage scenarios. Each cell contains a blue dot indicating support, except for the first row which uses a circle icon.

Características admitidas	Visual Studio Community	Visual Studio Professional	Visual Studio Enterprise
(-) Escenarios de uso admitidos	●●●○	●●●●	●●●●
Desarrolladores individuales	✓	✓	✓
Aprendizaje en clase	✓	✓	✓
Investigación académica	✓	✓	✓
Contribución a proyectos de código abierto	✓	✓	✓
Organizaciones no empresariales, ⁵ para un máximo de 5 usuarios	✓		
Enterprise		✓	✓
Compatibilidad con la	●●●●	●●●●	Comentarios

Ilustración 3: Condiciones de uso de Visual Studio 2022

Una vez instalado el entorno de desarrollo, al ejecutar aparece esta pantalla:

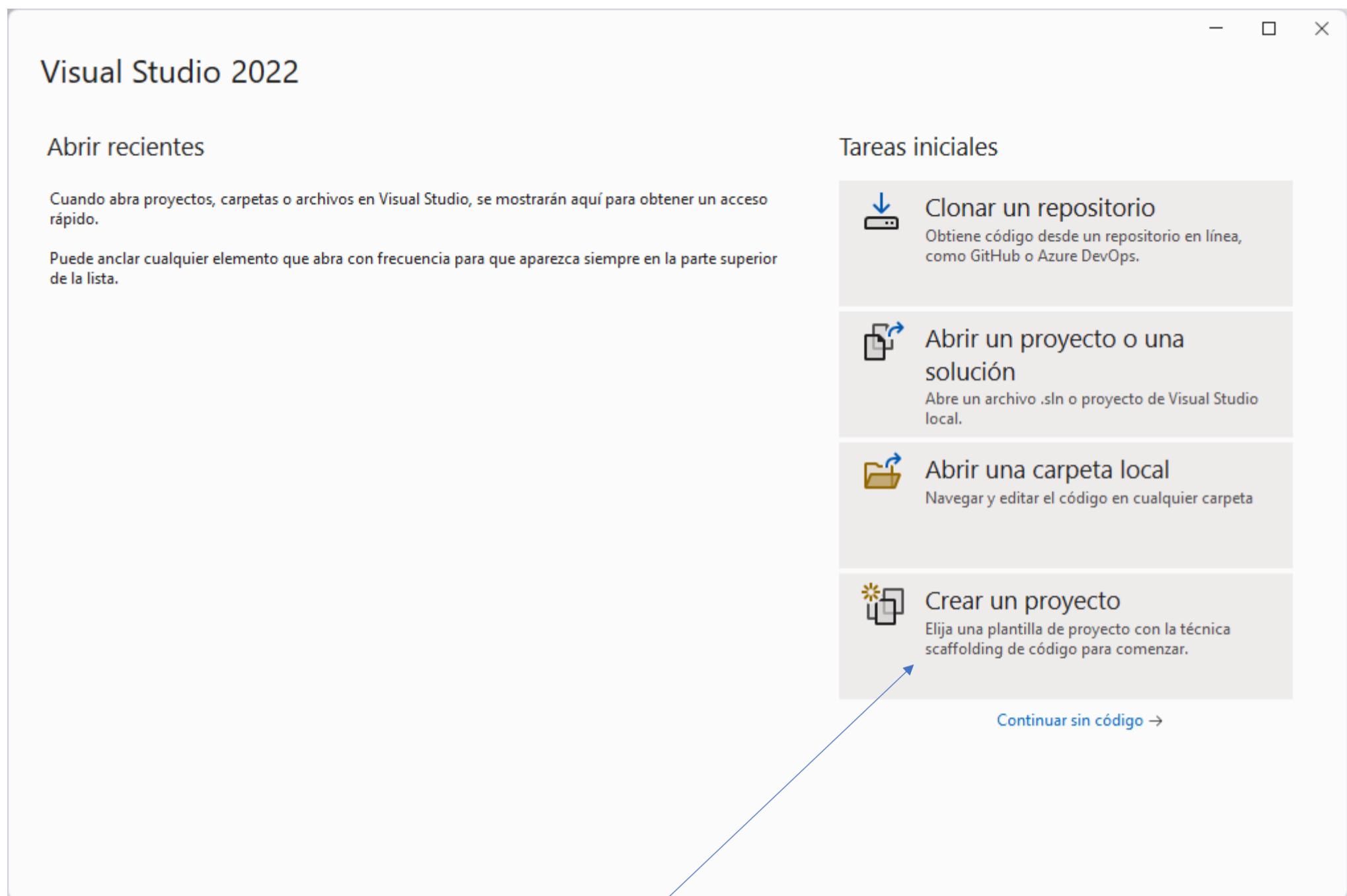


Ilustración 4: Pantalla de inicio de Visual Studio 2022. Se presiona "Crear un proyecto"

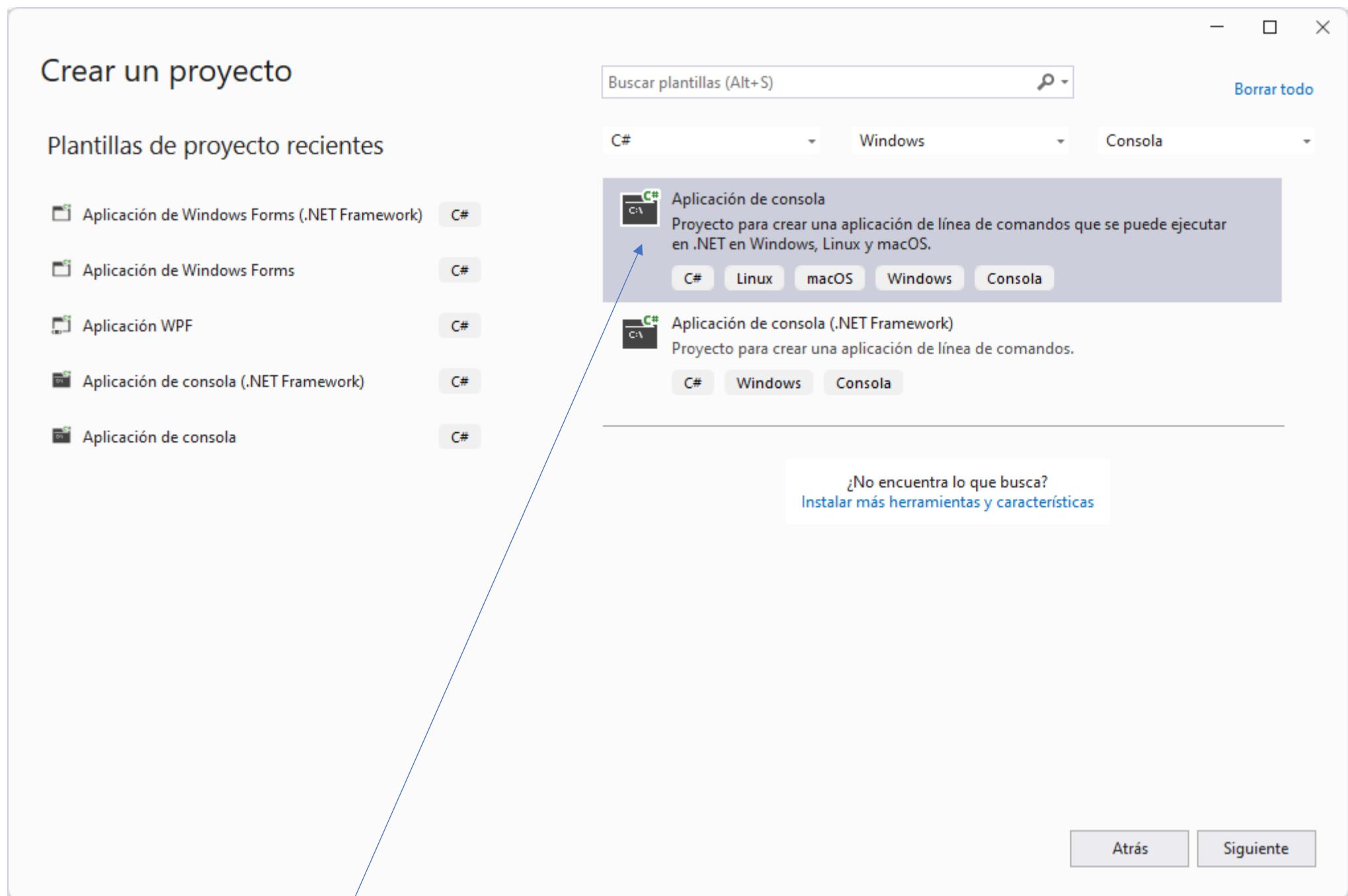


Ilustración 5: Se selecciona "Aplicación de consola"

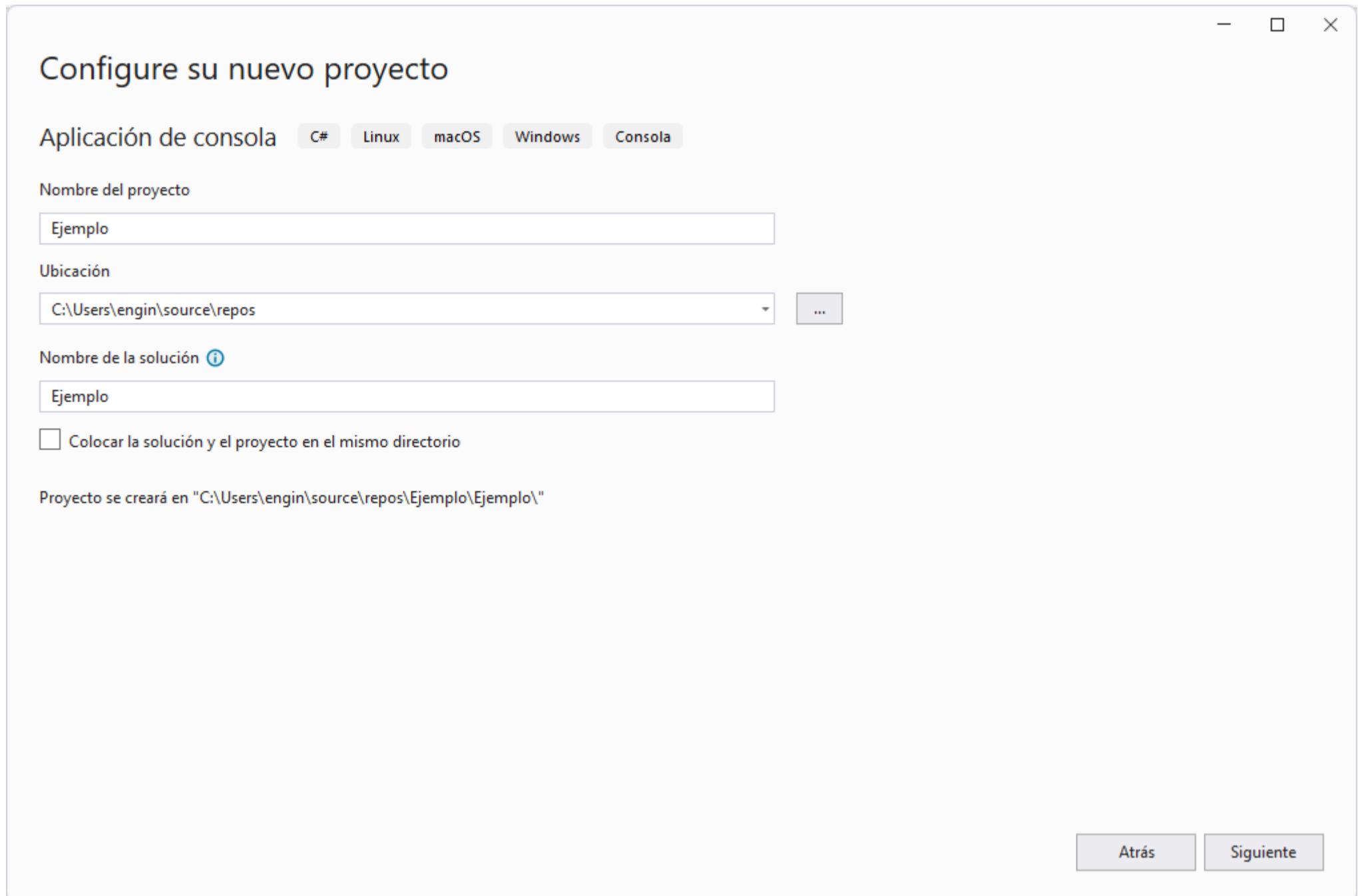


Ilustración 6: Se le pone un nombre al proyecto.

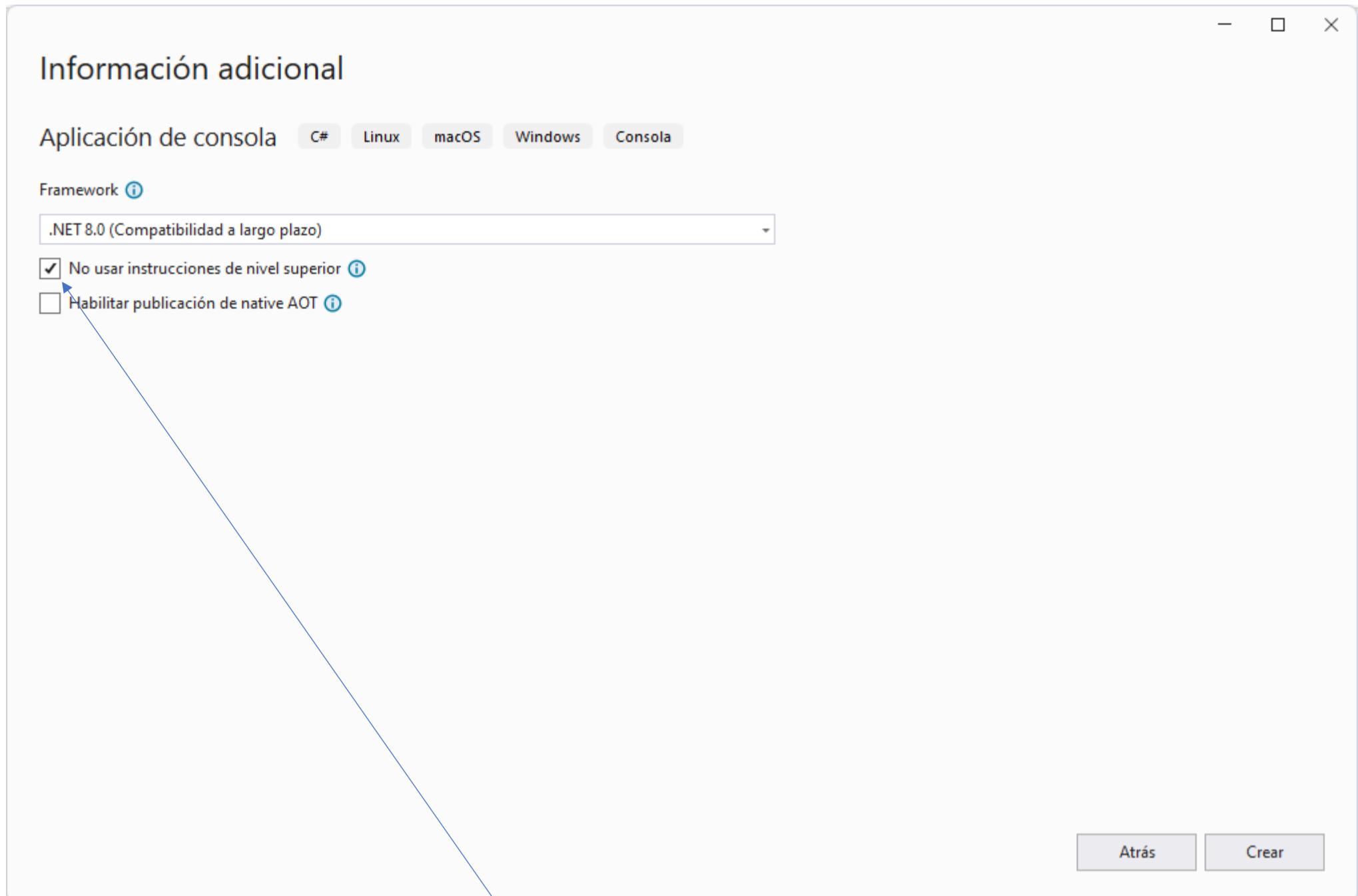


Ilustración 7: Se utiliza el .NET 8 y se marca la opción "No usar instrucciones de nivel superior"

Es recomendado activar el “No usar instrucciones de nivel superior” para que en el código fuente aparezcan todas las instrucciones mínimas requeridas para que compile y ejecute un programa en C#.

En “Habilitar publicación de native AOT”, se explica mejor aquí: <https://dev.to/bytehide/native-aot-the-future-of-net-app-development-47ha> y <https://learn.microsoft.com/en-us/dotnet/core/deploying/native-aot/>

The screenshot shows the Microsoft Visual Studio IDE interface. The top menu bar includes Archivo, Editar, Ver, Git, Proyecto, Compilar, Depurar, Prueba, Analizar, Herramientas, Extensiones, Ventana, Ayuda, Buscar, and Ejemplo. The toolbar below has various icons for file operations like Open, Save, and Print. The status bar at the bottom shows 'Any CPU' and 'Ejemplo'. The main window displays the code for 'Program.cs' in the 'Ejemplo' project. The code is as follows:

```
1  namespace Ejemplo {  
2      internal class Program {  
3          static void Main(string[] args) {  
4              Console.WriteLine("Hello, World!");  
5          }  
6      }  
7  }
```

Ilustración 8: Se crea una aplicación por defecto. Se da clic en el botón de ejecutar

The screenshot shows the 'Consola de depuración de Mi...' window. The title bar says 'Consola de depuración de Mi...'. The window contains the following text:

```
Hello, World!  
C:\Users\engin\source\repos\Ejemplo\Ejemplo\bin\Debug\net8.0\Ejemplo.exe (proceso 17988)  
se cerró con el código 0.  
Para cerrar automáticamente la consola cuando se detiene la depuración, habilite Herramientas ->Opciones ->Depuración ->Cerrar la consola automáticamente al detenerse la depuración.  
Presione cualquier tecla para cerrar esta ventana. . .
```

Ilustración 9: Ejecución del programa

Parte 1: Variables, si condicional, ciclos y funciones

Comentarios en el código

En C#, similar a lenguajes como C, C++ o Java, los comentarios pueden ser de una línea usando los caracteres // o de varias líneas iniciando con /* y finalizando con */

A/001.cs

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Este es un comentario de una sola linea

            /* Este es un
               comentario de
               varias lineas */
        }
    }
}
```

El tradicional "Hola Mundo"

Para mostrar datos en la consola se utiliza la instrucción `Console.WriteLine` o `Console.Write`, la diferencia entre ambas instrucciones es que la primera imprime y hace un salto de renglón, la segunda no hace ese salto de renglón.

A/002.cs

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Imprime en consola
            Console.WriteLine("Hola Mundo");
        }
    }
}
```

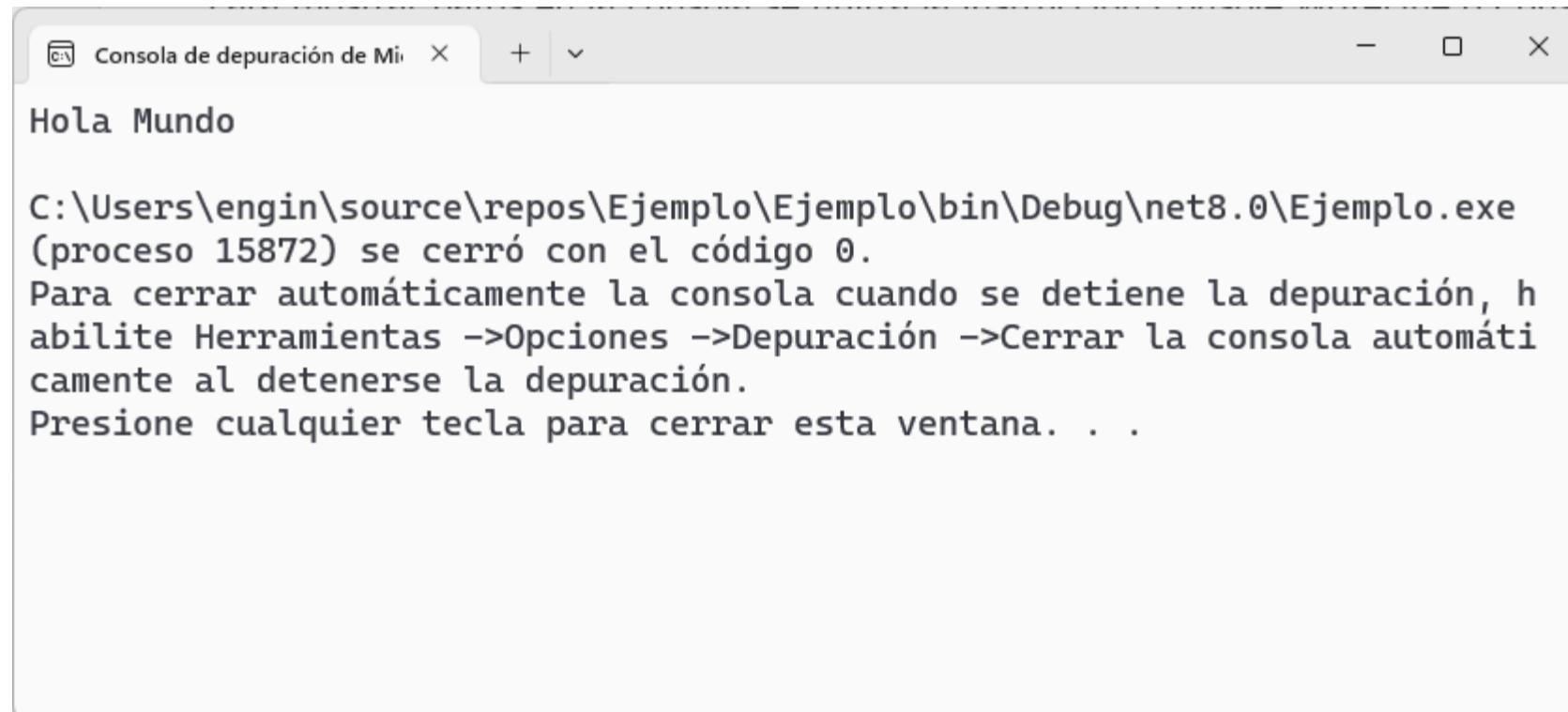


Ilustración 10: Ejecución del programa "Hola Mundo". Impresión en consola.

A/003.cs

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Imprime en consola en dos líneas
            Console.WriteLine("Primera Línea");
            Console.WriteLine("Segunda Línea");

            //Imprime en la misma línea
            Console.Write("Este es un texto");
            Console.Write(" de una sola línea");
        }
    }
}
```

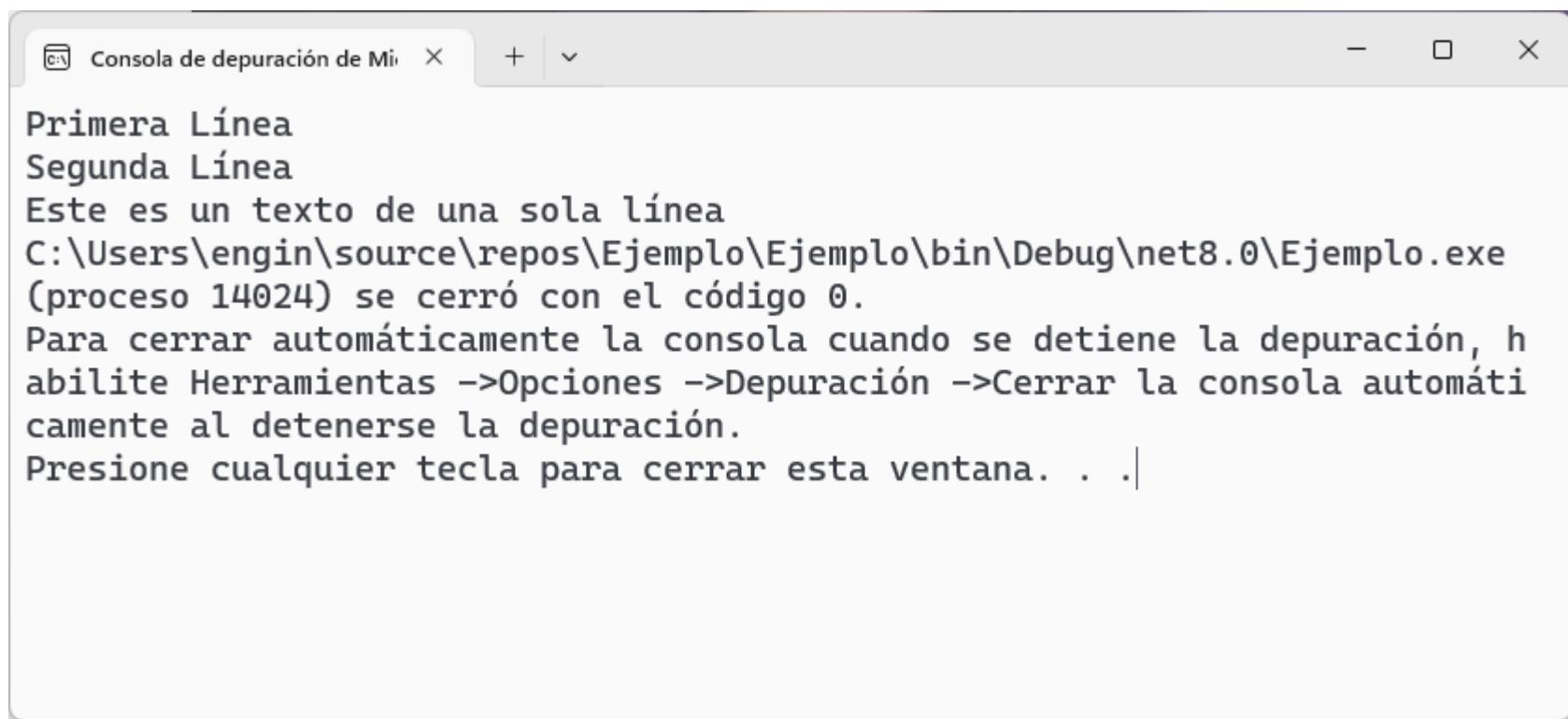


Ilustración 11: Diferencia entre Write y WriteLine

Variables

C# es un lenguaje de programación tipado, luego al crear las variables se debe poner su tipo:

Tipo	Descripción	Valor mínimo	Valor máximo
byte	Número entero sin signo	0	255
sbyte	Número entero con signo	-128	127
short	Número entero con signo	-32768	32767
ushort	Número entero sin signo	0	65535
int	Número entero con signo (32 bits)	-2147483648	2147483647
uint	Número entero sin signo (32 bits)	0	4294967295
long	Número entero con signo (64 bits)	-9223372036854775808	9223372036854775807
ulong	Número entero sin signo (64 bits)	0	18446744073709551615
float	Número real de precisión simple (32 bits)		
double	Número real de precisión doble (64 bits)		
decimal	Número real de precisión más alta (128 bits)		
char	Carácter (1 byte)		
bool	Booleano (1 bit)		
string	Almacenamiento de caracteres alfanuméricos		
var	Es una forma de crear una variable local que no requiere especificar el tipo, ya que el compilador de C# lo deduce a partir de la expresión que se asigna.		

A/004.cs

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Declara variables de tipo entero
            int ladoA = 4;
            int ladoB = 8;
            int ladoC = 10;

            //Imprime esas variables
            Console.WriteLine("Lado A es: " + ladoA.ToString());
            Console.WriteLine("Lado B es: " + ladoB.ToString());
            Console.WriteLine("Lado C es: " + ladoC.ToString());
        }
    }
}
```

The screenshot shows the 'Consola de depuración de Mi...' (Debug Console) window. It displays the output of the program, which consists of three lines of text: 'Lado A es: 4', 'Lado B es: 8', and 'Lado C es: 10'. Below this, there is additional text from the operating system: 'C:\Users\engin\source\repos\Ejemplo\Ejemplo\bin\Debug\net8.0\Ejemplo.exe (proceso 17572) se cerró con el código 0.' followed by instructions for closing the console.

Ilustración 12: Variables de tipo entero

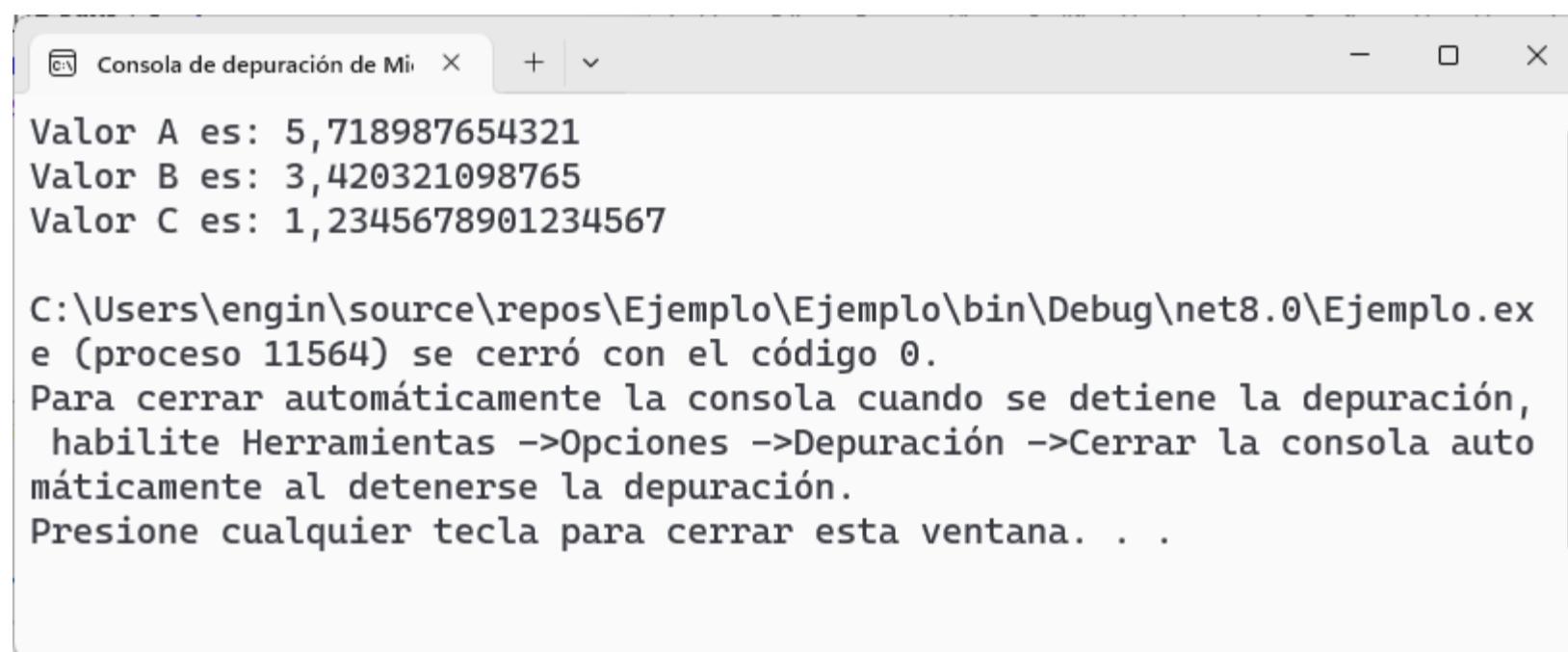


Ilustración 13: Valores de tipo "double"

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Declara variables de tipo char
            char letraA = 'R';
            char letraB = 'a';
            char letraC = 'f';

            //Imprime esas variables
            Console.WriteLine("Letra A es: " + letraA.ToString());
            Console.WriteLine("Letra B es: " + letraB.ToString());
            Console.WriteLine("Letra C es: " + letraC.ToString());
        }
    }
}
```

```
Letra A es: R
Letra B es: a
Letra C es: f

C:\Users\engin\source\repos\Ejemplo\Ejemplo\bin\Debug\net8.0\Ejemplo.exe (proceso 8928) se cerró con el código 0.
Para cerrar automáticamente la consola cuando se detiene la depuración, habilite Herramientas ->Opciones ->Depuración ->Cerrar la consola automáticamente al detenerse la depuración.
Presione cualquier tecla para cerrar esta ventana. . .
```

Ilustración 14: Valores de tipo "char"

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Declara variables de tipo string
            string cadenaA = "Esta es una prueba";
            string cadenaB = "Programando en C#";
            string cadenaC = "Con Visual Studio 2022";

            //Imprime esas variables
            Console.WriteLine("Cadena A es: " + cadenaA);
            Console.WriteLine("Cadena B es: " + cadenaB);
            Console.WriteLine("Cadena C es: " + cadenaC);
        }
    }
}
```

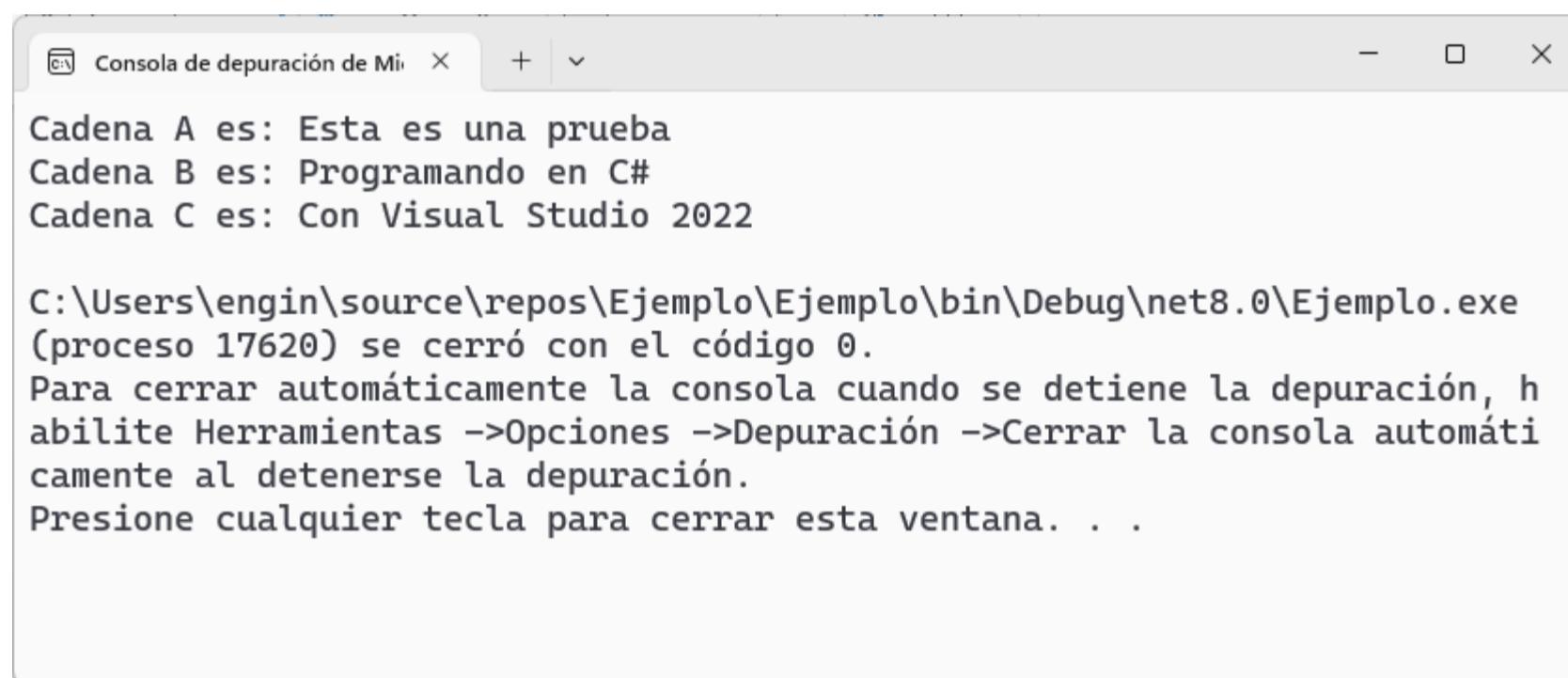


Ilustración 15: Valores de tipo "string"

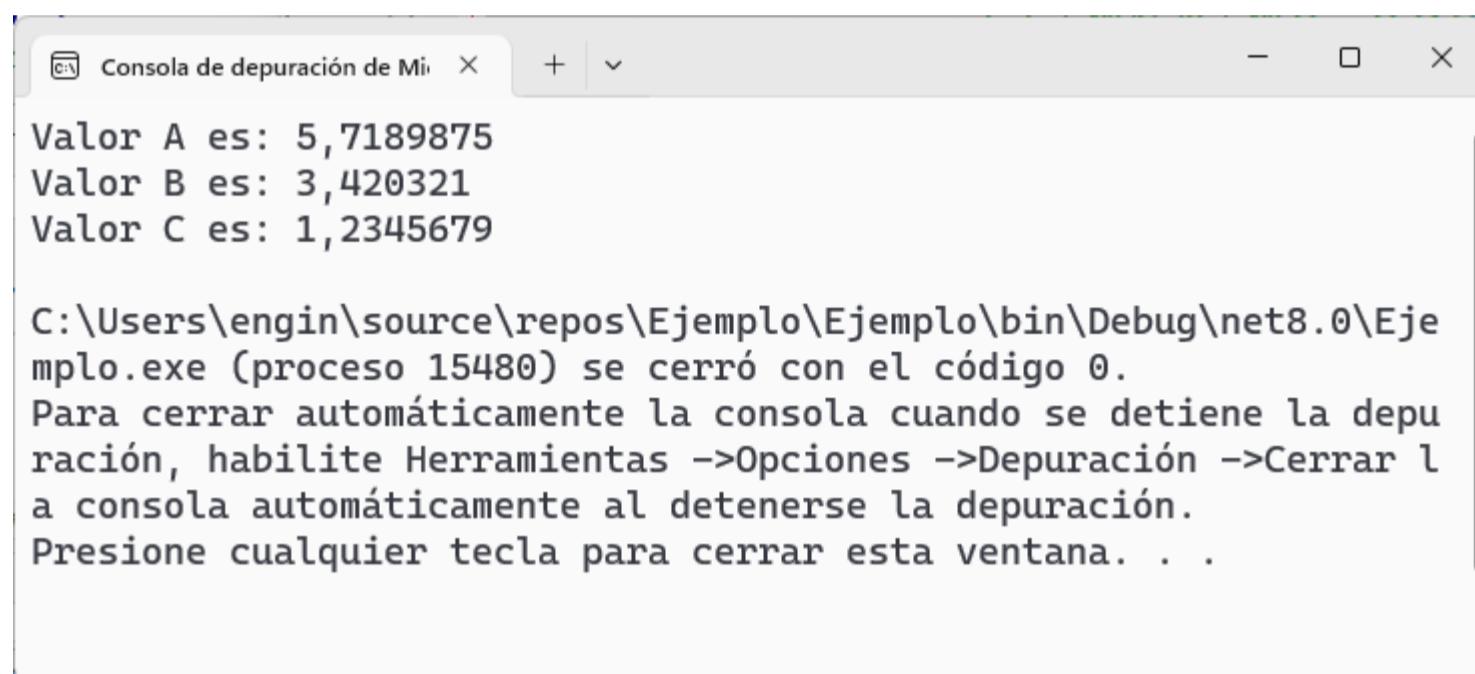


Ilustración 16: Valores almacenados por variables de tipo "float"

Por lo tanto, se recomienda hacer uso mejor de variables tipo double.

Imprimiendo con formato

Se requiere imprimir los números en consola con un determinado formato. Este es el código:

A/009.cs

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Declara variables de tipo double
            double valA = 5.718987654321;
            double valB = -3.420821098765;
            double valC = 1.234567890123456789;
            double valD = 7.8;

            //Imprime con un formato de al menos dos decimales
            Console.WriteLine("Valor A es: {0:0.00}", valA);

            //Imprime con un formato de al menos tres decimales
            Console.WriteLine("Valor B es: {0:0.000}", valB);

            //Imprime con un formato de al menos cuatro decimales
            Console.WriteLine("Valor C es: {0:0.0000}", valC);

            //Imprime con un formato de al menos cinco decimales
            Console.WriteLine("Valor D es: {0:0.00000}", valD);

            /* Imprime las tres variables. Ver el inicio {N: donde N es la posición de la variable
             * en la instrucción de impresión iniciando en cero. */
            Console.WriteLine("\r\nvalor A: {0:0.00}; valor B: {1:0.00}; valor C: {2:0.00}", valA, valB,
valC);

            /* Máximo tres decimales */
            Console.WriteLine("\r\nValor C con máximo tres decimales es: {0:0.###}", valC);
            Console.WriteLine("Valor D con máximo tres decimales es: {0:0.###}", valD);

            /* Mínimo tres dígitos antes del punto decimal */
            Console.WriteLine("Valor A con tres dígitos antes del punto decimal: {0:000.###}", valA);
            Console.WriteLine("Valor B con tres dígitos antes del punto decimal: {0:000.###}", valB);
        }
    }
}
```

```
Consola de depuración de Mi... + ▾ - □ ×

Valor A es: 5,72
Valor B es: -3,421
Valor C es: 1,2346
Valor D es: 7,80000

valor A: 5,72; valor B: -3,42; valor C: 1,23

Valor C con máximo tres decimales es: 1,235
Valor D con máximo tres decimales es: 7,8
Valor A con tres dígitos antes del punto decimal: 005,719
Valor B con tres dígitos antes del punto decimal: -003,421
```

Ilustración 17: Números impresos con determinados formatos

```

namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Declara variables de tipo double
            double valA = 17902.8421;
            double valB = -871901372.420821098765;
            double valC = 89341759342.1678;

            //Imprime con un formato de miles
            Console.WriteLine("Valor A con formato de miles es: {0:0,0.0}", valA);
            Console.WriteLine("Valor A con formato de miles y tres decimales es: {0:0,0.000}", valA);

            Console.WriteLine("\r\nValor B con formato de miles es: {0:0,0.0}", valB);
            Console.WriteLine("Valor B con formato de miles y tres decimales es: {0:0,0.000}", valB);

            Console.WriteLine("\r\nValor C con formato de miles es: {0:0,0.0}", valC);
            Console.WriteLine("Valor C con formato de miles y tres decimales es: {0:0,0.000}", valC);
        }
    }
}

```

```

Valor A con formato de miles es: 17.902,8
Valor A con formato de miles y tres decimales es: 17.902,842

Valor B con formato de miles es: -871.901.372,4
Valor B con formato de miles y tres decimales es: -871.901.372,421

Valor C con formato de miles es: 89.341.759.342,2
Valor C con formato de miles y tres decimales es: 89.341.759.342,168

```

Ilustración 18: Impresión de números de tipo double con formato

```

namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Declara variables de tipo double
            double valA = 17902.8421;
            double valB = -871901372.420821098765;
            double valC = 529403.1678;

            //Imprime con alineación a la derecha (20 espacios para poner el número)
            Console.WriteLine("Valor A es: {0,20:0.0}", valA);
            Console.WriteLine("Valor B es: {0,20:0.0}", valB);
            Console.WriteLine("Valor C es: {0,20:0.0}", valC);
        }
    }
}

```

```

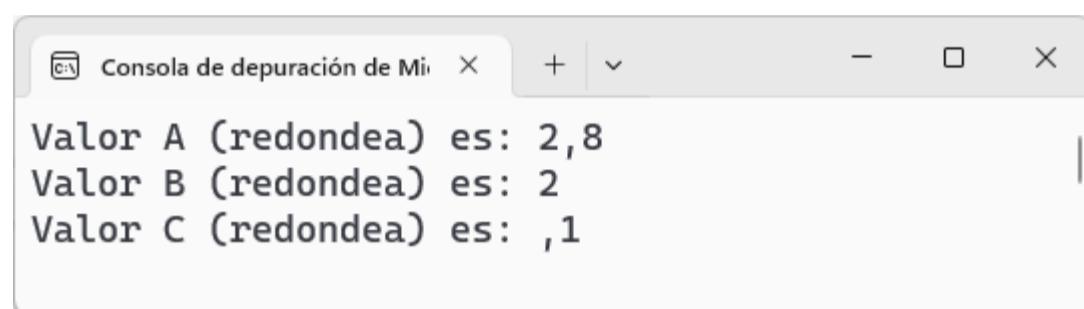
Valor A es: 17902,8
Valor B es: -871901372,4
Valor C es: 529403,2

```

Ilustración 19: Valores tipo "double" con formato

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Declara variables de tipo double
            double valA = 2.76;
            double valB = 2.04;
            double valC = 0.14;

            //Imprime solo si la cifra es significativa (redondea también)
            Console.WriteLine("Valor A (redondea) es: {0:#.#}", valA);
            Console.WriteLine("Valor B (redondea) es: {0:#.#}", valB);
            Console.WriteLine("Valor C (redondea) es: {0:#.#}", valC);
        }
    }
}
```



```
Consola de depuración de Mi + - □ ×
Valor A (redondea) es: 2,8
Valor B (redondea) es: 2
Valor C (redondea) es: ,1
```

Ilustración 20: Valores tipo "double" con formato de redondeo

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            double valA = 1 + 2 + 3; //Operador suma
            double valB = 4 - 5 - 6; //Operador resta
            double valC = 7 * 8 * 9; //Operador multiplicación
            double valD = 178 / 2 / 3; //Operador división usando enteros
            double valE = 178 / 2.0 / 3.0; //Operador división usando números reales
            double valF = (double) 178 / 2 / 3; //Operador división usando números enteros haciendo cast
            double valG = 70 % 6; //Operador división modular

            Console.WriteLine("Valor A es: {0:0.0000}", valA);
            Console.WriteLine("Valor B es: {0:0.0000}", valB);
            Console.WriteLine("Valor C es: {0:0.0000}", valC);
            Console.WriteLine("Valor D es: {0:0.0000}", valD);
            Console.WriteLine("Valor E es: {0:0.0000}", valE);
            Console.WriteLine("Valor F es: {0:0.0000}", valF);
            Console.WriteLine("Valor G es: {0:0.0000}", valG);
        }
    }
}
```

```
Valor A es: 6,0000
Valor B es: -7,0000
Valor C es: 504,0000
Valor D es: 29,0000
Valor E es: 29,6667
Valor F es: 29,6667
Valor G es: 4,0000
```

Ilustración 21: Resultados de operaciones

En valD el resultado es un entero porque C# lo toma como si fuese una división entera de números enteros por lo que el resultado es entero. En cambio, valE y valF retornan el resultado real, el primero por usar notación de número real (el uso del punto) y el otro por usar cast (double).

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Compara la precisión entre float y double
            float valA = (float)(1.7 / 2.8 * 4.8 / 6.7 / 5.3);
            double valB = 1.7 / 2.8 * 4.8 / 6.7 / 5.3;
            decimal valC = (decimal) 1.7 / (decimal) 2.8 * (decimal) 4.8 / (decimal) 6.7 / (decimal)
5.3;

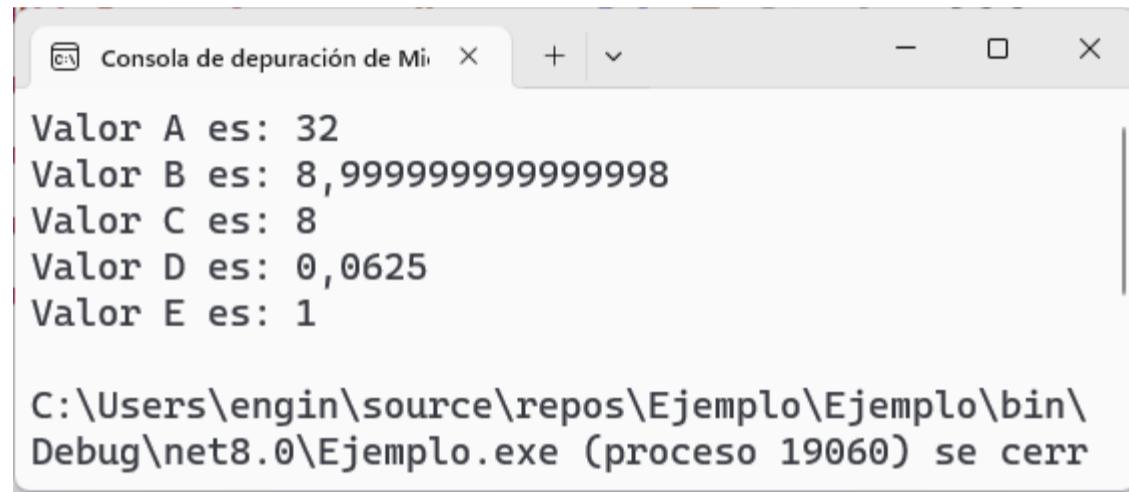
            Console.WriteLine("Valor A es: " + valA.ToString());
            Console.WriteLine("Valor B es: " + valB.ToString());
            Console.WriteLine("Valor C es: " + valC.ToString());
        }
    }
}
```

```
Valor A es: 0,082069434
Valor B es: 0,08206943718067346
Valor C es: 0,0820694371806734521462767027
```

Ilustración 22: Diferencias entre float, double y decimal

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Potencia (^)
            double valA = Math.Pow(2, 5); // 2 ^ 5
            double valB = Math.Pow(729, (double)1 / 3); //Raíz cúbica de 729
            double valC = Math.Pow(64, (double)1 / 2); //Raíz cuadrada de 64
            double valD = Math.Pow(4, -2); // 4^(-2) = 1/(4^2) = 1/16
            double valE = Math.Pow(4, 0); // 4^0

            Console.WriteLine("Valor A es: " + valA.ToString());
            Console.WriteLine("Valor B es: " + valB.ToString());
            Console.WriteLine("Valor C es: " + valC.ToString());
            Console.WriteLine("Valor D es: " + valD.ToString());
            Console.WriteLine("Valor E es: " + valE.ToString());
        }
    }
}
```



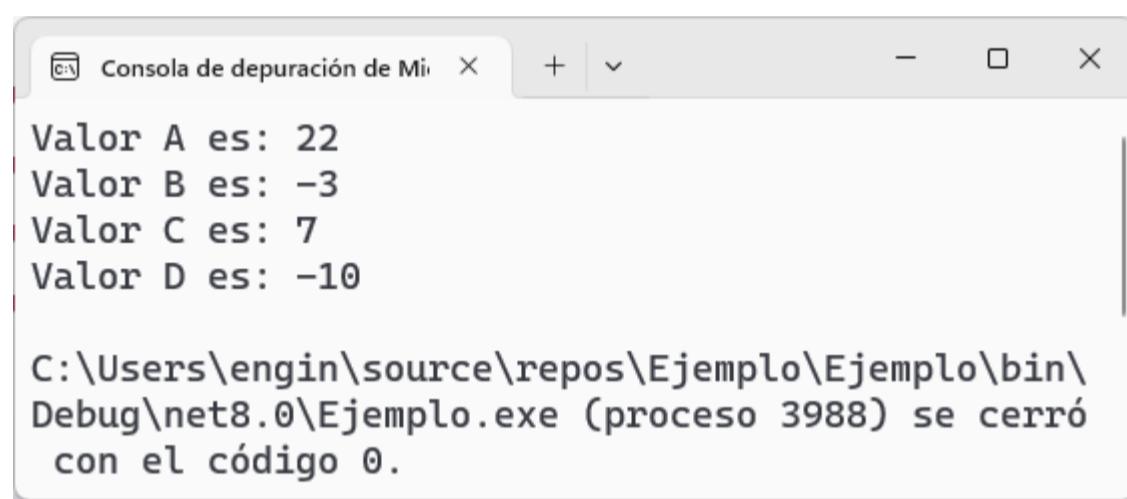
```
Valor A es: 32
Valor B es: 8,999999999999998
Valor C es: 8
Valor D es: 0,0625
Valor E es: 1

C:\Users\engin\source\repos\Ejemplo\Ejemplo\bin\Debug\net8.0\Ejemplo.exe (proceso 19060) se cerr
```

Ilustración 23: Función de potencia

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            /* Orden de evaluación de los operadores:
             * Primero potencia
             * Segundo multiplicación y división
             * Tercero suma y resta
             * */
            double valA = (double)7 - 1 + 2 * Math.Pow(2, 5) / 4;
            double valB = (double)1 + 2 * 3 / 4 - 5;
            double valC = (double)3 + 5 - 2 * 4 / 8;
            double valD = (double)3 * 2 + 5 / 10 - 2 * Math.Pow(3, 2) + 4 * 4 / 8;

            Console.WriteLine("Valor A es: " + valA.ToString());
            Console.WriteLine("Valor B es: " + valB.ToString());
            Console.WriteLine("Valor C es: " + valC.ToString());
            Console.WriteLine("Valor D es: " + valD.ToString());
        }
    }
}
```



```
Valor A es: 22
Valor B es: -3
Valor C es: 7
Valor D es: -10

C:\Users\engin\source\repos\Ejemplo\Ejemplo\bin\Debug\net8.0\Ejemplo.exe (proceso 3988) se cerró
con el código 0.
```

Ilustración 24: Orden de evaluación de los operadores

```

namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Es mejor usar reales que el cast
            double valA = 9.0 / 2.0 - 1.0 + 2.0 * Math.Pow(2.0, 5.0) / 4.0;
            double valB = (double)9 / 2 - 1 + 2 * Math.Pow(2, 5) / 4;

            //Falla porque se interpreta completamente como enteros 9 / 2 es 4 en división entera
            double valC = 9 / 2 - 1 + 2 * Math.Pow(2, 5) / 4;

            /* Falla el cast porque primero se interpreta la expresión
            como entera y luego se pasa a double */
            double valD = (double) (9 / 2 - 1 + 2 * Math.Pow(2, 5) / 4);

            Console.WriteLine("Valor A es: " + valA.ToString());
            Console.WriteLine("Valor B es: " + valB.ToString());
            Console.WriteLine("Valor C es: " + valC.ToString());
            Console.WriteLine("Valor D es: " + valD.ToString());
        }
    }
}

```

```

Valor A es: 19,5
Valor B es: 19,5
Valor C es: 19
Valor D es: 19

```

Ilustración 25: Problemas del uso del "cast"

```

namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Falla porque se interpreta como operación de enteros
            double valA = (4 / 3) + (1 / 2) - (25 / 4);

            //Falla porque el cast solo cubre la primera operación
            double valB = (double)4 / 3 + (1 / 2) - (25 / 4);

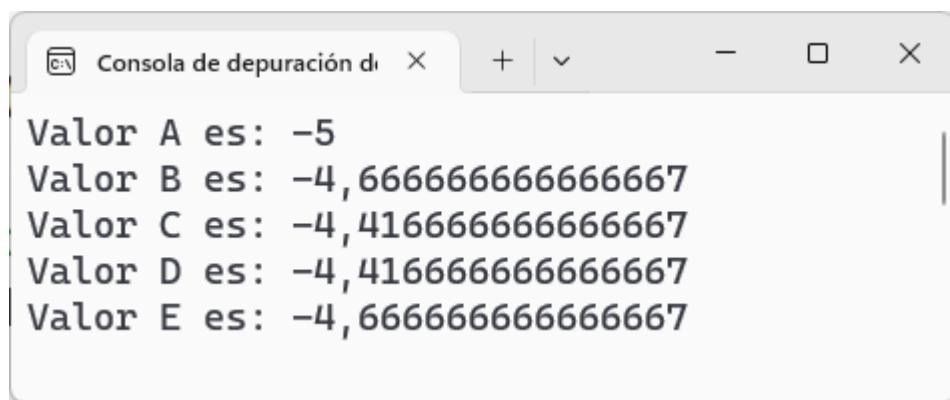
            //Operación correcta
            double valC = (4.0 / 3.0) + (1.0 / 2.0) - (25.0 / 4.0);

            //Operación correcta pero se deben usar varios cast
            double valD = (double)4 / 3 + (double)1 / 2 - (double)25 / 4;

            //Falla porque el cast solo cubre la primera operación
            double valE = (double)4 / 3 + 1 / 2 - 25 / 4;

            Console.WriteLine("Valor A es: " + valA.ToString());
            Console.WriteLine("Valor B es: " + valB.ToString());
            Console.WriteLine("Valor C es: " + valC.ToString());
            Console.WriteLine("Valor D es: " + valD.ToString());
            Console.WriteLine("Valor E es: " + valE.ToString());
        }
    }
}

```



A screenshot of a debugger's "Consola de depuración" (Debug Console) window. The title bar says "Consola de depuración di". The window contains five lines of text, each starting with "Valor" followed by a letter (A-E) and a colon, followed by a value. The values are: -5, -4,66666666666667, -4,41666666666667, -4,41666666666667, and -4,66666666666667. The values for B, C, D, and E are identical, showing a precision loss or rounding error.

```
Valor A es: -5
Valor B es: -4,66666666666667
Valor C es: -4,41666666666667
Valor D es: -4,41666666666667
Valor E es: -4,66666666666667
```

Ilustración 26: Problemas del uso del "cast"

```

namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Pasar correctamente ecuaciones a formato en C#
            double X = 5.7;
            double Y;

            //Ejemplos
            Y = 3 / (X - 1);
            Console.WriteLine("Valor Y es: " + Y.ToString());

            Y = 3 / (X - 1) + 2;
            Console.WriteLine("Valor Y es: " + Y.ToString());

            Y = 3 / X + X / 7;
            Console.WriteLine("Valor Y es: " + Y.ToString());

            Y = 3 / (X * X);
            Console.WriteLine("Valor Y es: " + Y.ToString());

            Y = 3 / (2 - (X * X + 5));
            Console.WriteLine("Valor Y es: " + Y.ToString());

            Y = (5 + X) / (6 - X * X * X);
            Console.WriteLine("Valor Y es: " + Y.ToString());

            Y = 2 + ((4 / X) / 3);
            Console.WriteLine("Valor Y es: " + Y.ToString());

            Y = 5 * ((X * X / 4) / (3 / (X + 5)));
            Console.WriteLine("Valor Y es: " + Y.ToString());

            Y = ((X * X) + ((2 * X - 5) / (X + 1))) / (1 - X * X);
            Console.WriteLine("Valor Y es: " + Y.ToString());
        }
    }
}

```

```

Valor Y es: 0,6382978723404255
Valor Y es: 2,6382978723404253
Valor Y es: 1,3406015037593986
Valor Y es: 0,09233610341643582
Valor Y es: -0,08453085376162299
Valor Y es: -0,05971215393458449
Valor Y es: 2,2339181286549707
Valor Y es: 144,85125
Valor Y es: -1,0620903105937447

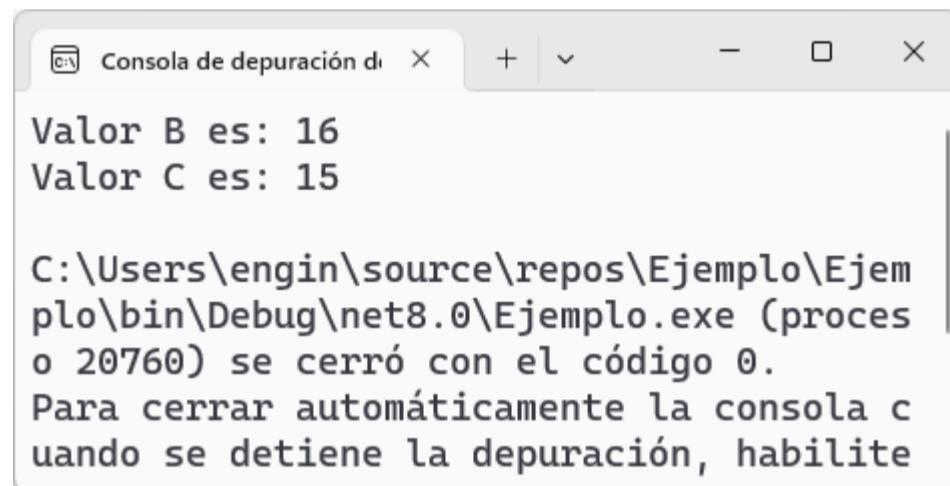
```

Ilustración 27: Formato algebraico a C#

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Conversión vs Cast
            double valA = 15.7;

            int valB = Convert.ToInt32(valA); //Redondea
            int valC = (int)valA; //Trunca

            //Ejemplos
            Console.WriteLine("Valor B es: " + valB.ToString());
            Console.WriteLine("Valor C es: " + valC.ToString());
        }
    }
}
```



```
Consola de depuración di x + - □ ×
Valor B es: 16
Valor C es: 15

C:\Users\engin\source\repos\Ejemplo\Ejemplo\bin\Debug\net8.0\Ejemplo.exe (proceso 20760) se cerró con el código 0.
Para cerrar automáticamente la consola cuando se detiene la depuración, habilite
```

Ilustración 28: Diferencias entre usar el cast y la conversión

```
using System.Globalization;

namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Conversión de reales
            string valA = "4.78";

            //Da a problemas porque se ignora el punto decimal
            double valB = Convert.ToDouble(valA);
            Console.WriteLine("Valor B es: " + valB.ToString());

            //Aquí si funciona la conversión
            string valC = "9,21";
            double valD = Convert.ToDouble(valC);
            Console.WriteLine("Valor D es: " + valD.ToString());

            //Para usar la conversión con punto decimal, se debe hacer uso de CultureInfo
            string valE = "6.8315";
            double valF = double.Parse(valE, CultureInfo.InvariantCulture);
            Console.WriteLine("Valor F es: " + valF.ToString());
        }
    }
}
```

```
Consola de depuración de Ejemplo + - □ ×
Valor B es: 478
Valor D es: 9,21
Valor F es: 6,8315

C:\Users\engin\source\repos\Ejemplo\Ejemplo\bin\Debug\net8.0\Ejemplo.exe (proceso 22824) se cerró con el código 0.
Para cerrar automáticamente la consola c
```

Ilustración 29: Conversión de números reales con el punto decimal

```
using System.Globalization;

namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Fallos de conversión
            string valA = ""; //Una cadena nula o vacía daña la conversión

            //El programa se cae
            double valB = Convert.ToDouble(valA);
            Console.WriteLine("Valor B es: " + valB.ToString());

            //El programa se cae
            double valF = double.Parse(valA, CultureInfo.InvariantCulture);
            Console.WriteLine("Valor F es: " + valF.ToString());
        }
    }
}
```

//El programa se cae
double valB = Convert.ToDouble(valA); X
Console.WriteLine("Valor B es: " + valB.ToString());

Excepción no controlada ✖

System.FormatException: 'The input string " was not in a correct format.'

[Mostrar pila de llamadas](#) | [Ver detalles](#) | [Copiar detalles](#) | [Iniciar sesión de Live Share](#)

► [Configuración de excepciones](#)

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Constantes matemáticas
            double valA = Math.PI;
            double valB = Math.E;
            Console.WriteLine("PI es: " + valA.ToString());
            Console.WriteLine("E es: " + valB.ToString());

            //Máximo y mínimo valor entero
            int maximoEntero = int.MaxValue;
            int minimoEntero = int.MinValue;
            Console.WriteLine("Máximo entero: " + maximoEntero.ToString());
            Console.WriteLine("Mínimo entero: " + minimoEntero.ToString());

            //Máximo y mínimo valor float
            float maximofloat = float.MaxValue;
            float minimofloat = float.MinValue;
            float minimoCero = float.Epsilon; //El mínimo valor antes de ser cero
            Console.WriteLine("Máximo float: " + maximofloat.ToString());
            Console.WriteLine("Mínimo float: " + minimofloat.ToString());
            Console.WriteLine("El mínimo float antes de ser cero: " + minimoCero.ToString());

            //Máximo y mínimo valor double
            double maximodouble = double.MaxValue;
            double minimodouble = double.MinValue;
            double minimoCerodouble = double.Epsilon; //El mínimo valor antes de ser cero
            Console.WriteLine("Máximo double: " + maximodouble.ToString());
            Console.WriteLine("Mínimo double: " + minimodouble.ToString());
            Console.WriteLine("El mínimo double antes de ser cero: " + minimoCerodouble.ToString());
        }
    }
}
```

```
PI es: 3,141592653589793
E es: 2,718281828459045
Máximo entero: 2147483647
Mínimo entero: -2147483648
Máximo float: 3,4028235E+38
Mínimo float: -3,4028235E+38
El mínimo float antes de ser cero: 1E-45
Máximo double: 1,7976931348623157E+308
Mínimo double: -1,7976931348623157E+308
El mínimo double antes de ser cero: 5E-324
```

Ilustración 30: Constantes matemáticas

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Funciones trigonométricas
            double anguloGrados = 60;
            double anguloRadian = anguloGrados * Math.PI / 180;

            double valSen = Math.Sin(anguloRadian);
            double valCoseno = Math.Cos(anguloRadian);
            double valTangente = Math.Tan(anguloRadian);

            double arcoSen = Math.Asin(valSen);
            double arcoCoseno = Math.Acos(valCoseno);
            double arcoTangente = Math.Atan(valTangente);

            Console.WriteLine("Ángulo en grados es: " + anguloGrados.ToString());
            Console.WriteLine("Ángulo en radianes: " + anguloRadian.ToString());
            Console.WriteLine("Sen es: " + valSen.ToString());
            Console.WriteLine("Coseno es: " + valCoseno.ToString());
            Console.WriteLine("Tangente es: " + valTangente.ToString());
            Console.WriteLine("arcoSen es: " + arcoSen.ToString());
            Console.WriteLine("arcoCoseno es: " + arcoCoseno.ToString());
            Console.WriteLine("arcoTangente es: " + arcoTangente.ToString());
        }
    }
}
```

```
Consola de depuración de Mi... X + - □ ×
Ángulo en grados es: 60
Ángulo en radianes: 1,0471975511965976
Sen es: 0,8660254037844386
Coseno es: 0,5000000000000001
Tangente es: 1,7320508075688767
arcoSen es: 1,0471975511965976
arcoCoseno es: 1,0471975511965976
arcoTangente es: 1,0471975511965976

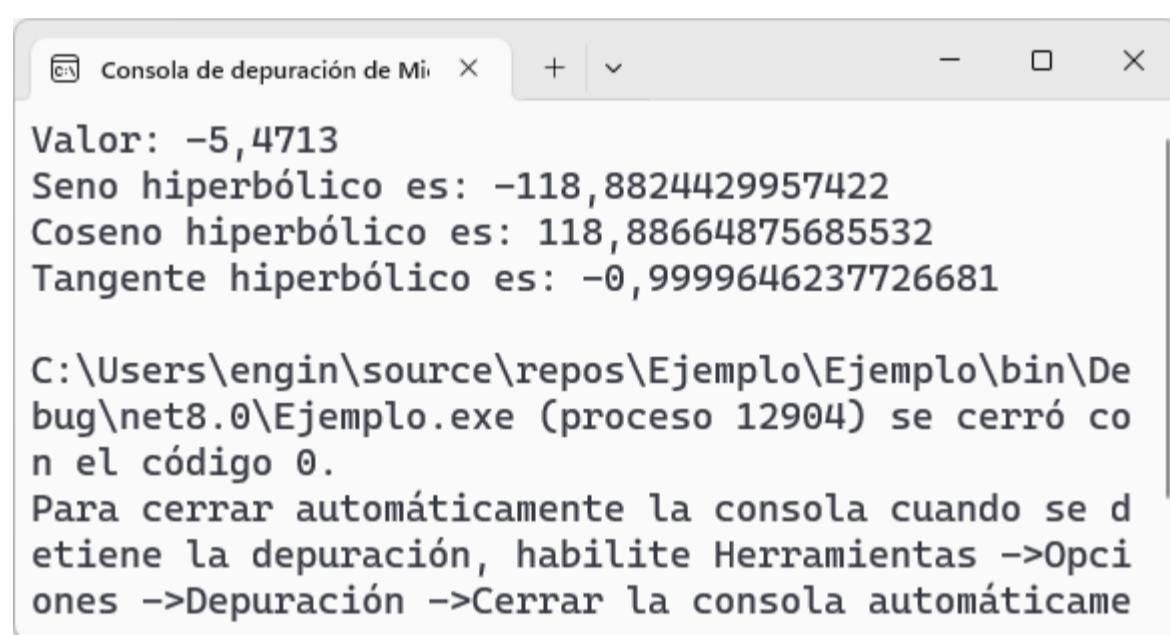
C:\Users\engin\source\repos\Ejemplo\Ejemplo\bin\De
bug\net8.0\Ejemplo.exe (proceso 15560) se cerró co
```

Ilustración 31: Funciones trigonométricas

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Funciones hiperbólicas
            double valorReal = -5.4713;

            double valSenoH = Math.Sinh(valorReal);
            double valCosenoH = Math.Cosh(valorReal);
            double valTangenteH = Math.Tanh(valorReal);

            Console.WriteLine("Valor: " + valorReal.ToString());
            Console.WriteLine("Seno hiperbólico es: " + valSenoH.ToString());
            Console.WriteLine("Coseno hiperbólico es: " + valCosenoH.ToString());
            Console.WriteLine("Tangente hiperbólico es: " + valTangenteH.ToString());
        }
    }
}
```



Consola de depuración de Mi... X + - □ ×

Valor: -5,4713
Seno hiperbólico es: -118,8824429957422
Coseno hiperbólico es: 118,88664875685532
Tangente hiperbólico es: -0,9999646237726681

C:\Users\engin\source\repos\Ejemplo\Ejemplo\bin\Debug\net8.0\Ejemplo.exe (proceso 12904) se cerró con el código 0.
Para cerrar automáticamente la consola cuando se detiene la depuración, habilite Herramientas ->Opciones ->Depuración ->Cerrar la consola automáticamente

Ilustración 32: Funciones hiperbólicas

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Otras funciones matemáticas
            double valorReal = 5.0713;
            double valAbsoluto = Math.Abs(valorReal); //valor absoluto
            double valTecho = Math.Ceiling(valorReal); //entero superior
            double valExponencial = Math.Exp(valorReal);
            double valPiso = Math.Floor(valorReal);
            double valLogaritmoNatural = Math.Log(valorReal);
            double valLogaritmoBase10 = Math.Log10(valorReal);
            double valMaximoEntero = Math.Max(7, 19);
            double valMinimoEntero = Math.Min(7, 19);
            double valRedondea = Math.Round(valorReal);
            double valSigno = Math.Sign(valorReal);
            double valRaizC = Math.Sqrt(valorReal); //raíz cuadrada
            double valTrunca = Math.Truncate(valorReal);

            Console.WriteLine("Valor: " + valorReal.ToString());
            Console.WriteLine("Valor absoluto es: " + valAbsoluto.ToString());
            Console.WriteLine("Valor techo es: " + valTecho.ToString());
            Console.WriteLine("Exponencial es: " + valExponencial.ToString());
            Console.WriteLine("Valor piso es: " + valPiso.ToString());
            Console.WriteLine("Logaritmo Natural es: " + valLogaritmoNatural.ToString());
            Console.WriteLine("Logaritmo Base 10 es: " + valLogaritmoBase10.ToString());
            Console.WriteLine("Máximo entero es: " + valMaximoEntero.ToString());
            Console.WriteLine("Mínimo entero es: " + valMinimoEntero.ToString());
            Console.WriteLine("Redondea: " + valRedondea.ToString());
            Console.WriteLine("Signo es: " + valSigno.ToString());
            Console.WriteLine("Raíz cuadrada es: " + valRaizC.ToString());
            Console.WriteLine("Trunca: " + valTrunca.ToString());

            //Funciones con dos salidas
            int residuo;
            int cociente = Math.DivRem(29, 4, out residuo);
            Console.WriteLine("29/4 cociente: " + cociente.ToString() + " residuo: " +
residuo.ToString());

            //Multiplicación enorme
            long valMultiplica = Math.BigMul(123456789, 987654321);
            Console.WriteLine("Multiplicación enorme: " + valMultiplica.ToString());

            //División modular, operación y funciones
            decimal dividendo = 100, divisor = 34;
            decimal residuoA = dividendo % divisor;
            decimal residuoB = (Math.Abs(dividendo) - (Math.Abs(divisor) *
(Math.Floor(Math.Abs(dividendo) / Math.Abs(divisor))))) * Math.Sign(dividendo);
            Console.WriteLine("ResiduoA: " + residuoA.ToString() + " ResiduoB: " + residuoB.ToString());
        }
    }
}
```

```
Consola de depuración de Mi... + - X
Valor: 5,0713
Valor absoluto es: 5,0713
Valor techo es: 6
Exponencial es: 159,38138852945326
Valor piso es: 5
Logaritmo Natural es: 1,6235971949920105
Logaritmo Base 10 es: 0,7051193026186281
Máximo entero es: 19
Mínimo entero es: 7
Redondea: 5
Signo es: 1
Raiz cuadrada es: 2,251954706471691
Trunca: 5
29/4 cociente: 7 residuo: 1
Multiplicación enorme: 121932631112635269
ResiduoA: 32 ResiduoB: 32
```

Ilustración 33: Otras funciones matemáticas

Manejo del NaN (Not a Number) y el infinito

Cuando algunas operaciones matemáticas generan error (por ejemplo, raíz cuadrada de un número negativo), la función retorna NaN (Not a Number). También hay operaciones que dan infinito como la división entre cero, en ese caso en consola se muestra como si fuese el número 8 (ochos), y es más bien un infinito girado 90 grados.

A/027.cs

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            // Genera un NaN: Not a Number
            double valorReal = -70;
            double valLogaritmoNatural = Math.Log(valorReal);
            double valLogaritmoBase10 = Math.Log10(valorReal);
            double valRaizC = Math.Sqrt(valorReal); //raiz cuadrada
            double arcoSeno = Math.Asin(valorReal);
            double arcoCoseno = Math.Acos(valorReal);

            Console.WriteLine("Valor: " + valorReal.ToString());
            Console.WriteLine("Logaritmo Natural es: " + valLogaritmoNatural.ToString());
            Console.WriteLine("Logaritmo Base 10 es: " + valLogaritmoBase10.ToString());
            Console.WriteLine("Raiz cuadrada es: " + valRaizC.ToString());
            Console.WriteLine("Arcoseno es: " + arcoSeno.ToString());
            Console.WriteLine("Arcocoseno es: " + arcoCoseno.ToString());

            //Genera infinito positivo
            float valA = 10;
            float valB = 0;
            float valC = valA / valB;
            Console.WriteLine("Valor C es: " + valC);

            //Genera infinito negativo
            valA = -10;
            valB = 0;
            valC = valA / valB;
            Console.WriteLine("Valor C es: " + valC);
        }
    }
}
```

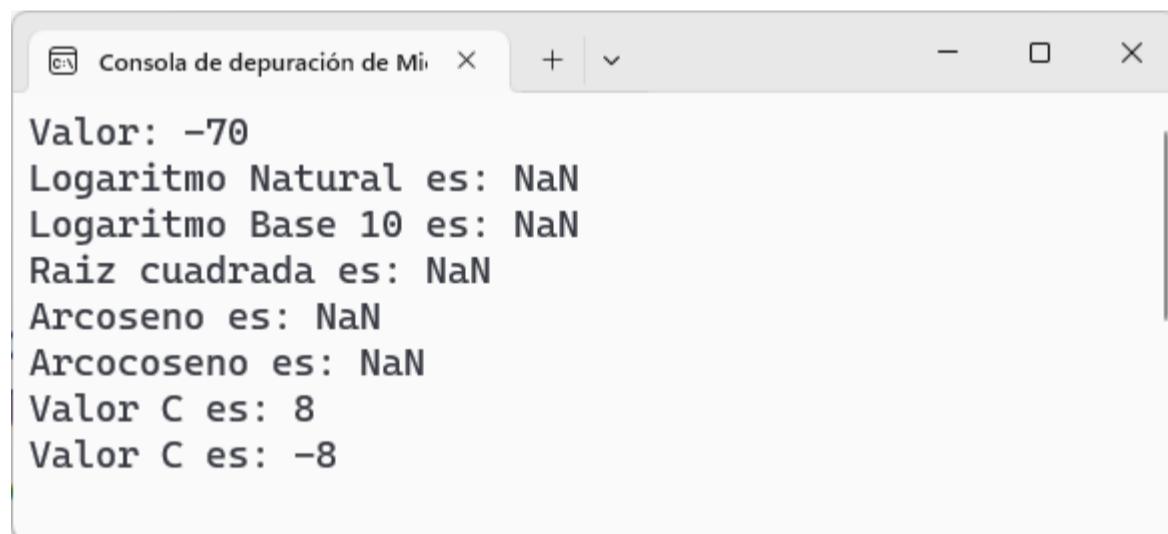


Ilustración 34: Manejo del NaN (Not a Number) y el infinito

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Leer un número por consola
            Console.Write("Escriba un número: ");
            double valorReal = Convert.ToDouble(Console.ReadLine());
            Console.WriteLine("Escribió: " + valorReal.ToString());
        }
    }
}
```

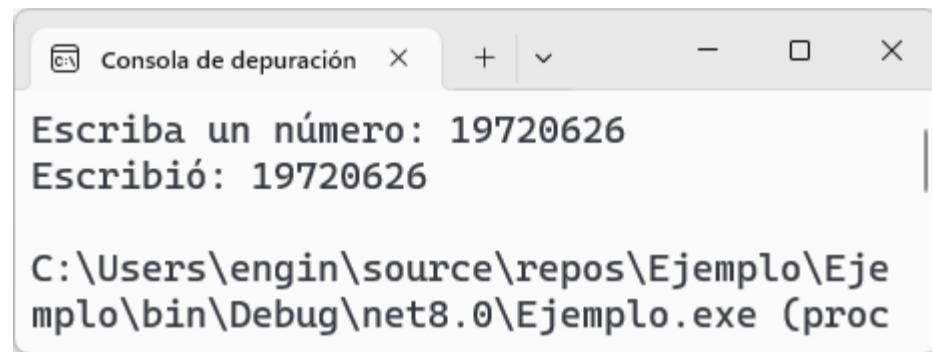


Ilustración 35: Leer un número por consola

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Lee dos números por consola
            Console.Write("Escriba un primer número: ");
            double valorA = Convert.ToDouble(Console.ReadLine());

            Console.Write("Escriba un segundo número: ");
            double valorB = Convert.ToDouble(Console.ReadLine());

            //Si condicional
            if (valorA > valorB) {
                Console.WriteLine(valorA.ToString() + " es mayor que " + valorB.ToString());
            }

            if (valorA >= valorB) {
                Console.WriteLine(valorA.ToString() + " es mayor o igual que " + valorB.ToString());
            }

            if (valorA < valorB) {
                Console.WriteLine(valorA.ToString() + " es menor que " + valorB.ToString());
            }

            if (valorA <= valorB) {
                Console.WriteLine(valorA.ToString() + " es menor o igual que " + valorB.ToString());
            }

            if (valorA == valorB) {
                Console.WriteLine(valorA.ToString() + " es igual a " + valorB.ToString());
            }

            if (valorA != valorB) {
                Console.WriteLine(valorA.ToString() + " es diferente de " + valorB.ToString());
            }
        }
    }
}
```

The screenshot shows a command prompt window with the title 'Consola de depuración de Mi'. The window contains the following text:

```
Escriba un primer número: 31276
Escriba un segundo número: 97902
31276 es menor que 97902
31276 es menor o igual que 97902
31276 es diferente de 97902
```

Ilustración 36: Si condicional

```

namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Lee dos números por consola
            Console.Write("Escriba un primer número: ");
            double valorA = Convert.ToDouble(Console.ReadLine());

            Console.Write("Escriba un segundo número: ");
            double valorB = Convert.ToDouble(Console.ReadLine());

            //Si condicional
            if (valorA > valorB) {
                Console.WriteLine(valorA.ToString() + " es mayor que " + valorB.ToString());
            }
            else { //de lo contrario
                Console.WriteLine(valorA.ToString() + " es menor o igual que " + valorB.ToString());
            }
        }
    }
}

```

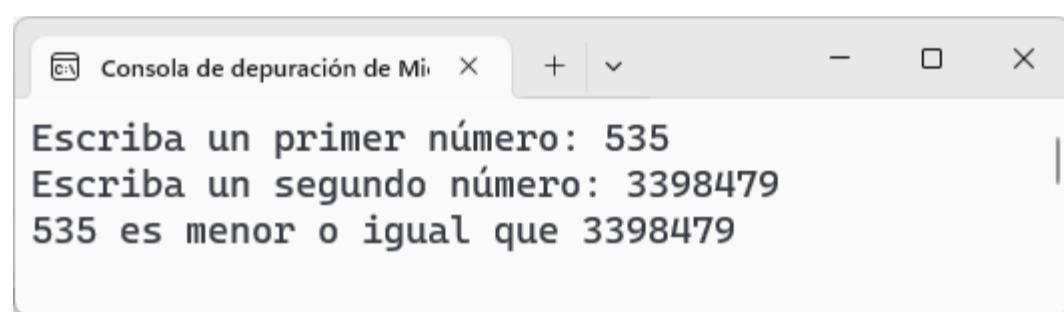


Ilustración 37: Uso de if...else

```

namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Lee dos números por consola
            Console.Write("Escriba un primer número: ");
            double valorA = Convert.ToDouble(Console.ReadLine());

            Console.Write("Escriba un segundo número: ");
            double valorB = Convert.ToDouble(Console.ReadLine());

            //Si condicional
            if (valorA > valorB) {
                Console.WriteLine(valorA.ToString() + " es mayor que " + valorB.ToString());
            }
            else if (valorA < valorB) {
                Console.WriteLine(valorA.ToString() + " es menor que " + valorB.ToString());
            }
            else {
                Console.WriteLine(valorA.ToString() + " es igual a " + valorB.ToString());
            }
        }
    }
}

```

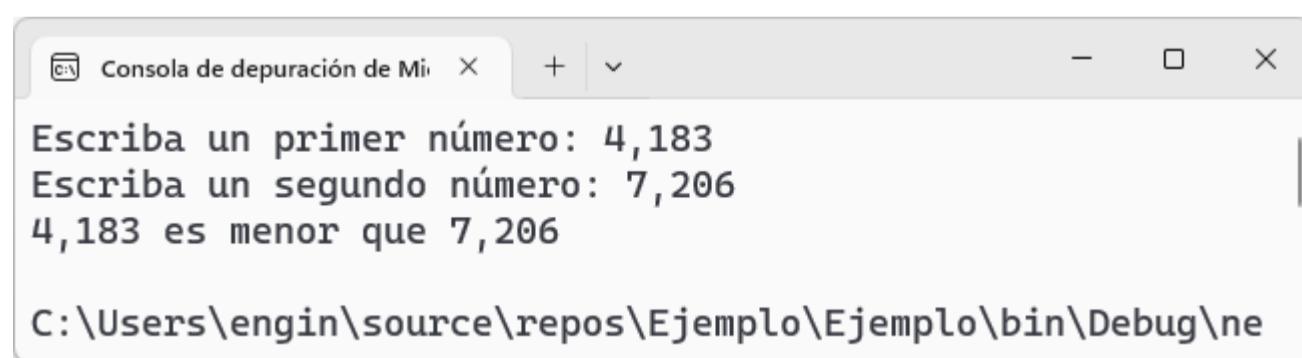


Ilustración 38: Uso de if... else if ... else

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Lee los lados de un triángulo
            Console.WriteLine("Escriba valor lado A: ");
            double ladoA = Convert.ToDouble(Console.ReadLine());

            Console.WriteLine("Escriba valor lado B: ");
            double ladoB = Convert.ToDouble(Console.ReadLine());

            Console.WriteLine("Escriba valor lado C: ");
            double ladoC = Convert.ToDouble(Console.ReadLine());

            //Si condicional, uso del AND &&
            if (ladoA == ladoB && ladoA == ladoC) {
                Console.WriteLine("Triángulo equilátero");
            }
            else if (ladoA != ladoB && ladoA != ladoC && ladoB != ladoC) {
                Console.WriteLine("Triángulo escaleno");
            }
            else {
                Console.WriteLine("Triángulo isósceles");
            }
        }
    }
}
```

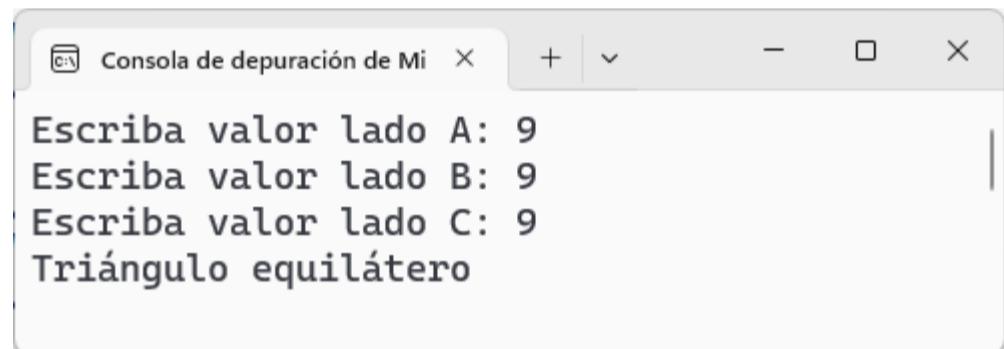


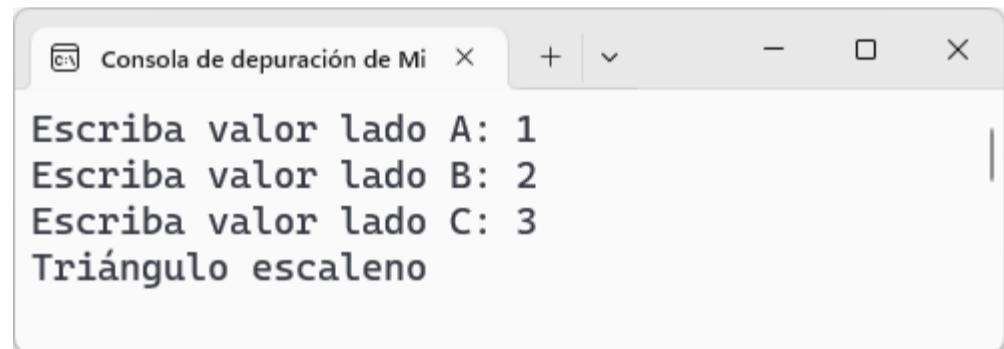
Ilustración 39: Uso del AND &&

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Lee los lados de un triángulo
            Console.WriteLine("Escriba valor lado A: ");
            double ladoA = Convert.ToDouble(Console.ReadLine());

            Console.WriteLine("Escriba valor lado B: ");
            double ladoB = Convert.ToDouble(Console.ReadLine());

            Console.WriteLine("Escriba valor lado C: ");
            double ladoC = Convert.ToDouble(Console.ReadLine());

            //Si condicional, uso del OR ||
            if (ladoA == ladoB || ladoA == ladoC || ladoB == ladoC) {
                Console.WriteLine("Triángulo equilátero o isósceles");
            }
            else {
                Console.WriteLine("Triángulo escaleno");
            }
        }
    }
}
```



```
Escriba valor lado A: 1
Escriba valor lado B: 2
Escriba valor lado C: 3
Triángulo escaleno
```

Ilustración 40: Uso del OR ||

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Lee un valor entero
            Console.WriteLine("Escriba un valor entero: ");
            int valor = Convert.ToInt32(Console.ReadLine());

            switch (valor) {
                case 1: Console.WriteLine("Escribió uno"); break;
                case 2: Console.WriteLine("Escribió dos, ");
                    Console.WriteLine("que es un número par");
                    break; //Hay que terminar con break
                case 3: //Esto sería el equivalente a un OR
                case 4:
                case 5: Console.WriteLine("Escribió 3 o 4 o 5");
                    break;
                default: Console.WriteLine("Escribió un número por fuera del rango de 1 a 5");
                    break; //Inclusive el default requiere break
            }
        }
    }
}
```

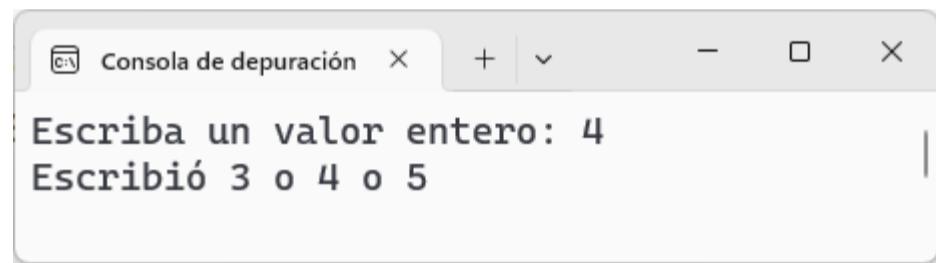


Ilustración 41: Uso del switch

```

namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Operadores booleanos
            bool valA = true;
            bool valB = false;

            bool Resultado1 = valA & valB; //Operador Y
            bool Resultado2 = valA | valB; //Operador O
            bool Resultado3 = valA ^ valB; //Operador XOR
            bool Resultado4 = !valA; //Operador negación

            Console.WriteLine(Resultado1);
            Console.WriteLine(Resultado2);
            Console.WriteLine(Resultado3);
            Console.WriteLine(Resultado4);
        }
    }
}

```

```

False
True
True
False

```

Ilustración 42: Operadores booleanos

```

namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Tabla del AND
            Console.WriteLine("\r\nTabla del operador AND");
            Console.WriteLine(true & true);
            Console.WriteLine(true & false);
            Console.WriteLine(false & true);
            Console.WriteLine(false & false);

            //Tabla del OR
            Console.WriteLine("\r\nTabla del operador OR");
            Console.WriteLine(true | true);
            Console.WriteLine(true | false);
            Console.WriteLine(false | true);
            Console.WriteLine(false | false);

            //Tabla del XOR
            Console.WriteLine("\r\nTabla del operador XOR");
            Console.WriteLine(true ^ true);
            Console.WriteLine(true ^ false);
            Console.WriteLine(false ^ true);
            Console.WriteLine(false ^ false);
        }
    }
}

```

```
Consola de depuración + - X

Tabla del operador AND
True
False
False
False

Tabla del operador OR
True
True
True
False

Tabla del operador XOR
False
True
True
False
```

Ilustración 43: Tablas de verdad

A/037.cs

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Tabla del AND
            Console.WriteLine("\r\nTabla del operador AND");
            Console.WriteLine(true && true);
            Console.WriteLine(true && false);
            Console.WriteLine(false && true);
            Console.WriteLine(false && false);

            //Tabla del OR
            Console.WriteLine("\r\nTabla del operador OR");
            Console.WriteLine(true || true);
            Console.WriteLine(true || false);
            Console.WriteLine(false || true);
            Console.WriteLine(false || false);

            /* La diferencia entre & y &&, | y || es que cuando se
             * usa & se evalúan ambos operandos a la izquierda y derecha del &,
             * en cambio cuando se usa && se evalúa primero el operando de la izquierda
             * y si da falso, ya se sabe que toda la expresión es falsa. Luego en
             * un si condicional, se ejecuta más rápido si se usa && en vez de &.
             * Similar se aplica para || o | , si se usa en un si condicional
             * evalúa el primer operando, si es verdadero, toda la expresión es verdadera. */
        }
    }
}
```

```
Consola de depuración × + - □ X

Tabla del operador AND
True
False
False
False

Tabla del operador OR
True
True
True
False
```

Ilustración 44: Tablas de verdad usando && y ||

A/038.cs

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            Console.WriteLine("\r\nCompleja expresión booleana");
            Console.WriteLine(true && true || false && true || true || false);
            Console.WriteLine(true || false && true || false && true && true);
            Console.WriteLine(!false && !true || !true && false && !false);

            /* No se recomienda en una expresión lógica poner dos o más operadores
             * lógicos. Llega a confundir.
             * Hacer uso de paréntesis. */
        }
    }
}
```

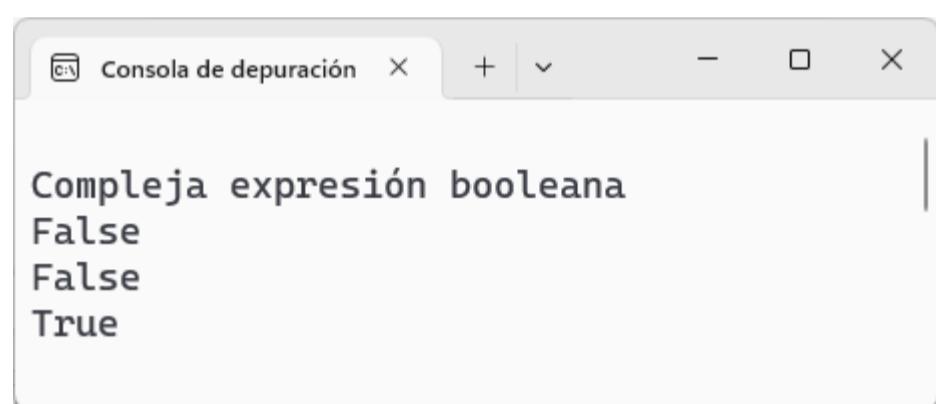
```
Consola de depuración × + - □ X

Compleja expresión booleana
True
True
False
```

Ilustración 45: Expresión booleana

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            Console.WriteLine("\r\nCompleja expresión booleana");
            Console.WriteLine(!true && true | !false & true ^ true & !false);
            Console.WriteLine(!true | false & true ^ false && !true & true);
            Console.WriteLine(!false | !true ^ !true & false & !false);

            /* Precedencia de los operadores.
Tomado de: https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/boolean-logical-operators
            Logical negation operator !
            Logical AND operator &
            Logical exclusive OR operator ^
            Logical OR operator |
            Conditional logical AND operator &&
            Conditional logical OR operator ||
        }
    }
}
```



```
Compleja expresión booleana
False
False
True
```

Ilustración 46: Precedencia de los operadores

El operador "?", un si condicional

Usado en una asignación, funciona de esta manera:

```
Variable = condición ? valor si es verdadero: valor si es falso;
```

A/040.cs

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Operador ?
            int valA = 10;
            int valB = 13;
            string resultado = valA > valB ? "A es mayor": "B es mayor o igual";
            Console.WriteLine(resultado);

            //Segundo ejemplo
            int valC = 10;
            int valD = 13;
            int valE = 34;
            string imprimir = valE > valD && valC <= valD ? "Primero" : "de lo contrario, Segundo";
            Console.WriteLine(imprimir);
        }
    }
}
```

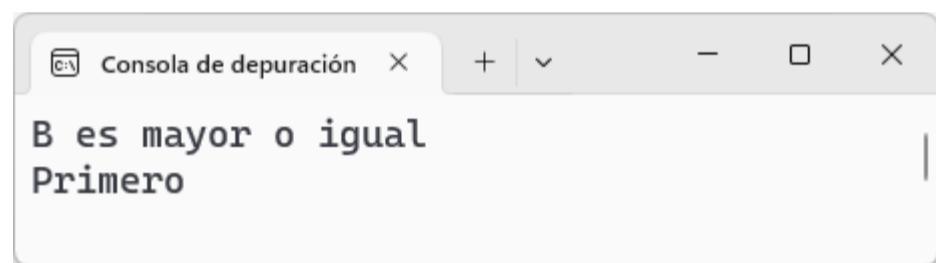


Ilustración 47: El operador "?", un si condicional

Captura de error con try...catch

Se captura el error y así se evita que el programa colapse.

A/041.cs

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Uso del try...catch
            int valA = 17;
            int valB = 0;
            int resultado;
            try {
                resultado = valA / valB; //Intenta dividir entre cero
                Console.WriteLine("Resultado es: " + resultado.ToString());
            }
            catch { //Captura el error
                Console.WriteLine("División entre cero");
            }
        }
    }
}
```

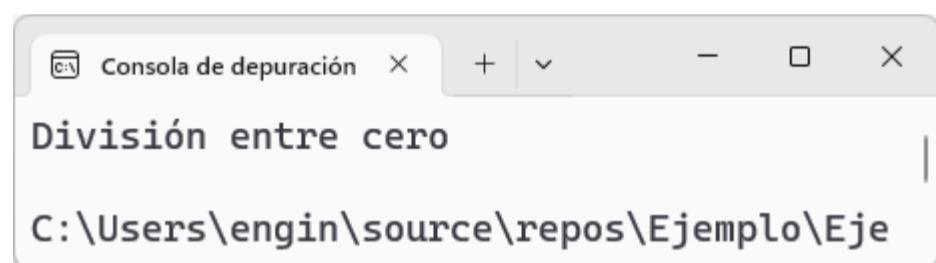


Ilustración 48: Captura de error con try...catch

```

namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Uso del TryParse. Trata de convertir un string a entero
            string Numero = "150";
            int valorEntero;
            if (Int32.TryParse(Numero, out valorEntero)) {
                Console.WriteLine("Se pudo convertir. valorEntero: " + valorEntero.ToString());
            }
            else {
                Console.WriteLine("1. No se puede convertir a entero");
            }

            //Segundo ejemplo
            Numero = "4.178"; //Un punto en la cadena
            if (Int32.TryParse(Numero, out valorEntero)) {
                Console.WriteLine("Se pudo convertir. valorEntero: " + valorEntero.ToString());
            }
            else {
                Console.WriteLine("2. No se puede convertir a entero");
            }

            //Tercer ejemplo
            Numero = "      90 "; //espacios en la cadena
            if (Int32.TryParse(Numero, out valorEntero)) {
                Console.WriteLine("Se pudo convertir. valorEntero: " + valorEntero.ToString());
            }
            else {
                Console.WriteLine("3. No se puede convertir a entero");
            }

            //Cuarto ejemplo
            Numero = "-90";
            if (Int32.TryParse(Numero, out valorEntero)) {
                Console.WriteLine("Se pudo convertir. valorEntero: " + valorEntero.ToString());
            }
            else {
                Console.WriteLine("4. No se puede convertir a entero");
            }

            //Quinto ejemplo
            Numero = "- 90"; //Espacio intermedio
            if (Int32.TryParse(Numero, out valorEntero)) {
                Console.WriteLine("Se pudo convertir. valorEntero: " + valorEntero.ToString());
            }
            else {
                Console.WriteLine("5. No se puede convertir a entero");
            }
        }
    }
}

```

Se pudo convertir. valorEntero: 150
2. No se puede convertir a entero
Se pudo convertir. valorEntero: 90
Se pudo convertir. valorEntero: -90
5. No se puede convertir a entero

Ilustración 49: Uso de TryParse para conversión

Ciclo for

Su sintaxis es:

```
for (variable=valor inicio; condición para que se mantenga el ciclo;  
incremento/decremento de esa variable) { }
```

A/043.cs

```
namespace Ejemplo {  
    internal class Program {  
        static void Main() {  
            //Ciclo for ascendente  
            Console.WriteLine("Ciclo ascendente:");  
            for (int Contador = 1; Contador <= 20; Contador++) {  
                Console.Write(Contador.ToString() + ", ");  
            }  
  
            //Ciclo for descendente  
            Console.WriteLine("\r\nCiclo descendente:");  
            for (int Contador = 20; Contador >= 1; Contador--) {  
                Console.Write(Contador.ToString() + ", ");  
            }  
  
            //Ciclo for ascendente, avance de 2 en 2  
            Console.WriteLine("\r\nCiclo ascendente (avance de 2 en 2):");  
            for (int Contador = 1; Contador <= 20; Contador += 2) {  
                Console.Write(Contador.ToString() + ", ");  
            }  
  
            //Ciclo for descendente, retrocede de 2 en 2  
            Console.WriteLine("\r\nCiclo descendente (retrocede de 2 en 2):");  
            for (int Contador = 20; Contador >= 1; Contador -= 2) {  
                Console.Write(Contador.ToString() + ", ");  
            }  
  
            //Ciclo for ascendente, avance doble  
            Console.WriteLine("\r\nCiclo ascendente (avance doble):");  
            for (int Contador = 1; Contador <= 20; Contador *= 2) {  
                Console.Write(Contador.ToString() + ", ");  
            }  
  
            //Ciclo for descendente, retrocede la mitad  
            Console.WriteLine("\r\nCiclo descendente (retrocede la mitad):");  
            for (int Contador = 20; Contador >= 1; Contador /= 2) {  
                Console.Write(Contador.ToString() + ", ");  
            }  
        }  
    }  
}
```

```
Ciclo ascendente:  
1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20,  
Ciclo descendente:  
20, 19, 18, 17, 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1,  
Ciclo ascendente (avance de 2 en 2):  
1, 3, 5, 7, 9, 11, 13, 15, 17, 19,  
Ciclo descendente (retrocede de 2 en 2):  
20, 18, 16, 14, 12, 10, 8, 6, 4, 2,  
Ciclo ascendente (avance doble):  
1, 2, 4, 8, 16,  
Ciclo descendente (retrocede la mitad):  
20, 10, 5, 2, 1,
```

Ilustración 50: Ciclo for

Ciclo while

Su sintaxis es:

```
while (condición para que se mantenga en el ciclo) {  
}
```

A/044.cs

```
namespace Ejemplo {  
    internal class Program {  
        static void Main() {  
            int Contador;  
  
            //Ciclo while ascendente  
            Console.WriteLine("Ciclo ascendente:");  
            Contador = 1;  
            while (Contador <= 20) {  
                Console.Write(Contador.ToString() + ", ");  
                Contador++;  
            }  
  
            //Ciclo while descendente  
            Console.WriteLine("\r\nCiclo descendente:");  
            Contador = 20;  
            while (Contador >= 1) {  
                Console.Write(Contador.ToString() + ", ");  
                Contador--;  
            }  
  
            //Ciclo while ascendente, avance de 2 en 2  
            Console.WriteLine("\r\nCiclo ascendente (avance de 2 en 2):");  
            Contador = 1;  
            while (Contador <= 20) {  
                Console.Write(Contador.ToString() + ", ");  
                Contador += 2;  
            }  
  
            //Ciclo while descendente, retrocede de 2 en 2  
            Console.WriteLine("\r\nCiclo descendente (retrocede de 2 en 2):");  
            Contador = 20;  
            while (Contador >= 1) {  
                Console.Write(Contador.ToString() + ", ");  
                Contador -= 2;  
            }  
  
            //Ciclo while ascendente, avance doble  
            Console.WriteLine("\r\nCiclo ascendente (avance doble):");  
            Contador = 1;  
            while (Contador <= 20) {  
                Console.Write(Contador.ToString() + ", ");  
                Contador *= 2;  
            }  
  
            //Ciclo while descendente, retrocede la mitad  
            Console.WriteLine("\r\nCiclo descendente (retrocede la mitad):");  
            Contador = 20;  
            while (Contador >= 1) {  
                Console.Write(Contador.ToString() + ", ");  
                Contador /= 2;  
            }  
        }  
    }  
}
```

```
Consola de depuración de Mi... X + ▾ - □ ×
Ciclo ascendente:
1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20,
Ciclo descendente:
20, 19, 18, 17, 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1,
Ciclo ascendente (avance de 2 en 2):
1, 3, 5, 7, 9, 11, 13, 15, 17, 19,
Ciclo descendente (retrocede de 2 en 2):
20, 18, 16, 14, 12, 10, 8, 6, 4, 2,
Ciclo ascendente (avance doble):
1, 2, 4, 8, 16,
Ciclo descendente (retrocede la mitad):
20, 10, 5, 2, 1,
```

Ilustración 51: Ciclo while

Ciclo do...while

Su sintaxis es:

```
do {
```

```
} while (condición para que se mantenga en el ciclo)
```

A/045.cs

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            int Contador;

            //Ciclo do-while ascendente
            Console.WriteLine("Ciclo ascendente:");
            Contador = 1;
            do {
                Console.Write(Contador.ToString() + ", ");
                Contador++;
            } while (Contador <= 20);

            //Ciclo do-while descendente
            Console.WriteLine("\r\nCiclo descendente:");
            Contador = 20;
            do {
                Console.Write(Contador.ToString() + ", ");
                Contador--;
            } while (Contador >= 1);

            //Ciclo do-while ascendente, avance de 2 en 2
            Console.WriteLine("\r\nCiclo ascendente (avance de 2 en 2):");
            Contador = 1;
            do {
                Console.Write(Contador.ToString() + ", ");
                Contador += 2;
            } while (Contador <= 20);

            //Ciclo do-while descendente, retrocede de 2 en 2
            Console.WriteLine("\r\nCiclo descendente (retrocede de 2 en 2):");
            Contador = 20;
            do {
                Console.Write(Contador.ToString() + ", ");
                Contador -= 2;
            } while (Contador >= 1);

            //Ciclo do-while ascendente, avance doble
            Console.WriteLine("\r\nCiclo ascendente (avance doble):");
            Contador = 1;
            do {
                Console.Write(Contador.ToString() + ", ");
                Contador *= 2;
            } while (Contador <= 20);

            //Ciclo do-while descendente, retrocede la mitad
            Console.WriteLine("\r\nCiclo descendente (retrocede la mitad):");
            Contador = 20;
            do {
                Console.Write(Contador.ToString() + ", ");
                Contador /= 2;
            } while (Contador >= 1);
        }
    }
}
```

```
Consola de depuración de Mi... X + ▾ - □ X
Ciclo ascendente:
1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20,
Ciclo descendente:
20, 19, 18, 17, 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1,
Ciclo ascendente (avance de 2 en 2):
1, 3, 5, 7, 9, 11, 13, 15, 17, 19,
Ciclo descendente (retrocede de 2 en 2):
20, 18, 16, 14, 12, 10, 8, 6, 4, 2,
Ciclo ascendente (avance doble):
1, 2, 4, 8, 16,
Ciclo descendente (retrocede la mitad):
20, 10, 5, 2, 1,
```

Ilustración 52: Ciclo do...while

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            int Contador;

            //Rompe el ciclo con break
            Console.WriteLine("Ciclo ascendente:");
            Contador = 1;
            do {
                //Si Contador es múltiplo de 13 se rompe el ciclo con break
                if (Contador % 13 == 0) break;
                Console.Write(Contador.ToString() + ", ");
                Contador++;
            } while (Contador <= 20);
        }
    }
}
```

Consola de depuración d... X + | - □ ×

Ciclo ascendente:

1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12,

Ilustración 53: Romper ciclo con break

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            int Contador;

            Console.WriteLine("Ciclo ascendente:");
            Contador = 0;
            do {
                Contador++;

                //Si Contador es par entonces va a la siguiente iteración, no
                //ejecuta lo que está después.
                if (Contador % 2 == 0) continue;

                Console.Write(Contador.ToString() + ", ");
            } while (Contador <= 20);
        }
    }
}
```



```
Ciclo ascendente:
1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21,
```

Ilustración 54: Uso del continue en ciclos

```
namespace Ejemplo {
    internal class Program {
        static void Main() {

            //Ciclos anidados simulando un minutero y un segundero
            for (int minuto = 0; minuto <= 10; minuto++) {
                for (int segundo = 0; segundo < 60; segundo++) {
                    Console.WriteLine(minuto.ToString() + ":" + segundo.ToString());
                }
            }
        }
    }
}
```

The screenshot shows a Windows Command Prompt window titled "Consola de depuración". The window has standard minimize, maximize, and close buttons at the top. The text area contains the following output from the C# program:

```
0:0
0:1
0:2
0:3
0:4
0:5
0:6
0:7
0:8
0:9
0:10
0:11
0:12
0:13
0:14
```

Ilustración 55: Ciclos anidados

```
namespace Ejemplo {
    internal class Program {
        static void Main() {

            for (int numA = 1; numA <= 3; numA++) {
                for (int numB = 1; numB <= 6; numB++) {
                    for (int numC = 1; numC <= 9; numC++) {
                        Console.WriteLine(numA.ToString() + " | " + numB.ToString() + " | " +
numC.ToString());
                        if (numC == 5) break; //Sólo rompe el ciclo más interno
                    }
                }
                Console.WriteLine("Final");
            }
        }
    }
}
```

```
1|1|1
1|1|2
1|1|3
1|1|4
1|1|5
1|2|1
1|2|2
1|2|3
1|2|4
1|2|5
1|3|1
1|3|2
```

Ilustración 56: Rompiendo ciclos anidados

```
namespace Ejemplo {
    internal class Program {
        static void Main() {

            for (int numA = 1; numA <= 3; numA++) {
                for (int numB = 1; numB <= 6; numB++) {
                    for (int numC = 1; numC <= 9; numC++) {
                        Console.WriteLine(numA.ToString() + " | " + numB.ToString() + " | " +
numC.ToString());
                        if (numC == 5) goto afuera; //Sale de todos los ciclos
                    }
                }
            }

            afuera: Console.WriteLine("Final");
        }
    }
}
```

```
1|1|1
1|1|2
1|1|3
1|1|4
1|1|5
Final
```

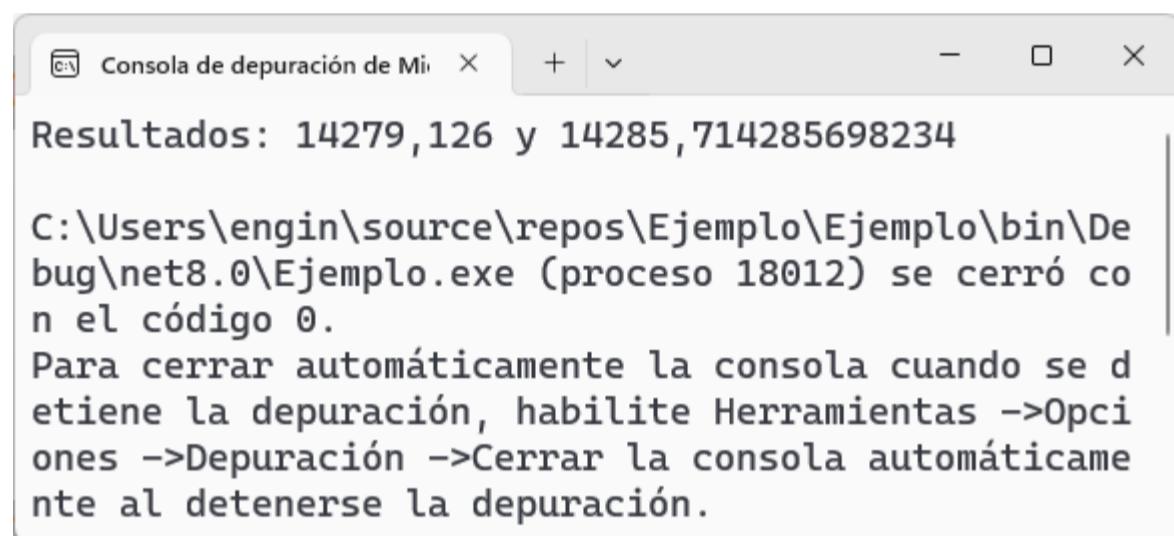
Ilustración 57: Uso del goto

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Diferencias en precisión entre float y double

            float acumA = 0;
            float divideA = 7;
            for (float numA = 1; numA <= 100000; numA++) {
                acumA += 1 / divideA;
            }

            double acumB = 0;
            double divideB = 7;
            for (double numB = 1; numB <= 100000; numB++) {
                acumB += 1 / divideB;
            }

            Console.WriteLine("Resultados: " + acumA.ToString() + " y " + acumB.ToString());
        }
    }
}
```



```
Consola de depuración de Mi... + - X
Resultados: 14279,126 y 14285,714285698234

C:\Users\engin\source\repos\Ejemplo\Ejemplo\bin\De
bug\net8.0\Ejemplo.exe (proceso 18012) se cerró co
n el código 0.
Para cerrar automáticamente la consola cuando se d
eteiene la depuración, habilite Herramientas ->Opc
iones ->Depuración ->Cerrar la consola automáticame
nte al detenerse la depuración.
```

Ilustración 58: Diferencias de precisión entre float y double

Muy recomendado: usar double.

```

namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Uso de funciones
            double ladoA = 3;
            double ladoB = 4;
            double ladoC = 5;
            double Area = AreaTrianguloHeron(ladoA, ladoB, ladoC);
            Console.WriteLine("Área triángulo es: " + Area.ToString());
        }

        //Fórmula de Herón para el cálculo del área de un triángulo dado los lados
        static double AreaTrianguloHeron(double valA, double valB, double valC) {
            double s = (valA + valB + valC) / 2;
            double area = Math.Sqrt(s * (s - valA) * (s - valB) * (s - valC));
            return area;
        }
    }
}

```

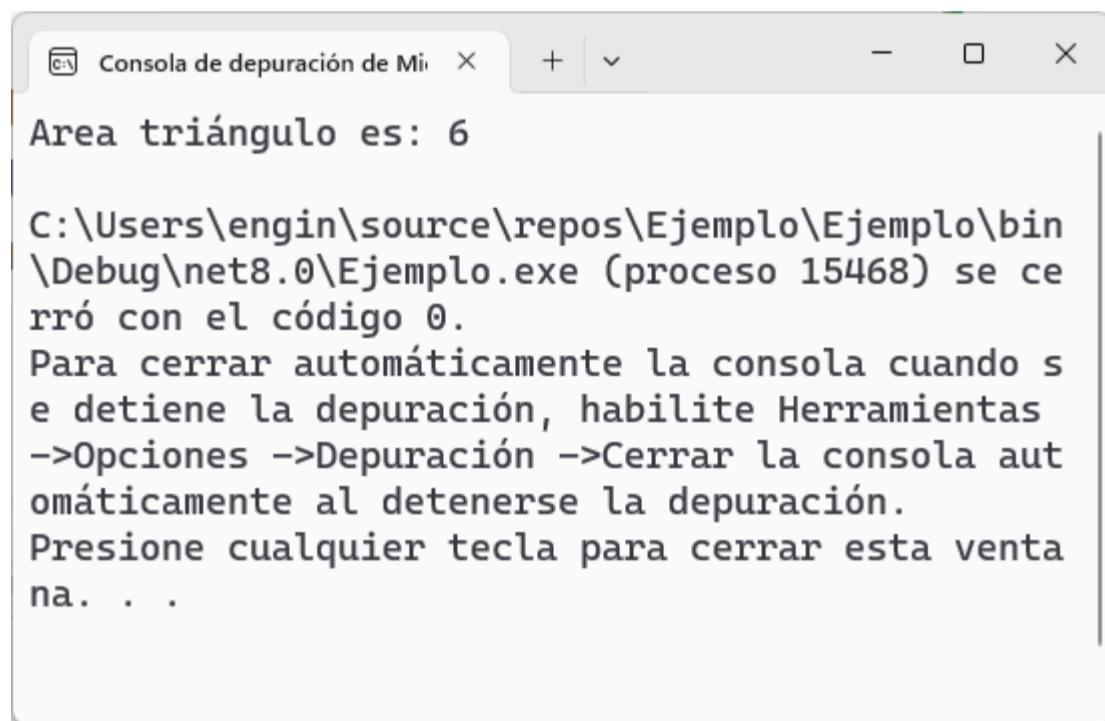


Ilustración 60: Funciones

```

namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Uso de funciones
            double ladoA = 3;
            double ladoB = 4;
            double ladoC = 5;
            bool Posible = TrianguloPosible(ladoA, ladoB, ladoC);
            Console.WriteLine("Posibilidad del triángulo: " + Posible.ToString());
        }

        //Retorna true si el triángulo es posible
        static bool TrianguloPosible(double valA, double valB, double valC) {
            return valA + valB >= valC && valA + valC >= valB && valB + valC >= valA;
        }
    }
}

```



Ilustración 60: Función que retorna dato de tipo bool

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Función con doble salida.
            double radio = 1;
            double perimetro, area;
            area = DatosCirculo(radio, out perimetro);
            Console.WriteLine("Área del círculo es: " + area.ToString());
            Console.WriteLine("Perímetro del círculo es: " + perimetro.ToString());
        }

        //Retorna dos valores: el área y el perímetro del círculo
        static double DatosCirculo(double Radio, out double Perímetro) {
            double area = Math.PI * Radio * Radio;
            Perímetro = 2 * Math.PI * Radio;
            return area;
        }
    }
}
```

```
Consola de depuración de Mi... + - X
Area del círculo es: 3,141592653589793
Perímetro del círculo es: 6,283185307179586

C:\Users\engin\source\repos\Ejemplo\Ejemplo\bin\Debug\net8.0\Ejemplo.exe (proceso 13224) se cerró con el código
0.

Para cerrar automáticamente la consola cuando se detiene
la depuración, habilite Herramientas ->Opciones ->Depuración ->Cerrar la consola automáticamente al detenerse l
a depuración.
```

Ilustración 61: Funciones con doble salida

```

namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Función con doble salida. Nuevo en C# 7.0
            double radio = 1;
            double perimetro, area;
            DatosCirculo(radio, out perimetro, out area);
            Console.WriteLine("Área del círculo es: " + area.ToString());
            Console.WriteLine("Perímetro del círculo es: " + perimetro.ToString());
        }

        //Retorna dos valores: el área y el perímetro del círculo
        static void DatosCirculo(double Radio, out double Perimetro, out double Area) {
            Area = Math.PI * Radio * Radio;
            Perimetro = 2 * Math.PI * Radio;
        }
    }
}

```

Consola de depuración de Mi... + - X

Perímetro del círculo es: 6,283185307179586

C:\Users\engin\source\repos\Ejemplo\Ejemplo\bin\Debug\net8.0\Ejemplo.exe (proceso 17316) se cerró con el código 0.

Para cerrar automáticamente la consola cuando se detiene la depuración, habilite Herramientas ->Opciones ->Depuración ->Cerrar la consola automáticamente al detenerse la depuración.

Presione cualquier tecla para cerrar esta ventana. .

Ilustración 62: Funciones con doble salida

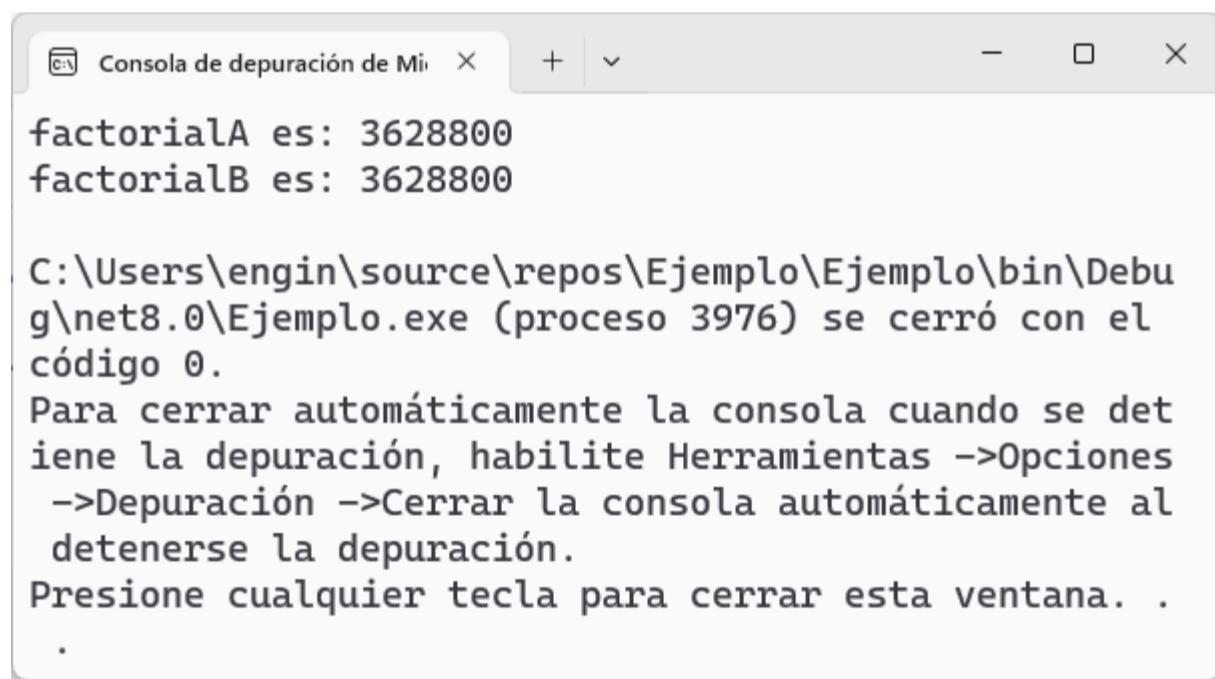
```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Función iterativa y recursiva
            long valor = 10;
            long factorialA, factorialB;

            factorialA = CalculaFactorialIterativa(valor);
            factorialB = CalculaFactorialRecursivo(valor);

            Console.WriteLine("factorialA es: " + factorialA.ToString());
            Console.WriteLine("factorialB es: " + factorialB.ToString());
        }

        //Retorna el factorial de un número, de forma iterativa
        static long CalculaFactorialIterativa(long numero) {
            long resultado = 1;
            for (long num=2; num <= numero; num++) {
                resultado *= num;
            }
            return resultado;
        }

        //Retorna el factorial de un número, de forma recursiva
        static long CalculaFactorialRecursivo(long numero) {
            if (numero == 1) return 1;
            return numero*CalculaFactorialRecursivo(numero-1);
        }
    }
}
```



```
factorialA es: 3628800
factorialB es: 3628800

C:\Users\engin\source\repos\Ejemplo\Ejemplo\bin\Debug\net8.0\Ejemplo.exe (proceso 3976) se cerró con el
código 0.

Para cerrar automáticamente la consola cuando se detiene la depuración, habilite Herramientas ->Opciones
->Depuración ->Cerrar la consola automáticamente al
detenerse la depuración.

Presione cualquier tecla para cerrar esta ventana. .
```

Ilustración 63: Funciones iterativas y recursivas

```
namespace Ejemplo {
    internal class Program {
        static int valor; //Una variable que es conocida por todas las funciones

        static void Main() {
            //Ámbito de las variables
            valor = 17;
            Console.WriteLine("Valor al iniciar es: " + valor.ToString());
            UnProcedimiento();
            Console.WriteLine("Valor después de primer procedimiento es: " + valor.ToString());
            OtroProcedimiento();
            Console.WriteLine("Valor después de segundo procedimiento es: " + valor.ToString());
        }

        //Un procedimiento
        static void UnProcedimiento() {
            valor = 590;
        }

        //Otro procedimiento
        static void OtroProcedimiento() {
            valor = 3246;
        }
    }
}
```

```
Consola de depuración de Mi... + X - □ ×

Valor al iniciar es: 17
Valor después de primer procedimiento es: 590
Valor después de segundo procedimiento es: 3246

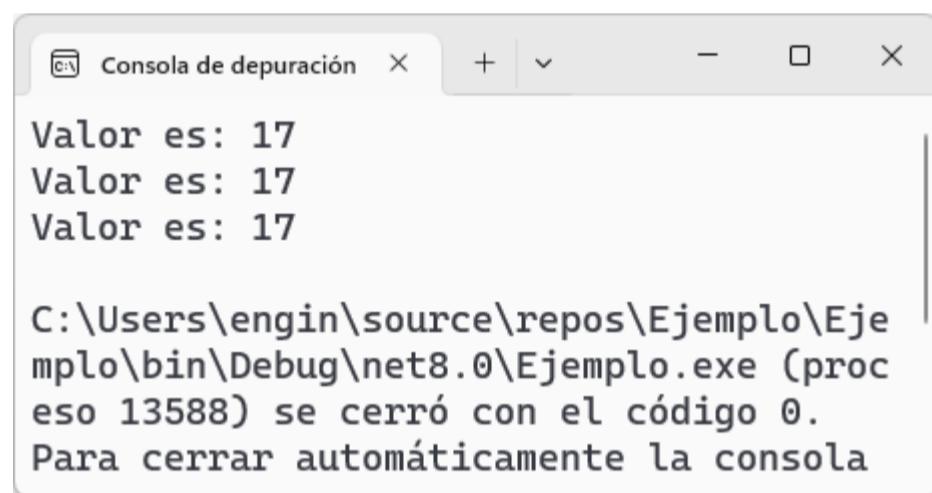
C:\Users\engin\source\repos\Ejemplo\Ejemplo\bin\Debug\net8.0
\Ejemplo.exe (proceso 17572) se cerró con el código 0.
Para cerrar automáticamente la consola cuando se detiene la
depuración, habilite Herramientas ->Opciones ->Depuración ->
Cerrar la consola automáticamente al detenerse la depuración
```

Ilustración 64: Ámbito de las variables

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Ámbito de las variables. Variable local
            int valor = 17;
            Console.WriteLine("Valor es: " + valor.ToString());
            UnProcedimiento();
            Console.WriteLine("Valor es: " + valor.ToString());
            OtroProcedimiento();
            Console.WriteLine("Valor es: " + valor.ToString());
        }

        //Un procedimiento. La variable "valor" es otra distinta a la del "Main"
        static void UnProcedimiento() {
            int valor = 590;
        }

        //Otro procedimiento. La variable "valor" es otra distinta a la del "Main"
        static void OtroProcedimiento() {
            int valor = 3246;
        }
    }
}
```



```
Consola de depuración  X  +  -  □  X
Valor es: 17
Valor es: 17
Valor es: 17

C:\Users\engin\source\repos\Ejemplo\Eje
mple\bin\Debug\net8.0\Ejemplo.exe (proc
eso 13588) se cerró con el código 0.
Para cerrar automáticamente la consola
```

Ilustración 65: Ámbito de las variables

```

namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Funciones de números
            int Numero = 16832929;
            Console.WriteLine(Numero.ToString() + " es impar: " + EsImpar(Numero).ToString());
            Console.WriteLine(Numero.ToString() + " es par: " + EsPar(Numero).ToString());
            Console.WriteLine(Numero.ToString() + " total de cifras: " +
TotalCifras(Numero).ToString());
            Console.WriteLine(Numero.ToString() + " sus dos últimas cifras: " +
RetornadosultimasCifras(Numero).ToString());
            Console.WriteLine(Numero.ToString() + " su antepenúltima cifra: " +
AntepenultimaCifra(Numero).ToString());
            Console.WriteLine(Numero.ToString() + " su penúltima cifra: " +
PenultimaCifra(Numero).ToString());
            Console.WriteLine(Numero.ToString() + " su última cifra: " +
UltimaCifra(Numero).ToString());
            Console.WriteLine(Numero.ToString() + " la cifra más alta: " +
LaCifraMasAlta(Numero).ToString());
            Console.WriteLine(Numero.ToString() + " la cifra más baja: " +
LaCifraMasBaja(Numero).ToString());
            Console.WriteLine(Numero.ToString() + " total de cifras iguales a 5 es: " +
Cifrashalladas(Numero, 5).ToString());
            Console.WriteLine(Numero.ToString() + " al invertirlo es: " +
InvierteNumero(Numero).ToString());
            Console.WriteLine(Numero.ToString() + " es palíndromo: " + EsPalindromo(Numero).ToString());
            Console.WriteLine(Numero.ToString() + " tercera cifra es: " + CifraPosicion(Numero,
3).ToString());
            Console.WriteLine(Numero.ToString() + " primera cifra es: " +
PrimeraCifra(Numero).ToString());
            Console.WriteLine(Numero.ToString() + " la suma de las cifras es: " +
SumaCifras(Numero).ToString());
            Console.WriteLine(Numero.ToString() + " la suma de las cifras pares es: " +
SumaCifrasPares(Numero).ToString());
            Console.WriteLine(Numero.ToString() + " la suma de las cifras impares es: " +
SumaCifrasImpares(Numero).ToString());
            Console.WriteLine(Numero.ToString() + " la multiplicación de las cifras es: " +
MultiplicaCifras(Numero).ToString());
            Console.WriteLine(Numero.ToString() + " la multiplicación de las cifras pares es: " +
MultiplicaCifrasPares(Numero).ToString());
            Console.WriteLine(Numero.ToString() + " la multiplicación de las cifras impares es: " +
MultiplicaCifrasImpares(Numero).ToString());
            Console.WriteLine(Numero.ToString() + " todas las cifras son pares: " +
TodasCifrasPares(Numero).ToString());
            Console.WriteLine(Numero.ToString() + " todas las cifras son impares: " +
TodasCifrasImpares(Numero).ToString());
            Console.WriteLine(Numero.ToString() + " total cifras pares: " +
TotalCifrasPares(Numero).ToString());
            Console.WriteLine(Numero.ToString() + " total cifras impares: " +
TotalCifrasImpares(Numero).ToString());
            Console.WriteLine(Numero.ToString() + " solo hay cifras menores o iguales a 5: " +
SoloCifrasMenorIgual(Numero, 5).ToString());
            Console.WriteLine(Numero.ToString() + " solo hay cifras mayores o iguales a 5: " +
SoloCifrasMayorIgual(Numero, 5).ToString());
            Console.WriteLine(Numero.ToString() + " usa distintas cifras: " +
DistintasCifras(Numero).ToString());
            Console.WriteLine(Numero.ToString() + " usa distintas cifras pares: " +
DistintasCifrasPares(Numero).ToString());
            Console.WriteLine(Numero.ToString() + " usa distintas cifras impares: " +
DistintasCifrasImPares(Numero).ToString());
            Console.WriteLine(Numero.ToString() + " extrayendo cifras pares: " +
NumeroSoloPares(Numero).ToString());
            Console.WriteLine(Numero.ToString() + " extrayendo cifras impares: " +
NumeroSoloImpares(Numero).ToString());
        }

        // Retorna true si un número es impar
        static bool EsImpar(int Numero) {
            return Numero % 2 == 1;
        }

        // Retorna true si un número es par
        static bool EsPar(int Numero) {
            return Numero % 2 == 0;
        }

        // Retorna el número de Cifras de un número
    }
}

```

```

static int TotalCifras(int Numero) {
    int Copia = Numero;
    int Cuenta = 0;
    while (Copia != 0) {
        Cuenta++;
        Copia /= 10;
    }
    return Cuenta;
}

// Retorna las dos últimas Cifras del número
static int RetornadosultimasCifras(int Numero) {
    return Numero % 100;
}

// Retorna la antepenúltima Cifra de un número entero
static int AntepenultimaCifra(int Numero) {
    return (Numero / 100) % 10;
}

// Retorna la penúltima Cifra de un número entero
static int PenultimaCifra(int Numero) {
    return (Numero / 10) % 10;
}

// Retorna la última Cifra de un número entero
static int UltimaCifra(int Numero) {
    return Numero % 10;
}

// Retorna la Cifra más alta
static int LaCifraMasAlta(int Numero) {
    int Copia = Numero;
    int Cifra = 0;
    while (Copia != 0) {
        if (Copia % 10 > Cifra) Cifra = Copia % 10;
        Copia /= 10;
    }
    return Cifra;
}

// Retorna la Cifra más baja
static int LaCifraMasBaja(int Numero) {
    if (Numero == 0) return 0;
    int Copia = Numero;
    int Cifra = 9;
    while (Copia != 0) {
        if (Copia % 10 < Cifra) Cifra = Copia % 10;
        Copia /= 10;
    }
    return Cifra;
}

// Dice cuántas Cifras de determinado número hay en el número enviado
static int Cifrashalladas(int Numero, int Cifra) {
    int Copia = Numero;
    int Acumula = 0;
    while (Copia != 0) {
        if (Copia % 10 == Cifra) Acumula++;
        Copia /= 10;
    }
    return Acumula;
}

// Invierte un número
static int InvierteNumero(int Numero) {
    int Copia = Numero;
    int Multiplica = (int) Math.Pow(10, TotalCifras(Copia) - 1);
    int Acumula = 0;
    while (Copia != 0) {
        int Cifra = Copia % 10;
        Acumula += Cifra * Multiplica;
        Multiplica /= 10;
        Copia /= 10;
    }
    return Acumula;
}

// Retorna true si el número es palíndromo
static bool EsPalindromo(int Numero) {

```

```

        if (Numero == InvierteNumero(Numero)) return true;
        return false;
    }

    // Retorna la Cifra de una determinada posición
    static int CifraPosicion(int Numero, int Posicion) {
        int Copia = InvierteNumero(Numero);
        int Pos = 1;
        while (Copia != 0) {
            int Cifra = Copia % 10;
            if (Pos == Posicion) return Cifra;
            Copia /= 10;
            Pos++;
        }
        return 0;
    }

    // Retorna la primera Cifra de un número
    static int PrimeraCifra(int Numero) {
        return CifraPosicion(Numero, 1);
    }

    // Retorna la sumatoria de las Cifras de un número
    static int SumaCifras(int Numero) {
        int Copia = Numero;
        int Acumula = 0;
        while (Copia != 0) {
            int Cifra = Copia % 10;
            Acumula += Cifra;
            Copia /= 10;
        }
        return Acumula;
    }

    // Retorna la sumatoria de las Cifras pares de un número
    static int SumaCifrasPares(int Numero) {
        int Copia = Numero;
        int Acumula = 0;
        while (Copia != 0) {
            int Cifra = Copia % 10;
            if (Cifra % 2 == 0) Acumula += Cifra;
            Copia /= 10;
        }
        return Acumula;
    }

    // Retorna la sumatoria de las Cifras impares de un número
    static int SumaCifrasImpares(int Numero) {
        int Copia = Numero;
        int Acumula = 0;
        while (Copia != 0) {
            int Cifra = Copia % 10;
            if (Cifra % 2 != 0) Acumula += Cifra;
            Copia /= 10;
        }
        return Acumula;
    }

    // Retorna el producto de las Cifras de un número
    static int MultiplicaCifras(int Numero) {
        if (Numero == 0) return 0;
        int Copia = Numero;
        int Acumula = 1;
        while (Copia != 0) {
            int Cifra = Copia % 10;
            Acumula *= Cifra;
            Copia /= 10;
        }
        return Acumula;
    }

    // Retorna el producto de las Cifras pares de un número
    static int MultiplicaCifrasPares(int Numero) {
        int Copia = Numero;
        int Acumula = 1;
        bool HayPar = false;
        while (Copia != 0) {
            int Cifra = Copia % 10;
            if (Cifra % 2 == 0) {
                Acumula *= Cifra;
            }
        }
        return Acumula;
    }
}

```

```

        HayPar = true;
    }
    Copia /= 10;
}
if (HayPar) return Acumula;
return 0;
}

// Retorna el producto de las Cifras impares de un número
static int MultiplicaCifrasImpares(int Numero) {
    int Copia = Numero;
    int Acumula = 1;
    bool HayImpar = false;
    while (Copia != 0) {
        int Cifra = Copia % 10;
        if (Cifra % 2 != 0) {
            Acumula *= Cifra;
            HayImpar = true;
        }
        Copia /= 10;
    }
    if (HayImpar) return Acumula;
    return 0;
}

// Retorna true si todas las Cifras son pares
static bool TodasCifrasPares(int Numero) {
    int Copia = Numero;
    while (Copia != 0) {
        int Cifra = Copia % 10;
        if (Cifra % 2 != 0) return false;
        Copia /= 10;
    }
    return true;
}

// Retorna true si todas las Cifras son impares
static bool TodasCifrasImpares(int Numero) {
    int Copia = Numero;
    if (Copia == 0) return false;
    while (Copia != 0) {
        int Cifra = Copia % 10;
        if (Cifra % 2 == 0) return false;
        Copia /= 10;
    }
    return true;
}

// Retorna el número de Cifras pares
static int TotalCifrasPares(int Numero) {
    int Copia = Numero;
    int Acumula = 0;
    while (Copia != 0) {
        int Cifra = Copia % 10;
        if (Cifra % 2 == 0) Acumula++;
        Copia /= 10;
    }
    return Acumula;
}

// Retorna el número de Cifras impares
static int TotalCifrasImpares(int Numero) {
    int Copia = Numero;
    int Acumula = 0;
    while (Copia != 0) {
        int Cifra = Copia % 10;
        if (Cifra % 2 != 0) Acumula++;
        Copia /= 10;
    }
    return Acumula;
}

// Retorna true si el número tiene sólo Cifras menores o iguales a Cifra
static bool SoloCifrasMenorIgual(int Numero, int Cifra) {
    int Copia = Numero;
    while (Copia != 0) {
        if (Copia % 10 > Cifra) return false;
        Copia /= 10;
    }
    return true;
}

```

```

}

// Retorna true si el número tiene sólo Cifras mayores o iguales a Cifra
static bool SoloCifrasMayorIgual(int Numero, int Cifra) {
    int Copia = Numero;
    while (Copia != 0) {
        if (Copia % 10 < Cifra) return false;
        Copia /= 10;
    }
    return true;
}

// Retorna true si todas las Cifras son distintas
static bool DistintasCifras(int Numero) {
    for (int Cifra = 0; Cifra <= 9; Cifra++) {
        int Copia = Numero;
        int Cuenta = 0;
        while (Copia != 0) {
            if (Copia % 10 == Cifra) Cuenta++;
            if (Cuenta > 1) return false;
            Copia /= 10;
        }
    }
    return true;
}

// Retorna si todas las Cifras pares son distintas
static bool DistintasCifrasPares(int Numero) {
    for (int Cifra = 0; Cifra <= 8; Cifra += 2) {
        int Copia = Numero;
        int Cuenta = 0;
        while (Copia != 0) {
            if (Copia % 10 == Cifra) Cuenta++;
            if (Cuenta > 1) return false;
            Copia /= 10;
        }
    }
    return true;
}

// Retorna si todas las Cifras impares son distintas
static bool DistintasCifrasImPares(int Numero) {
    for (int Cifra = 1; Cifra <= 9; Cifra += 2) {
        int Copia = Numero;
        int Cuenta = 0;
        while (Copia != 0) {
            if (Copia % 10 == Cifra) Cuenta++;
            if (Cuenta > 1) return false;
            Copia /= 10;
        }
    }
    return true;
}

//Retorna un número con solo las Cifras pares
static int NumeroSoloPares(int Numero) {
    int Copia = Numero;
    int Acumula = 0;
    int Posicion = 1;
    while (Copia != 0) {
        int Cifra = Copia % 10;
        if (Cifra % 2 == 0) {
            Acumula += Posicion * Cifra;
            Posicion *= 10;
        }
        Copia /= 10;
    }
    return Acumula;
}

//Retorna un número con solo las Cifras impares
static int NumeroSoloImpares(int Numero) {
    int Copia = Numero;
    int Acumula = 0;
    int Posicion = 1;
    while (Copia != 0) {
        int Cifra = Copia % 10;
        if (Cifra % 2 != 0) {
            Acumula += Posicion * Cifra;
            Posicion *= 10;
        }
        Copia /= 10;
    }
    return Acumula;
}

```

```
        }
        Copia /= 10;
    }
    return Acumula;
}
}
```

The screenshot shows a Windows Command Prompt window titled "Consola de depuración de Mi". The window contains the following text output:

```
16832929 es impar: True
16832929 es par: False
16832929 total de cifras: 8
16832929 sus dos últimas cifras: 29
16832929 su antepenúltima cifra: 9
16832929 su penúltima cifra: 2
16832929 su última cifra: 9
16832929 la cifra más alta: 9
16832929 la cifra más baja: 1
16832929 total de cifras iguales a 5 es: 0
16832929 al invertirlo es: 92923861
16832929 es palíndromo: False
16832929 tercera cifra es: 8
16832929 primera cifra es: 1
16832929 la suma de las cifras es: 40
16832929 la suma de las cifras pares es: 18
16832929 la suma de las cifras impares es: 22
16832929 la multiplicación de las cifras es: 46656
16832929 la multiplicación de las cifras pares es: 192
16832929 la multiplicación de las cifras impares es: 243
16832929 todas las cifras son pares: False
16832929 todas las cifras son impares: False
16832929 total cifras pares: 4
16832929 total cifras impares: 4
16832929 solo hay cifras menores o iguales a 5: False
16832929 solo hay cifras mayores o iguales a 5: False
16832929 usa distintas cifras: False
16832929 usa distintas cifras pares: False
16832929 usa distintas cifras impares: False
16832929 extrayendo cifras pares: 6822
16832929 extrayendo cifras impares: 1399
```

Ilustración 66: Manejo de cifras

Parte 2: Cadenas

Las cadenas (strings) son una de las estructuras con gran uso en la informática y de gran importancia (por ejemplo, el código HTML, que es una cadena, es interpretado por el navegador y de esa forma el usuario final ve una página web). En C# hay un rico conjunto de funciones para manipular cadenas. La primera letra de una cadena empieza en la posición cero (0). Si envía una cadena a una función como parámetro y la función cambia ese parámetro, la cadena NO cambia desde donde fue llamada la función, es decir, es un paso por valor.

Declaración de cadenas

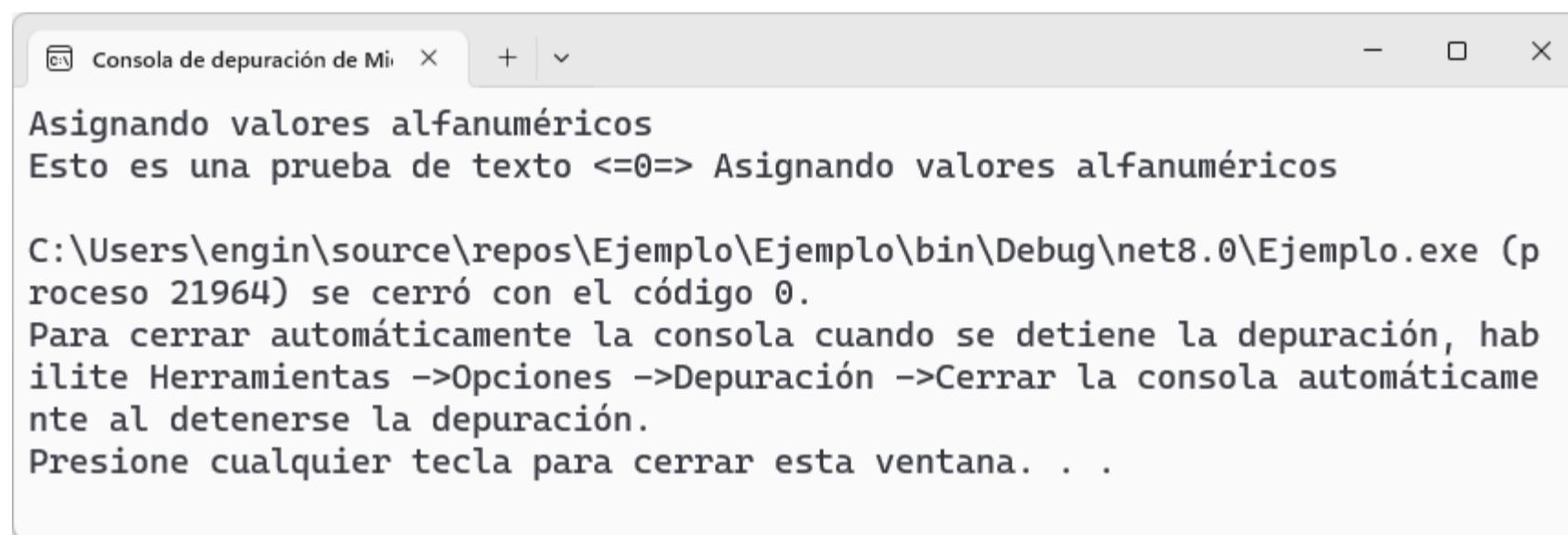
B/001.cs

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Declarar variable de tipo cadena (string). Dos formas.
            System.String cadenaA;
            string cadenaB;

            //Asignando valores de cadena
            cadenaA = "Esto es una prueba de texto";
            cadenaB = "Asignando valores alfanuméricos";

            //Imprimiendo por consola
            Console.WriteLine(cadenaA);
            Console.WriteLine(cadenaB);

            //Uniendo dos cadenas o concatenar
            string cadenaC = cadenaA + " <=> " + cadenaB;
            Console.WriteLine(cadenaC);
        }
    }
}
```



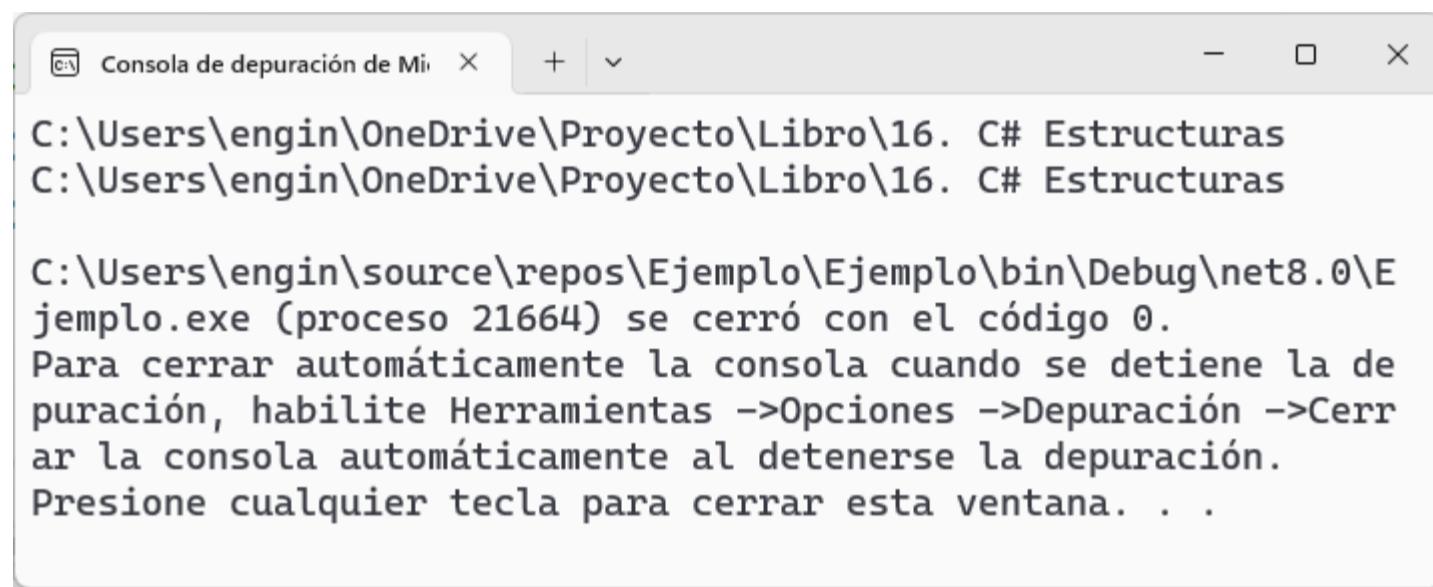
```
Consola de depuración de Mi... + v - □ ×
Asignando valores alfanuméricos
Esto es una prueba de texto <=> Asignando valores alfanuméricos
C:\Users\engin\source\repos\Ejemplo\Ejemplo\bin\Debug\net8.0\Ejemplo.exe (p
roceso 21964) se cerró con el código 0.
Para cerrar automáticamente la consola cuando se detiene la depuración, hab
ilite Herramientas ->Opciones ->Depuración ->Cerrar la consola automáticamente al detenerse la depuración.
Presione cualquier tecla para cerrar esta ventana. . .
```

Ilustración 67: Declaración de cadenas

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Rutas. Forma antigua
            string RutaA = "C:\\\\Users\\\\engin\\\\OneDrive\\\\Proyecto\\\\Libro\\\\16. C# Estructuras";

            //Ruta. Forma moderna
            string RutaB = @"C:\\Users\\engin\\OneDrive\\Proyecto\\Libro\\16. C# Estructuras";

            //Imprimiendo por consola
            Console.WriteLine(RutaA);
            Console.WriteLine(RutaB);
        }
    }
}
```



The screenshot shows a Windows Task Manager window titled 'Consola de depuración de Mi'. The window contains the following text:

```
C:\\\\Users\\\\engin\\\\OneDrive\\\\Proyecto\\\\Libro\\\\16. C# Estructuras
C:\\\\Users\\\\engin\\\\OneDrive\\\\Proyecto\\\\Libro\\\\16. C# Estructuras

C:\\\\Users\\\\engin\\\\source\\\\repos\\\\Ejemplo\\\\bin\\\\Debug\\\\net8.0\\\\E
jemplo.exe (proceso 21664) se cerró con el código 0.
Para cerrar automáticamente la consola cuando se detiene la de
puración, habilite Herramientas ->Opciones ->Depuración ->Cerr
ar la consola automáticamente al detenerse la depuración.
Presione cualquier tecla para cerrar esta ventana. . .
```

Ilustración 68: Uso de @

Constantes con cadenas

Las constantes una vez definidas con valores, no se pueden cambiar.

B/003.cs

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Constante
            const string cadena = "abcdefghijklm";
            //Se intenta cambiar la constante y da error en compilación
            cadena = "Hola Mundo";
            //Imprimiendo por consola
            Console.WriteLine(cadena);
        }
    }
}
```

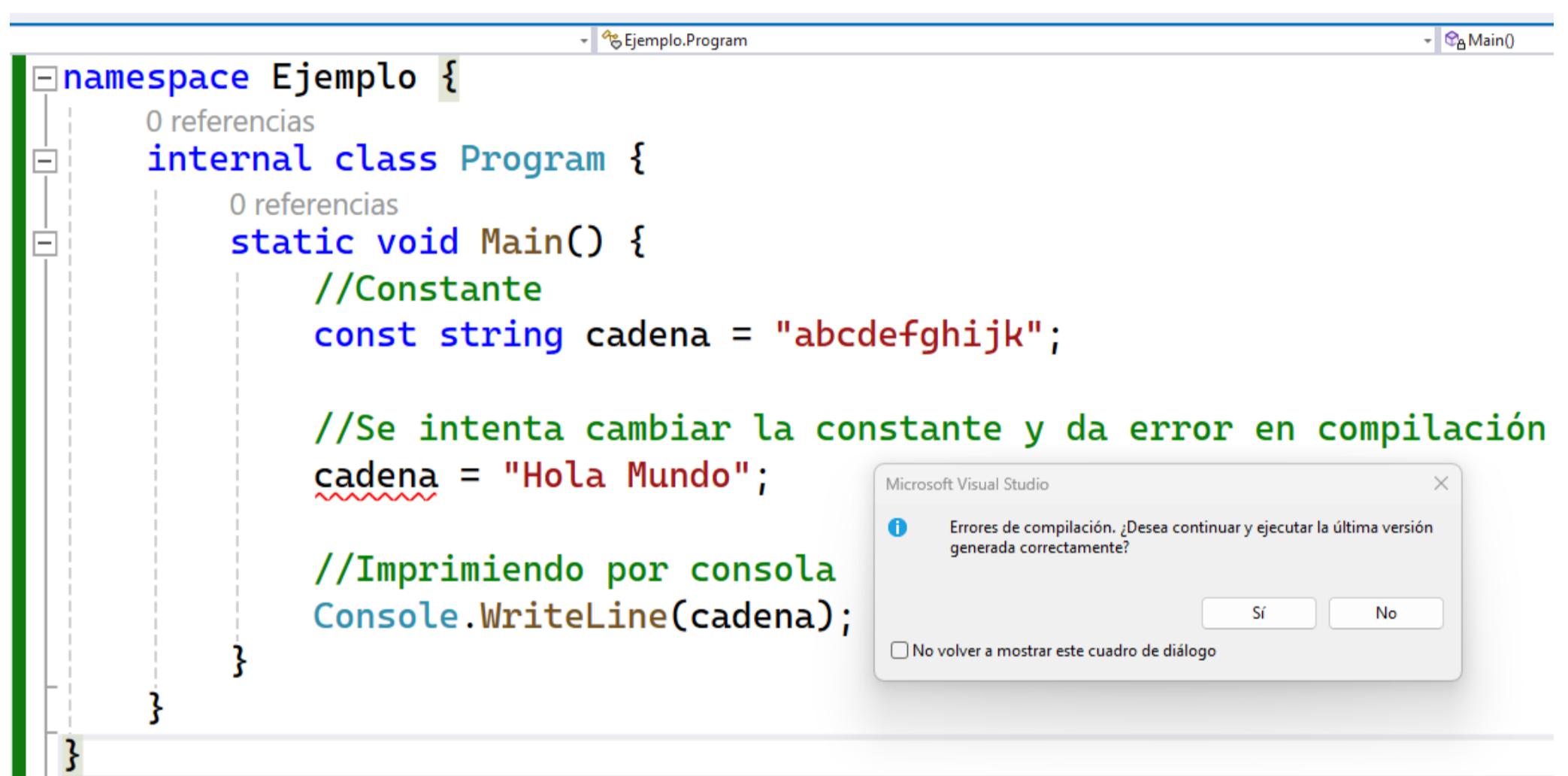


Ilustración 69: No se puede cambiar una constante

Copia de cadenas

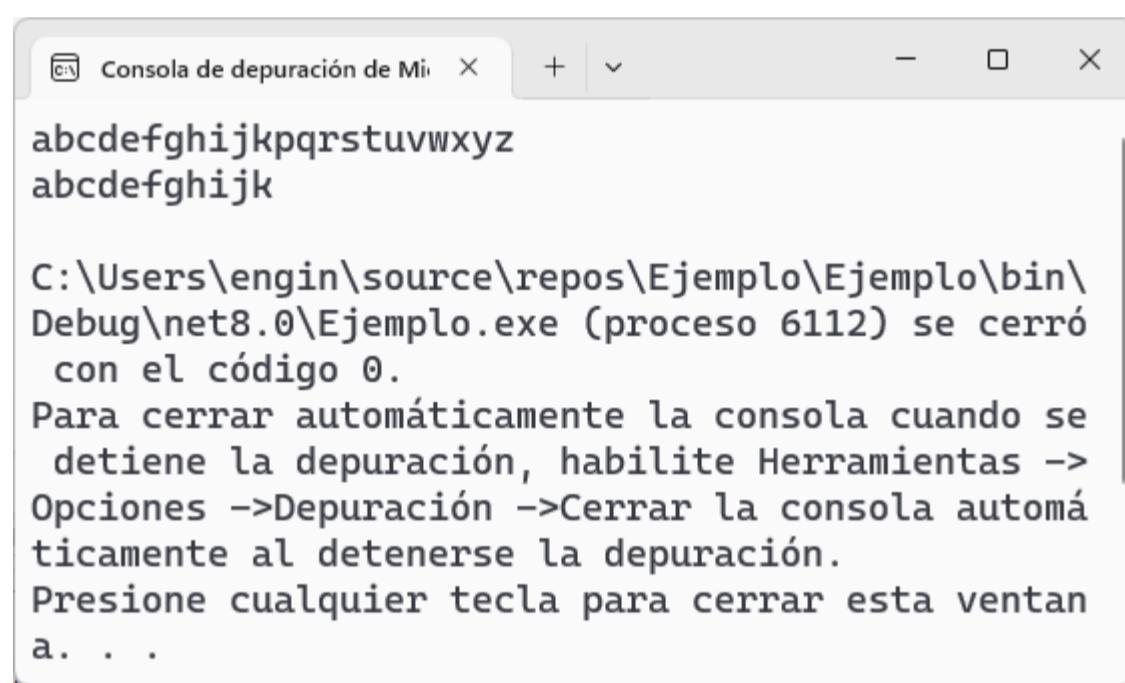
C# trata los strings como si fuesen un tipo de dato nativo en la práctica, por eso se copia el contenido de una variable a otra al usarse el operador de asignación (=).

B/004.cs

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Inmutabilidad
            string cadenaA = "abcdefghijkl";
            string cadenaB = cadenaA; //Se copian los datos de cadenaA en cadenaB

            //Se agregan datos a cadenaA ¿Qué sucederá con cadenaB?
            cadenaA += "uvwxyz";

            //Imprimiendo por consola
            Console.WriteLine(cadenaA);
            Console.WriteLine(cadenaB);
        }
    }
}
```



```
abcdehijkpqrstuvwxyz
abcdefghijk

C:\Users\engin\source\repos\Ejemplo\Ejemplo\bin\Debug\net8.0\Ejemplo.exe (proceso 6112) se cerró
con el código 0.
Para cerrar automáticamente la consola cuando se
detiene la depuración, habilite Herramientas ->
Opciones ->Depuración ->Cerrar la consola automá
ticamente al detenerse la depuración.
Presione cualquier tecla para cerrar esta ventan
a. . .
```

Ilustración 70: Copia de cadenas

Caracteres especiales

Los caracteres especiales no se imprimen, sino que generan otro comportamiento al intentar "imprimirse".

B/005.cs

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Caracteres especiales. Salto de línea
            string cadenaA = "Este es un salto \r\n de línea";
            Console.WriteLine(cadenaA);

            //Caracteres especiales. Tabuladores
            string cadenaB = "123\t456\t789\t012";
            Console.WriteLine(cadenaB);

            //Caracteres especiales. Imprimir las comillas dobles
            string cadenaC = "Esto \"acelebra\" la ejecución del programa";
            Console.WriteLine(cadenaC);

            //Usando el verbatim (toma los caracteres internos)
            string cadenaD = @"Uno puede seleccionar
                                C# o Visual Basic .NET
                                para programar en .NET
                                ambos generan el mismo código precompilado";
            Console.WriteLine(cadenaD);
        }
    }
}
```

```
Este es un salto
de línea
123      456      789      012
Esto "acelebra" la ejecución del programa
Uno puede seleccionar
                                C# o Visual Basic .NET
                                para programar en .NET
                                ambos generan el mismo código precompilado

C:\Users\engin\source\repos\Ejemplo\Ejemplo\bin\Debug\net8.0\Ejemplo.exe (proceso 15368) se cerró con el código 0.
Para cerrar automáticamente la consola cuando se detiene la depuración, habilite Herramientas ->Opciones ->Depuración ->
Cerrar la consola automáticamente al detenerse la depuración.
Presione cualquier tecla para cerrar esta ventana. . .
```

Ilustración 71: Caracteres especiales

Acceder a un determinado carácter

En C# la primera letra en un string está en la posición cero.

B/006.cs

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Acceder a determinado caracter
            string cadena = "QWERTYUIOPabcdefghijklmnñopqrstuvwxyz";
            char letra = cadena[0]; //Accede a la primera letra
            Console.WriteLine(letra);
        }
    }
}
```

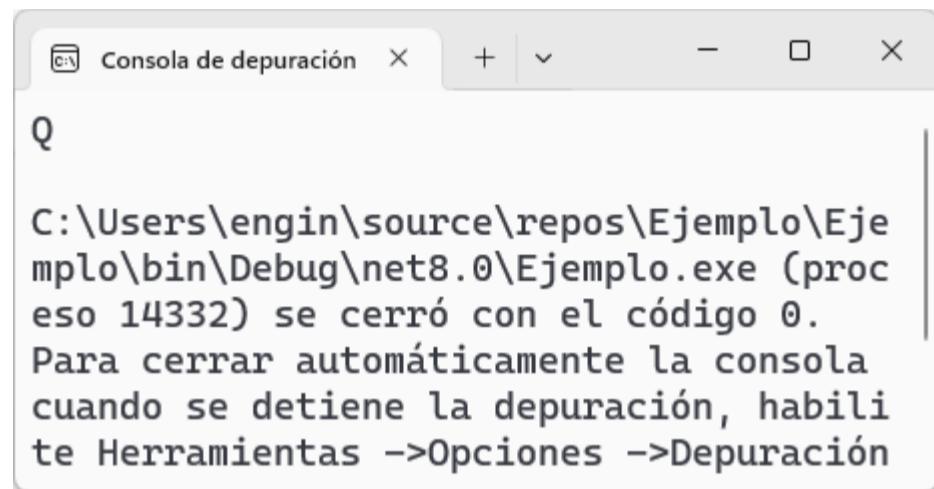


Ilustración 72: Acceder a un determinado carácter

Tamaño de la cadena y recorrerla

Para obtener el tamaño en caracteres de una cadena se hace uso de la instrucción Length. En cuanto a su recorrido, se inicia en cero y la última letra sería el tamaño restándole uno.

B/007.cs

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Tamaño de cadena y recorrerla
            string cadena = "QWERTYUIOPabcdefghijklmnñopqrstuvwxyz";
            int tamano = cadena.Length;
            Console.WriteLine(cadena);
            Console.WriteLine("Tamaño es: " + tamano.ToString());

            //Recorre la cadena
            for (int posicion=0; posicion < tamano; posicion++) {
                char letra = cadena[posicion]; //va de letra en letra
                Console.Write(letra.ToString() + " ; ");
            }
        }
    }
}
```

```
QWERTYUIOPabcdefghijklmnñopqrstuvwxyz
Tamaño es: 37
Q ; W ; E ; R ; T ; Y ; U ; I ; O ; P ; 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 ;
C:\Users\engin\source\repos\Ejemplo\Ejemplo\bin\Debug\net8.0\Ejemplo.e
xe (proceso 21916) se cerró con el código 0.
Para cerrar automáticamente la consola cuando se detiene la depuración
, habilite Herramientas ->Opciones ->Depuración ->Cerrar la consola au
tomáticamente al detenerse la depuración.
Presione cualquier tecla para cerrar esta ventana. . .
```

Ilustración 73: Tamaño de la cadena y recorrerla

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Subcadenas
            string cadena = "abcdefghijklmnñopqrstuvwxyz";

            string subCadA = cadena.Substring(3); //Del caracter 3 en adelante
            Console.WriteLine(subCadA);

            string subCadB = cadena.Substring(7, 4); //Del caracter 7 traiga 4 caracteres
            Console.WriteLine(subCadB);
        }
    }
}
```

```
defghijklmnñopqrstuvwxyz
hijk

C:\Users\engin\source\repos\Ejemplo\Ejemplo\bin\Debug\net8.0\Ejemplo.exe (proc)
```

Ilustración 74: Subcadenas

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Reemplazar
            string cadena = "manzana y naranja";
            Console.WriteLine(cadena);

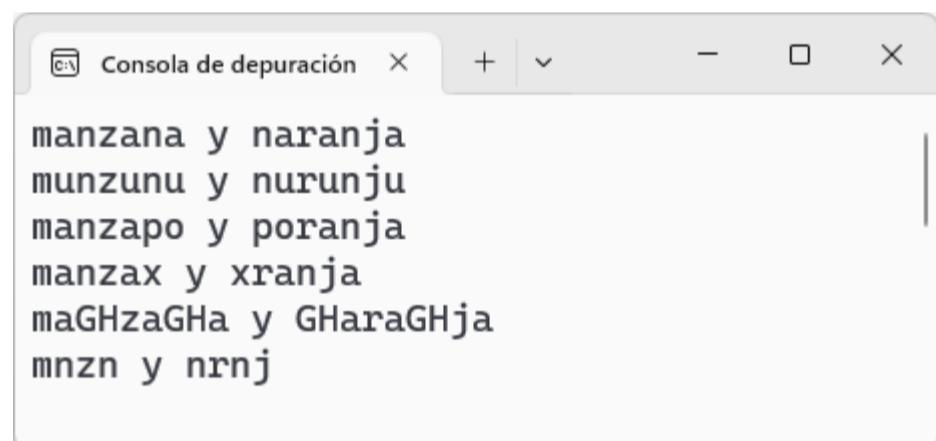
            //Cambia en toda la cadena una letra por otra
            string ReemplazaA = cadena.Replace('a', 'u');
            Console.WriteLine(ReemplazaA);

            //Cambia en toda la cadena una subcadena por otra subcadena
            string ReemplazaB = cadena.Replace("na", "po");
            Console.WriteLine(ReemplazaB);

            //Cambia en toda la cadena una subcadena por una letra
            string ReemplazaC = cadena.Replace("na", "x");
            Console.WriteLine(ReemplazaC);

            //Cambia en toda la cadena una letra por una subcadena
            string ReemplazaD = cadena.Replace("n", "GH");
            Console.WriteLine(ReemplazaD);

            //Cambia en toda la cadena una letra por vacío
            string ReemplazaE = cadena.Replace("a", "");
            Console.WriteLine(ReemplazaE);
        }
    }
}
```



The screenshot shows the 'Consola de depuración' (Debug Console) window with the following output:

```
manzana y naranja
munzunu y nurunju
manzapo y poranja
manzax y xranja
maGHzaGHa y GHaraGHja
mnzn y nrnj
```

Ilustración 75: Reemplazar caracteres

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Encontrar subcadenas
            string cadena = "manzana y naranja";
            Console.WriteLine(cadena);

            //Busca la primera posición de la letra "a"
            int posA = cadena.IndexOf('a');
            Console.WriteLine("Posición de la primera 'a' es: " + posA.ToString());

            //Busca una letra que no existe
            int posB = cadena.IndexOf('K');
            Console.WriteLine("Posición de la primera 'K' es: " + posB.ToString());

            //Busca la primera posición de la subcadena "na"
            int posC = cadena.IndexOf("na");
            Console.WriteLine("Posición de la primera 'na' es: " + posC.ToString());

            //Busca la segunda posición de la letra "a"
            int posD = cadena.IndexOf('a', posA+1);
            Console.WriteLine("Posición de la segunda 'a' es: " + posD.ToString());

            //Busca la tercera posición de la letra "a"
            int posE = cadena.IndexOf('a', posD + 1);
            Console.WriteLine("Posición de la tercera 'a' es: " + posE.ToString());
        }
    }
}
```

```
manzana y naranja
Posición de la primera 'a' es: 1
Posición de la primera 'K' es: -1
Posición de la primera 'na' es: 5
Posición de la segunda 'a' es: 4
Posición de la tercera 'a' es: 6
```

Ilustración 76: Encontrar subcadenas o caracteres

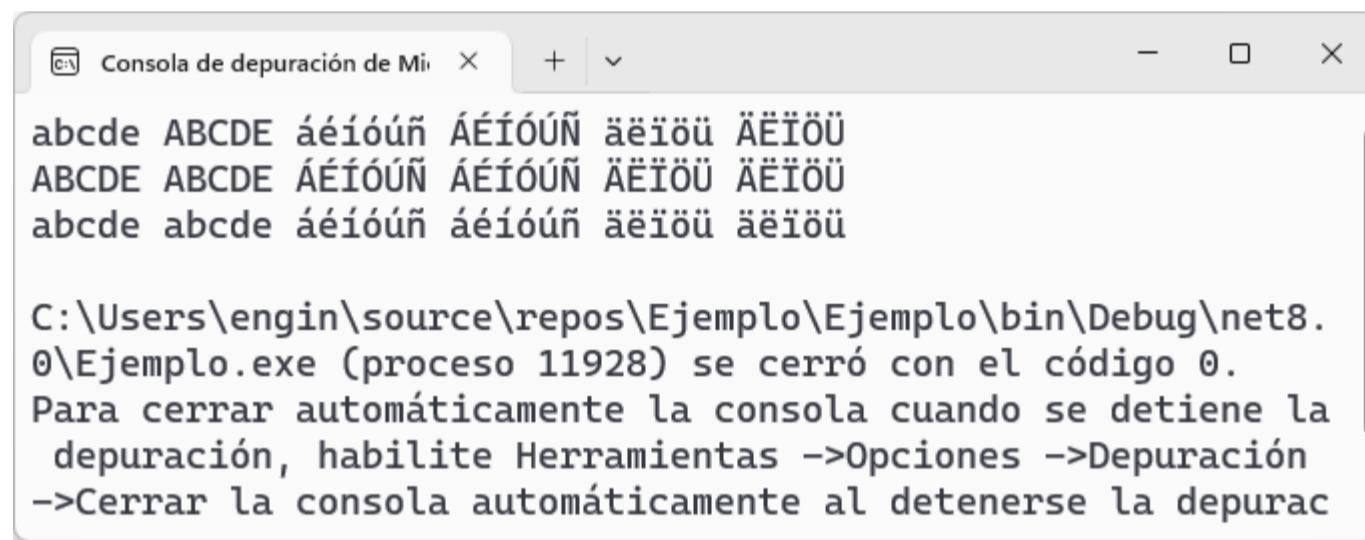
Convertir a mayúsculas y minúsculas

B/011.cs

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Mayúsculas y minúsculas
            string cadena = "abcde ABCDE áéíóúñ ÁÉÍÓÚÑ äëïöü ÄËÏÖÜ";
            Console.WriteLine(cadena);

            //Convierte a mayúscula
            string mayuscula = cadena.ToUpper();
            Console.WriteLine(mayuscula);

            //Convierte a minúscula
            string minuscula = cadena.ToLower();
            Console.WriteLine(minuscula);
        }
    }
}
```



```
abcde ABCDE áéíóúñ ÁÉÍÓÚÑ äëïöü ÄËÏÖÜ
ABCDE ABCDE ÁÉÍÓÚÑ ÁÉÍÓÚÑ ÄËÏÖÜ ÄËÏÖÜ
abcde abcde áéíóúñ áéíóúñ äëïöü äëïöü

C:\Users\engin\source\repos\Ejemplo\Ejemplo\bin\Debug\net8.0\Ejemplo.exe (proceso 11928) se cerró con el código 0.
Para cerrar automáticamente la consola cuando se detiene la depuración, habilite Herramientas ->Opciones ->Depuración
->Cerrar la consola automáticamente al detenerse la depurac
```

Ilustración 77: Convertir a mayúsculas y minúsculas

```
using System.Text; //Requiere para StringBuilder

namespace Ejemplo {
    internal class Program {
        static void Main() {
            //StringBuilder, mayor velocidad y se puede modificar
            StringBuilder cadenaRapida = new StringBuilder("Esta es una prueba");
            Console.WriteLine(cadenaRapida.ToString());

            //Cambia el primer caracter
            cadenaRapida[0] = 'e';
            Console.WriteLine(cadenaRapida.ToString());

            //Agrega caracteres
            for (int numero=0; numero <= 9; numero++) {
                cadenaRapida.Append(numero.ToString());
            }
            Console.WriteLine(cadenaRapida.ToString());
        }
    }
}
```

```
Esta es una prueba
esta es una prueba
esta es una prueba0123456789

C:\Users\engin\source\repos\Ejemplo\Ejemplo\bin\Debug\net8.0\Ejemplo.exe (proceso 14692) se cerró con el código 0.
Para cerrar automáticamente la consola cuando se detiene la depuración, habilite Herramientas ->Opciones ->Depuración ->Cerrar la consola automáticamente al detenerse la depuración.
Presione cualquier tecla para cerrar esta ventana. . .
```

Ilustración 78: StringBuilder, más veloz y se puede modificar su contenido

```

using System.Diagnostics; //Requiere para medir tiempos
using System.Text; //Requiere para StringBuilder

namespace Ejemplo {
    internal class Program {
        static void Main() {
            //StringBuilder, comparando la velocidad
            StringBuilder cadenaRapida = new StringBuilder();
            string cadenaClasica = "";

            //Medidor de tiempos
            Stopwatch cronometro = new Stopwatch();

            //Agrega caracteres a un StringBuilder
            cronometro.Reset();
            cronometro.Start();
            for (int numero=0; numero <= 20000; numero++) {
                cadenaRapida.Append(numero.ToString());
            }
            long TBuilder = cronometro.ElapsedMilliseconds;

            //Agrega caracteres a un string
            cronometro.Reset();
            cronometro.Start();
            for (int numero = 0; numero <= 20000; numero++) {
                cadenaClasica += numero.ToString();
            }
            long TClasica = cronometro.ElapsedMilliseconds;

            Console.WriteLine("Tiempo en milisegundos StringBuilder: " + TBuilder.ToString());
            Console.WriteLine("Tiempo en milisegundos cadena clásica: " + TClasica.ToString());
            Console.WriteLine("Tamaño StringBuilder: " + cadenaRapida.Length);
            Console.WriteLine("Tamaño cadena clásica: " + cadenaClasica.Length);
        }
    }
}

```

```

C:\Users\engin\source\repos\Ejemplo\Ejemplo\bin\Debug\net8.0\Ejemplo.exe (proceso 10080) se cerró con el código 0.
Para cerrar automáticamente la consola cuando se detiene la depuración, habilite Herramientas ->Opciones ->Depuración
->Cerrar la consola automáticamente al detenerse la depurac

```

Ilustración 79: StringBuilder, métricas de velocidad

```

namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Cortando espacios
            string cadenaA = "    espacios al inicio y al final      ";
            Console.WriteLine("[" + cadenaA + "]");

            //Quita los espacios de inicio y fin
            string cadenaB = cadenaA.Trim(' ');
            Console.WriteLine("[" + cadenaB + "]\r\n");

            //¿Y si son tabuladores? Los retira: \t
            string cadenaC = "\t\tUsando tabuladores al inicio y final\t\t";
            Console.WriteLine("[" + cadenaC + "]");
            string cadenaD = cadenaC.Trim('\t');
            Console.WriteLine("[" + cadenaD + "]");

            //Retirar los tabuladores de la izquierda
            string cadenaE = cadenaC.TrimStart('\t');
            Console.WriteLine("[" + cadenaE + "]");

            //Retirar los tabuladores de la derecha
            string cadenaF = cadenaC.TrimEnd('\t');
            Console.WriteLine("[" + cadenaF + "]\r\n");

            //Retirar otro caracter
            cadenaA = "aaaaaaaaaaUN CARACTER AL INICIO Y FINALaaaaaaaaaaa";
            Console.WriteLine("[" + cadenaA + "]");
            string cadenaG = cadenaA.Trim('a');
            Console.WriteLine("[" + cadenaG + "]");
        }
    }
}

```

```

[espacios al inicio y al final ]
[espacios al inicio y al final ]

[Usando tabuladores al inicio y final      ]
[Usando tabuladores al inicio y final]
[Usando tabuladores al inicio y final      ]
[Usando tabuladores al inicio y final]

[aaaaaaaaaaUN CARACTER AL INICIO Y FINALaaaaaaaaaaa]
[UN CARACTER AL INICIO Y FINAL]

C:\Users\engin\source\repos\Ejemplo\Ejemplo\bin\Debug\net8.

```

Ilustración 80: Eliminar caracteres al inicio y al final

Comparar cadenas. Forma no recomendada

No se recomienda hacer uso del operador == al usar cadenas.

B/015.cs

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Comparar cadenas
            string cadenaA = "abcdefghijkl";
            string cadenaB = "Abcdefghij";
            string cadenaC = "abcdefghijkl ";
            string cadenaD = "abcdefghijklhij";

            //Forma 1 de comparar. No recomendada.
            if (cadenaA == cadenaB)
                Console.WriteLine("1. Iguales");
            else
                Console.WriteLine("1. Diferentes");

            if (cadenaA == cadenaC)
                Console.WriteLine("2. Iguales");
            else
                Console.WriteLine("2. Diferentes");

            if (cadenaA == cadenaD)
                Console.WriteLine("3. Iguales");
            else
                Console.WriteLine("3. Diferentes");
        }
    }
}
```

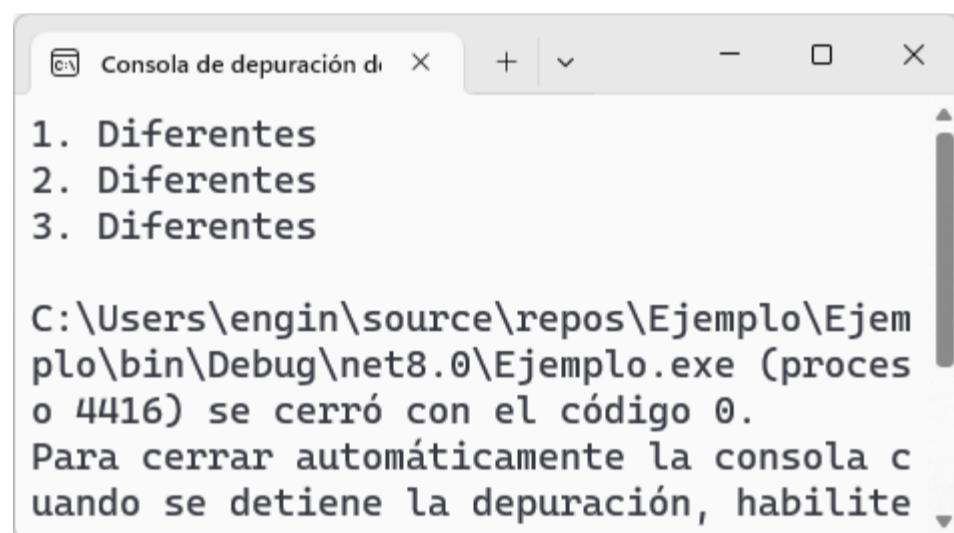


Ilustración 81: Comparar cadenas. Forma no recomendada

Comparar cadenas. Forma recomendada

Para comparar es mejor usar el operador Equals

B/016.cs

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Comparar cadenas
            string cadenaA = "abcdefghijkl";
            string cadenaB = "Abcdefghij";
            string cadenaC = "abcdefghijkl ";
            string cadenaD = "abcdefghijklhij";

            //Forma 2 de comparar. Recomendada.
            if (cadenaA.Equals(cadenaB))
                Console.WriteLine("1. Iguales");
            else
                Console.WriteLine("1. Diferentes");

            if (cadenaA.Equals(cadenaC))
                Console.WriteLine("2. Iguales");
            else
                Console.WriteLine("2. Diferentes");

            if (cadenaA.Equals(cadenaD))
                Console.WriteLine("3. Iguales");
            else
                Console.WriteLine("3. Diferentes");
        }
    }
}
```

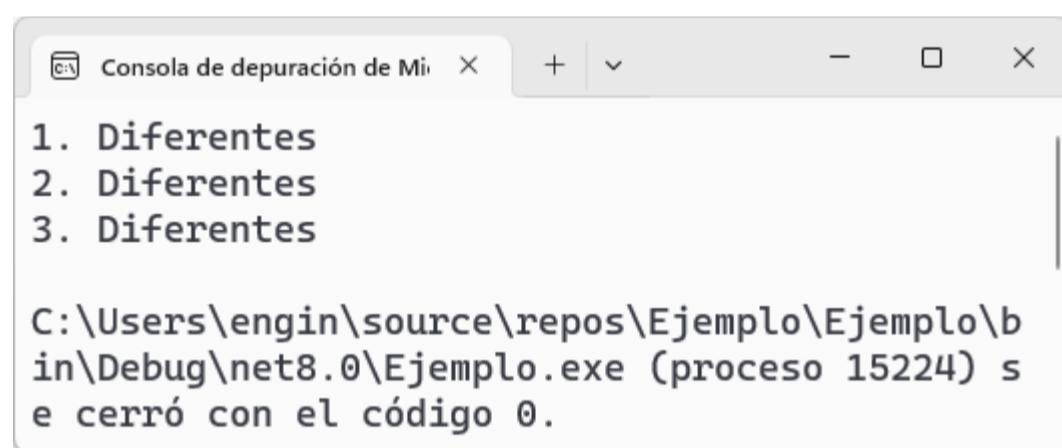


Ilustración 82: Comparar cadenas. Forma recomendada

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Comparar cadenas
            string cadenaA = "abcdefghijkl";
            string cadenaB = "Abcdefghij";
            string cadenaC = "aBCDEfghiJ";
            string cadenaD = "ABCDEFgHIj";

            //Forma 2 de comparar ignorando mayúsculas y minúsculas
            if (cadenaA.Equals(cadenaB, StringComparison.OrdinalIgnoreCase))
                Console.WriteLine("1. Iguales");
            else
                Console.WriteLine("1. Diferentes");

            if (cadenaA.Equals(cadenaC, StringComparison.OrdinalIgnoreCase))
                Console.WriteLine("2. Iguales");
            else
                Console.WriteLine("2. Diferentes");

            if (cadenaA.Equals(cadenaD, StringComparison.OrdinalIgnoreCase))
                Console.WriteLine("3. Iguales");
            else
                Console.WriteLine("3. Diferentes");
        }
    }
}

//Más información: https://docs.microsoft.com/en-us/dotnet/csharp/how-to/compare-strings
```

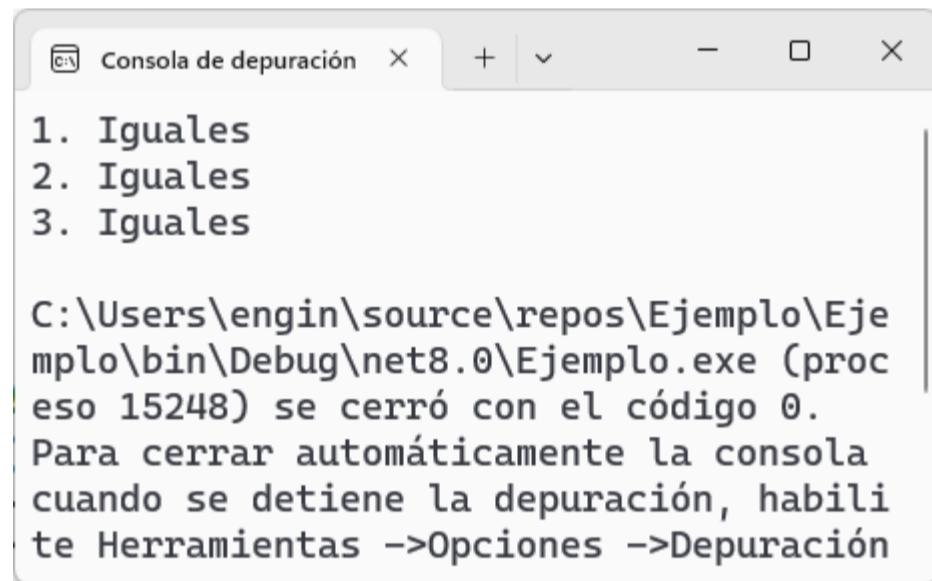


Ilustración 83: Comparar cadenas. Ignorando las mayúsculas y minúsculas

Parte 3: Arreglos estáticos

Los arreglos unidimensionales empiezan en la posición 0.

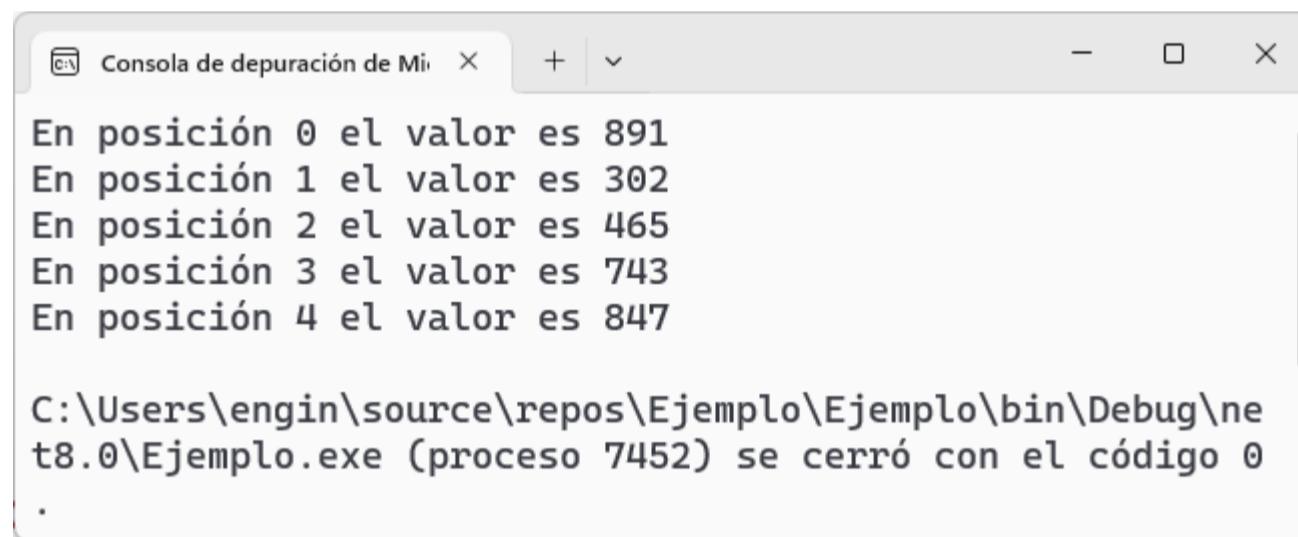
Declaración y asignar valores

C/001.cs

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Arreglo unidimensional. Definición.
            int[] arreglo = new int[5];

            //Asignando valores a cada posición del arreglo
            arreglo[0] = 891;
            arreglo[1] = 302;
            arreglo[2] = 465;
            arreglo[3] = 743;
            arreglo[4] = 847;

            //Imprimiendo valores
            Console.WriteLine("En posición 0 el valor es " + arreglo[0].ToString());
            Console.WriteLine("En posición 1 el valor es " + arreglo[1].ToString());
            Console.WriteLine("En posición 2 el valor es " + arreglo[2].ToString());
            Console.WriteLine("En posición 3 el valor es " + arreglo[3].ToString());
            Console.WriteLine("En posición 4 el valor es " + arreglo[4].ToString());
        }
    }
}
```



```
En posición 0 el valor es 891
En posición 1 el valor es 302
En posición 2 el valor es 465
En posición 3 el valor es 743
En posición 4 el valor es 847

C:\Users\engin\source\repos\Ejemplo\Ejemplo\bin\Debug\net8.0\Ejemplo.exe (proceso 7452) se cerró con el código 0
```

Ilustración 84: Declaración y asignar valores

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Arreglo unidimensional. Definición y asignación.
            int[] arreglo = { 709, 134, 568, 321, 517 };

            //Imprimiendo valores
            Console.WriteLine("En posición 0 el valor es " + arreglo[0].ToString());
            Console.WriteLine("En posición 1 el valor es " + arreglo[1].ToString());
            Console.WriteLine("En posición 2 el valor es " + arreglo[2].ToString());
            Console.WriteLine("En posición 3 el valor es " + arreglo[3].ToString());
            Console.WriteLine("En posición 4 el valor es " + arreglo[4].ToString());
        }
    }
}
```

```
En posición 0 el valor es 709
En posición 1 el valor es 134
En posición 2 el valor es 568
En posición 3 el valor es 321
En posición 4 el valor es 517
C:\Users\engin\source\repos\Ejemplo\Ejemplo\bin\Debug\net8.0\Ejemplo.exe (proceso 26)
```

Ilustración 85: Otra forma de definir y asignar

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Arreglo unidimensional de cadenas.
            string[] cadenas = [ "naranja", "manzana", "pera", "coco" ];

            //Imprimiendo valores
            Console.WriteLine("En posición 0 el valor es " + cadenas[0]);
            Console.WriteLine("En posición 1 el valor es " + cadenas[1]);
            Console.WriteLine("En posición 2 el valor es " + cadenas[2]);
            Console.WriteLine("En posición 3 el valor es " + cadenas[3]);
        }
    }
}
```

```
En posición 0 el valor es naranja
En posición 1 el valor es manzana
En posición 2 el valor es pera
En posición 3 el valor es coco

C:\Users\engin\source\repos\Ejemplo\Ejemplo\b
in\Debug\net8.0\Ejemplo.exe (proceso 18208) s
e cerró con el código 0.
```

Ilustración 86: Arreglo unidimensional de cadenas

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Arreglo unidimensional. Tamaños.
            string[] cadenas = [ "naranja", "manzana", "pera", "coco" ];
            int[] numeros = new int[8];

            //Tamaños de los arreglos
            int TamanoCadenas = cadenas.Length;
            int TamanoNumeros = numeros.Length;

            //Imprime
            Console.WriteLine("Tamaño de arreglo cadenas:" + TamanoCadenas.ToString());
            Console.WriteLine("Tamaño de arreglo numeros:" + TamanoNumeros.ToString());
        }
    }
}
```

```
Consola de depuración de Mi... X + - □ ×
Tamaño de arreglo cadenas:4
Tamaño de arreglo numeros:8

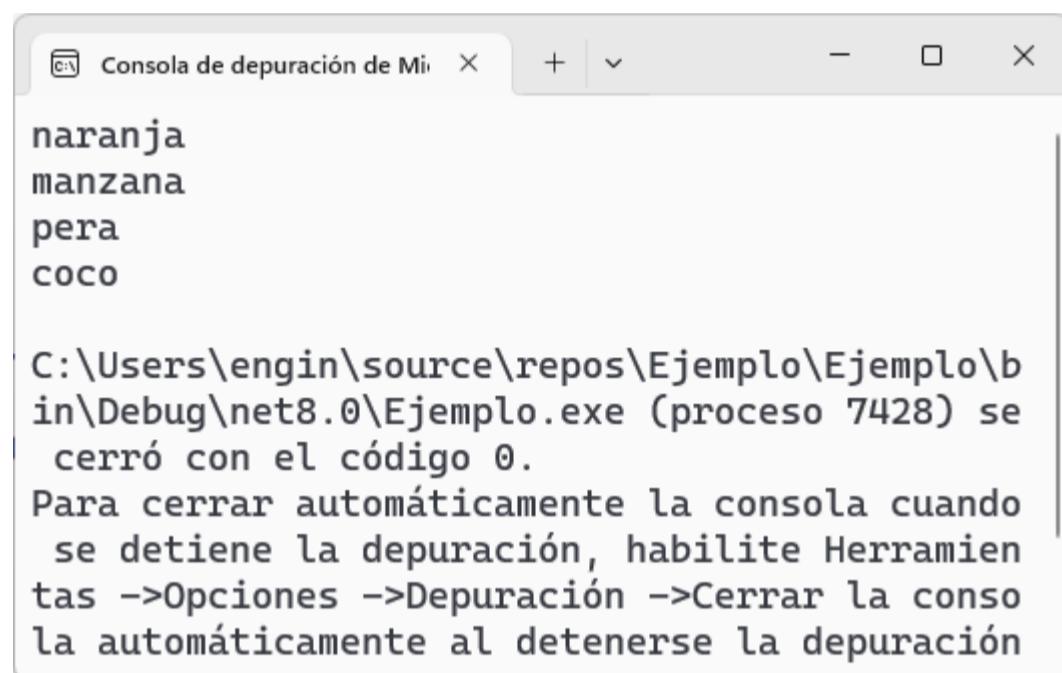
C:\Users\engin\source\repos\Ejemplo\Ejemplo\b
in\Debug\net8.0\Ejemplo.exe (proceso 2328) se
cerró con el código 0.
Para cerrar automáticamente la consola cuando
se detiene la depuración, habilite Herramien
tas ->Opciones ->Depuración ->Cerrar la conso
la automáticamente al detenerse la depuración
.
Presione cualquier tecla para cerrar esta ven
```

Ilustración 87: Arreglo unidimensional. Tamaños

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Arreglo unidimensional
            string[] cadenas = [ "naranja", "manzana", "pera", "coco" ];

            //Tamaño
            int TamanoCadenas = cadenas.Length;

            //Recorre el arreglo y lo imprime
            for (int posicion=0; posicion<TamanoCadenas; posicion++) {
                Console.WriteLine(cadenas[posicion]);
            }
        }
    }
}
```



```
naranja
manzana
pera
coco

C:\Users\engin\source\repos\Ejemplo\Ejemplo\b
in\Debug\net8.0\Ejemplo.exe (proceso 7428) se
cerró con el código 0.
Para cerrar automáticamente la consola cuando
se detiene la depuración, habilite Herramientas ->Opciones ->Depuración ->Cerrar la conso
la automáticamente al detenerse la depuración
```

Ilustración 88: Recorrer arreglos de cadenas

Uso de la instrucción foreach para recorrer un arreglo

C/006.cs

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Arreglo unidimensional
            string[] cadenas = [ "naranja", "manzana", "pera", "coco" ];

            //Recorre el arreglo y lo imprime
            foreach(string texto in cadenas) {
                Console.WriteLine(texto);
            }
        }
    }
}
```

The screenshot shows a Windows Command Prompt window titled 'Consola de depuración de'. The window displays the following text:
naranja
manzana
pera
coco

C:\Users\engin\source\repos\Ejemplo\Ejemplo\bin\Debug\net8.0\Ejemplo.exe (proceso 6548) se cerró con el código 0.
Para cerrar automáticamente la consola cuando se detiene la depuración, habilite Herramientas ->Opciones ->Depuración ->

Ilustración 89: Uso de la instrucción foreach para recorrer un arreglo

Ordenar un arreglo de cadenas

C/007.cs

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Arreglo unidimensional
            string[] cadenas = [ "raf", "ael", "alb", "ert", "omo", "ren", "opa", "rra" ];

            //Ordena el arreglo
            Array.Sort(cadenas);

            //Recorre el arreglo y lo imprime
            foreach(string texto in cadenas) {
                Console.WriteLine(texto);
            }
        }
    }
}
```

```
ael
alb
ert
omo
opa
raf
ren
rra

C:\Users\engin\source\repos\Ejemplo\Ejemplo\bin\Debug\net8.0\Ejemplo.exe (proces
```

Ilustración 90: Ordenar un arreglo de cadenas

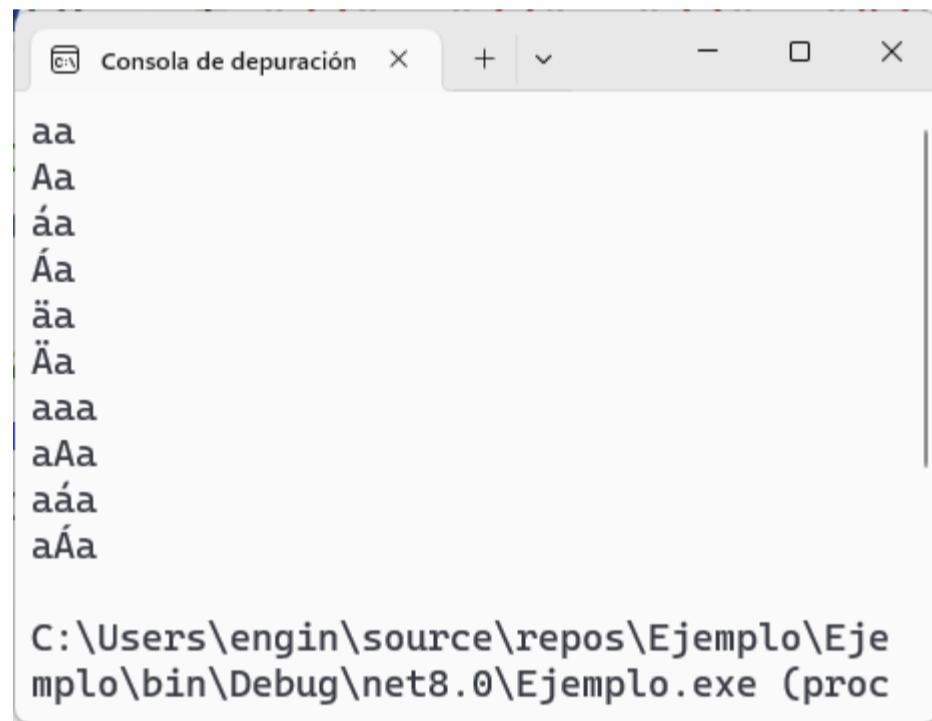
```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Arreglo unidimensional
            string[] cadenas = [ "áa", "aa", "äa", "Aa", "Äa", "Áa", "aaa", "aáa", "aAa", "aÁa" ];

            //Ordena el arreglo
            Array.Sort(cadenas);

            //Recorre el arreglo y lo imprime
            foreach(string texto in cadenas) {
                Console.WriteLine(texto);
            }
        }
    }
}
```

En caso de probar las tildes, diéresis, mayúsculas y minúsculas, se obtendría este resultado con cadenas de una sola letra.

```
string[] cadenas = { "á", "é", "í", "ó", "ú", "Á", "É", "Í", "Ó", "Ú", "A", "E", "I", "O", "U", "a", "e",
    "í", "ö", "ü", "Ä", "Ë", "Ï", "Ö", "Ü", "ä", "ë", "ï", "ö", "ü" };
```



```
aa
Aa
áa
Áa
äa
Äa
aaa
aAa
aáa
aÁa
```

```
C:\Users\engin\source\repos\Ejemplo\Ejemplo\bin\Debug\net8.0\Ejemplo.exe (proc
```

Ilustración 91: Ordenar un arreglo de cadenas. Las mayúsculas, minúsculas, tildes y diéresis

Ordenar un arreglo de tipo double

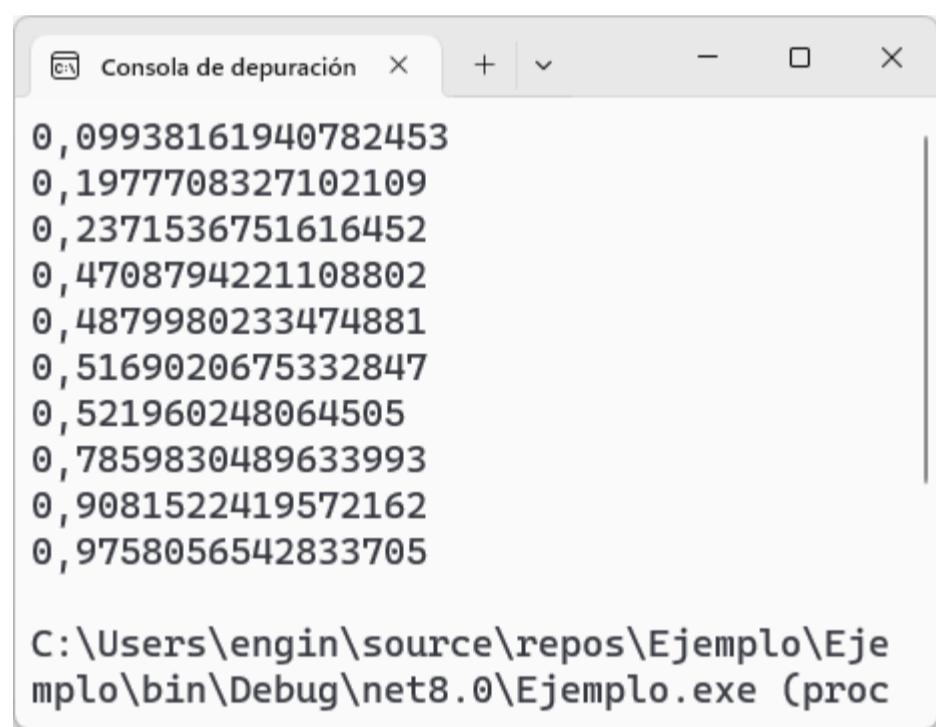
C/009.cs

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            Random Azar = new Random();

            //Declara y llena un arreglo con valores aleatorios
            double[] Numeros = new double[10];
            for(int cont=0; cont<Numeros.Length; cont++)
                Numeros[cont] = Azar.NextDouble();

            //Ordena el arreglo
            Array.Sort(Numeros);

            //Recorre el arreglo y lo imprime
            foreach (double unvalor in Numeros) {
                Console.WriteLine(unvalor);
            }
        }
    }
}
```



```
0,09938161940782453
0,1977708327102109
0,2371536751616452
0,4708794221108802
0,4879980233474881
0,5169020675332847
0,521960248064505
0,7859830489633993
0,9081522419572162
0,9758056542833705

C:\Users\engin\source\repos\Ejemplo\Ejemplo\bin\Debug\net8.0\Ejemplo.exe (proc)
```

Ilustración 92: Ordenar un arreglo de tipo double

Funciones genéricas para arreglos unidimensionales

Se crean funciones que se envía como parámetro el arreglo

C/010.cs

```
namespace Ejemplo {
    class Program {
        static void Main() {
            //Arreglo unidimensional. Funciones genéricas
            int[] numeros = new int[50];

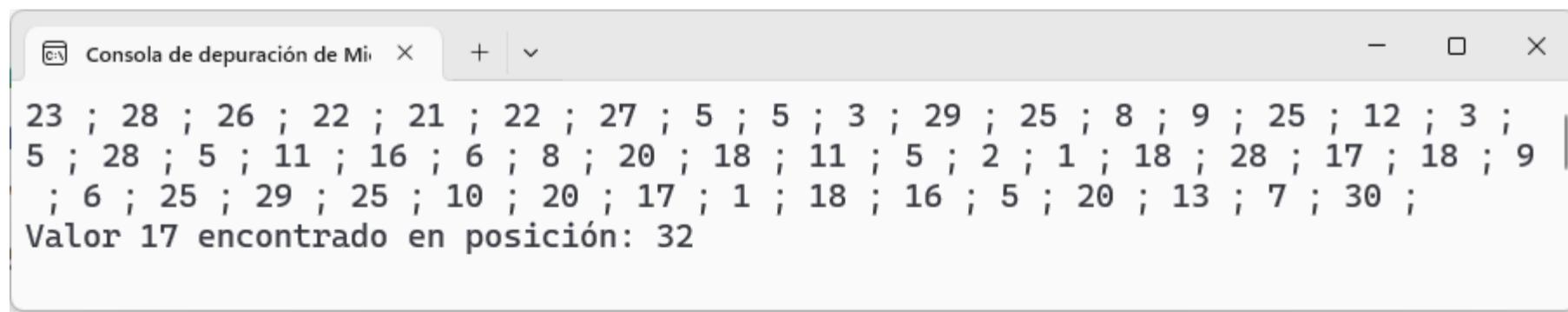
            //Llena con valores al azar
            int minimo = 1, maximo = 30;
            LlenaArreglo(numeros, minimo, maximo);
            ImprimeArreglo(numeros);

            //Busca un valor determinado y retorna su posición
            int valorBusca = 17;
            int encuentra = PosArregloDatos(numeros, valorBusca);
            if (encuentra == -1)
                Console.WriteLine("Valor no encontrado");
            else
                Console.WriteLine("Valor " + valorBusca.ToString() + " encontrado en posición: " +
                    encuentra.ToString());
        }

        //Llena el arreglo con valores al azar entre min y max (ambos incluidos)
        static void LlenaArreglo(int[] arreglo, int min, int max) {
            Random azar = new Random();
            for (int posicion = 0; posicion < arreglo.Length; posicion++) {
                arreglo[posicion] = azar.Next(min, max+1);
            }
        }

        //Imprime el arreglo en consola
        static void ImprimeArreglo(int[] arreglo) {
            for (int posicion = 0; posicion < arreglo.Length; posicion++) {
                Console.Write(arreglo[posicion].ToString() + " ; ");
            }
            Console.WriteLine(" ");
        }

        //Retorna la posición del dato en el arreglo o retorna -1 si no lo encuentra
        static int PosArregloDatos(int[] arreglo, int valor) {
            for (int posicion = 0; posicion < arreglo.Length; posicion++) {
                if (arreglo[posicion] == valor)
                    return posicion;
            }
            return -1;
        }
    }
}
```



23 ; 28 ; 26 ; 22 ; 21 ; 22 ; 27 ; 5 ; 5 ; 3 ; 29 ; 25 ; 8 ; 9 ; 25 ; 12 ; 3 ;
5 ; 28 ; 5 ; 11 ; 16 ; 6 ; 8 ; 20 ; 18 ; 11 ; 5 ; 2 ; 1 ; 18 ; 28 ; 17 ; 18 ; 9
; 6 ; 25 ; 29 ; 25 ; 10 ; 20 ; 17 ; 1 ; 18 ; 16 ; 5 ; 20 ; 13 ; 7 ; 30 ;
Valor 17 encontrado en posición: 32

Ilustración 93: Funciones genéricas para arreglos unidimensionales

Algoritmos de ordenación

Se prueban diversos algoritmos de ordenación enviando arreglos como parámetros y estos arreglos son modificados por las funciones mismas. Al enviar un arreglo como parámetro a una función o procedimiento, este es enviado por referencia (**no se copian los valores** como sucede con los tipos de datos nativos como int, double, float, string). Luego se debe tener especial cuidado con esto porque si la función o procedimiento modifica el arreglo, quedaría modificado también desde el sitio en que se llamó a la función o procedimiento.

C/011.cs

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Ordenación
            int Limite = 30;
            int[] numA = new int[Limite];
            int[] numB = new int[Limite];

            int minimo = 10, maximo = 99;
            LlenaArreglo(numB, minimo, maximo);

            //Ordena por inserción
            Console.WriteLine("\nOrdena por inserción");
            Array.Copy(numB, 0, numA, 0, numB.Length); //Copia arreglo numB ===> numA
            ImprimeArreglo(numA);
            Insercion(numA);
            ImprimeArreglo(numA);

            //Ordena por selección
            Console.WriteLine("\nOrdena por selección");
            Array.Copy(numB, 0, numA, 0, numB.Length);
            ImprimeArreglo(numA);
            Seleccion(numA);
            ImprimeArreglo(numA);

            //Ordena por burbuja
            Console.WriteLine("\nOrdena por burbuja");
            Array.Copy(numB, 0, numA, 0, numB.Length);
            ImprimeArreglo(numA);
            Burbuja(numA);
            ImprimeArreglo(numA);

            //Ordena por shell
            Console.WriteLine("\nOrdena por shell");
            Array.Copy(numB, 0, numA, 0, numB.Length);
            ImprimeArreglo(numA);
            Shell(numA);
            ImprimeArreglo(numA);

            //Ordena por QuickSort
            Console.WriteLine("\nOrdena por quicksort");
            Array.Copy(numB, 0, numA, 0, numB.Length);
            ImprimeArreglo(numA);
            QuickSort(numA, 0, numA.Length - 1);
            ImprimeArreglo(numA);
        }

        //Llena el arreglo con valores al azar entre min y max (ambos incluidos)
        static void LlenaArreglo(int[] arreglo, int min, int max) {
            Random azar = new Random();
            for (int posicion = 0; posicion < arreglo.Length; posicion++) {
                arreglo[posicion] = azar.Next(min, max + 1);
            }
        }

        //Imprime el arreglo en consola
        static void ImprimeArreglo(int[] arreglo) {
            for (int posicion = 0; posicion < arreglo.Length; posicion++) {
                Console.Write(arreglo[posicion].ToString() + " ");
            }
            Console.WriteLine();
        }

        //Ordenamiento por Insert
        static void Insercion(int[] arreglo) {
            int j;
            for (int i = 1; i < arreglo.Length; i++) {
                int tmp = arreglo[i];
                for (j = i - 1; j >= 0 && arreglo[j] > tmp; j--) {
```

```

        arreglo[j + 1] = arreglo[j];
    }
    arreglo[j + 1] = tmp;
}

//Ordenamiento por Selección
static void Seleccion(int[] arreglo) {
    for (int i = 0; i < arreglo.Length - 1; i++) {
        int min = i;
        for (int j = i + 1; j < arreglo.Length; j++) {
            if (arreglo[j] < arreglo[min]) {
                min = j;
            }
        }
        if (i != min) {
            int aux = arreglo[i];
            arreglo[i] = arreglo[min];
            arreglo[min] = aux;
        }
    }
}

//Ordenamiento por Burbuja
static void Burbuja(int[] arreglo) {
    int n = arreglo.Length;
    int tmp;
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - 1; j++) {
            if (arreglo[j] > arreglo[j + 1]) {
                tmp = arreglo[j];
                arreglo[j] = arreglo[j + 1];
                arreglo[j + 1] = tmp;
            }
        }
    }
}

//Ordenamiento por Shell
static void Shell(int[] arreglo) {
    int incremento = arreglo.Length;
    do {
        incremento /= 2;
        for (int k = 0; k < incremento; k++) {
            for (int i = incremento + k; i < arreglo.Length; i += incremento) {
                int j = i;
                while (j - incremento >= 0 && arreglo[j] < arreglo[j - incremento]) {
                    int tmp = arreglo[j];
                    arreglo[j] = arreglo[j - incremento];
                    arreglo[j - incremento] = tmp;
                    j -= incremento;
                }
            }
        }
    } while (incremento > 1);
}

//Ordenación por QuickSort
static void QuickSort(int[] arreglo, int primero, int ultimo) {
    int i, j, central;
    int pivote;
    central = (primero + ultimo) / 2;
    pivote = arreglo[central];
    i = primero;
    j = ultimo;
    do {
        while (arreglo[i] < pivote) i++;
        while (arreglo[j] > pivote) j--;
        if (i <= j) {
            int tmp = arreglo[i];
            arreglo[i] = arreglo[j];
            arreglo[j] = tmp;
            i++;
            j--;
        }
    } while (i <= j);

    if (primero < j) {
        QuickSort(arreglo, primero, j);
    }
}

```

```
    if (i < ultimo) {
        QuickSort(arreglo, i, ultimo);
    }
}
```

The screenshot shows a Windows Command Prompt window titled "Consola de depuración de Mi". The window contains the following text output:

```
Ordena por inserción
97;22;12;33;10;32;99;74;46;83;42;87;94;39;81;30;46;84;51;27;18;20;84;97;77;77;58;14;13;56;
10;12;13;14;18;20;22;27;30;32;33;39;42;46;46;51;56;58;74;77;77;81;83;84;84;87;94;97;97;99;

Ordena por selección
97;22;12;33;10;32;99;74;46;83;42;87;94;39;81;30;46;84;51;27;18;20;84;97;77;77;58;14;13;56;
10;12;13;14;18;20;22;27;30;32;33;39;42;46;46;51;56;58;74;77;77;81;83;84;84;87;94;97;97;99;

Ordena por burbuja
97;22;12;33;10;32;99;74;46;83;42;87;94;39;81;30;46;84;51;27;18;20;84;97;77;77;58;14;13;56;
10;12;13;14;18;20;22;27;30;32;33;39;42;46;46;51;56;58;74;77;77;81;83;84;84;87;94;97;97;99;

Ordena por shell
97;22;12;33;10;32;99;74;46;83;42;87;94;39;81;30;46;84;51;27;18;20;84;97;77;77;58;14;13;56;
10;12;13;14;18;20;22;27;30;32;33;39;42;46;46;51;56;58;74;77;77;81;83;84;84;87;94;97;97;99;

Ordena por quicksort
97;22;12;33;10;32;99;74;46;83;42;87;94;39;81;30;46;84;51;27;18;20;84;97;77;77;58;14;13;56;
10;12;13;14;18;20;22;27;30;32;33;39;42;46;46;51;56;58;74;77;77;81;83;84;84;87;94;97;97;99;

C:\Users\engin\source\repos\Ejemplo\Ejemplo\bin\Debug\net8.0\Ejemplo.exe (proceso 14060) se
cerró con el código 0.
Para cerrar automáticamente la consola cuando se detiene la depuración, habilite Herramient
as ->Opciones ->Depuración ->Cerrar la consola automáticamente al detenerse la depuración.
Presione cualquier tecla para cerrar esta ventana. . .
```

Ilustración 94: Algoritmos de ordenación

```

using System.Diagnostics;

namespace Ejemplo {
    internal class Program {
        static void Main() {
            int Limite = 15000;
            int[] numerosA = new int[Limite];
            int[] numerosB = new int[Limite];

            //Medidor de tiempos
            Stopwatch cronometro = new Stopwatch();

            long TPShell = 0, TPIIns = 0, TPSel = 0, TPBur = 0, TPQuick = 0;

            /* Si el compilador detecta que no se hace nada con el arreglo ordenado,
             * entonces quita la instrucción de llamado al método de ordenación, luego la
             * métrica de tiempo de cada método de ordenación sería errónea. Para evitar
             * esa optimización, se acumula el primer valor del arreglo ordenado en cada llamado
             * en la siguiente variable y luego se muestra su valor en consola */
            long ValorMuestra = 0;

            //Para disminuir oscilaciones en el tiempo, se hacen
            //N pruebas con cada grupo de pruebas
            int TotalPruebas = 5;
            for (int prueba = 1; prueba <= TotalPruebas; prueba++) {
                LlenaArreglo(numerosA, 10, 90);

                //Ordenación por método Shell
                Array.Copy(numerosA, 0, numerosB, 0, numerosA.Length);
                cronometro.Reset();
                cronometro.Start();
                Shell(numerosB);
                TPShell += cronometro.ElapsedMilliseconds;
                ValorMuestra += numerosB[0];

                //Ordenación por método Inserción
                Array.Copy(numerosA, 0, numerosB, 0, numerosA.Length);
                cronometro.Reset();
                cronometro.Start();
                Insercion(numerosB);
                TPIIns += cronometro.ElapsedMilliseconds;
                ValorMuestra += numerosB[0];

                //Ordenación por método Selección
                Array.Copy(numerosA, 0, numerosB, 0, numerosA.Length);
                cronometro.Reset();
                cronometro.Start();
                Seleccion(numerosB);
                TPSel += cronometro.ElapsedMilliseconds;
                ValorMuestra += numerosB[0];

                //Ordenación por método Burbuja
                Array.Copy(numerosA, 0, numerosB, 0, numerosA.Length);
                cronometro.Reset();
                cronometro.Start();
                Burbuja(numerosB);
                TPBur += cronometro.ElapsedMilliseconds;
                ValorMuestra += numerosB[0];

                //Ordenación por método QuickSort
                Array.Copy(numerosA, 0, numerosB, 0, numerosA.Length);
                cronometro.Reset();
                cronometro.Start();
                QuickSort(numerosB, 0, numerosB.Length - 1);
                TPQuick += cronometro.ElapsedMilliseconds;
                ValorMuestra += numerosB[0];
            }

            double TS = (double)TPShell / TotalPruebas;
            double TI = (double)TPIIns / TotalPruebas;
            double TL = (double)TPSel / TotalPruebas;
            double TB = (double)TPBur / TotalPruebas;
            double TQ = (double)TPQuick / TotalPruebas;

            Console.WriteLine("=====");
        }
    }
}

```

```

Console.WriteLine("Número de elementos: " + Limite.ToString());
Console.WriteLine("Valor muestra: " + ValorMuestra.ToString());
Console.WriteLine("ShellSort, tiempo promedio en milisegundos: " + TS.ToString());
Console.WriteLine("InsertSort, tiempo promedio en milisegundos: " + TI.ToString());
Console.WriteLine("Selección, tiempo promedio en milisegundos: " + TL.ToString());
Console.WriteLine("Burbuja, tiempo promedio en milisegundos: " + TB.ToString());
Console.WriteLine("QuickSort, tiempo promedio en milisegundos: " + TQ.ToString());
}

//Llena el arreglo con valores al azar entre min y max (ambos incluídos)
static void LlenaArreglo(int[] arreglo, int min, int max) {
    Random azar = new Random();
    for (int posicion = 0; posicion < arreglo.Length; posicion++) {
        arreglo[posicion] = azar.Next(min, max + 1);
    }
}

//Ordenamiento por Insert
static void Insercion(int[] arreglo) {
    int j;
    for (int i = 1; i < arreglo.Length; i++) {
        int tmp = arreglo[i];
        for (j = i - 1; j >= 0 && arreglo[j] > tmp; j--) {
            arreglo[j + 1] = arreglo[j];
        }
        arreglo[j + 1] = tmp;
    }
}

//Ordenamiento por Selección
static void Seleccion(int[] arreglo) {
    for (int i = 0; i < arreglo.Length - 1; i++) {
        int min = i;
        for (int j = i + 1; j < arreglo.Length; j++) {
            if (arreglo[j] < arreglo[min]) {
                min = j;
            }
        }
        if (i != min) {
            int aux = arreglo[i];
            arreglo[i] = arreglo[min];
            arreglo[min] = aux;
        }
    }
}

//Ordenamiento por Burbuja
static void Burbuja(int[] arreglo) {
    int n = arreglo.Length;
    int tmp;
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - 1; j++) {
            if (arreglo[j] > arreglo[j + 1]) {
                tmp = arreglo[j];
                arreglo[j] = arreglo[j + 1];
                arreglo[j + 1] = tmp;
            }
        }
    }
}

//Ordenamiento por Shell
static void Shell(int[] arreglo) {
    int incremento = arreglo.Length;
    do {
        incremento /= 2;
        for (int k = 0; k < incremento; k++) {
            for (int i = incremento + k; i < arreglo.Length; i += incremento) {
                int j = i;
                while (j - incremento >= 0 && arreglo[j] < arreglo[j - incremento]) {
                    int tmp = arreglo[j];
                    arreglo[j] = arreglo[j - incremento];
                    arreglo[j - incremento] = tmp;
                    j -= incremento;
                }
            }
        }
    } while (incremento > 1);
}

```

```

//Ordenación por QuickSort
static void QuickSort(int[] arreglo, int primero, int ultimo) {
    int i, j, central;
    int pivote;
    central = (primero + ultimo) / 2;
    pivote = arreglo[central];
    i = primero;
    j = ultimo;
    do {
        while (arreglo[i] < pivote) i++;
        while (arreglo[j] > pivote) j--;
        if (i <= j) {
            int tmp = arreglo[i];
            arreglo[i] = arreglo[j];
            arreglo[j] = tmp;
            i++;
            j--;
        }
    } while (i <= j);

    if (primero < j) {
        QuickSort(arreglo, primero, j);
    }
    if (i < ultimo) {
        QuickSort(arreglo, i, ultimo);
    }
}
}
}

```

```

Consola de depuración de Mi .NET Core App
=====
Número de elementos: 15000
Valor muestra: 250
ShellSort, tiempo promedio en milisegundos: 0,2
InsertSort, tiempo promedio en milisegundos: 41
Seleccion, tiempo promedio en milisegundos: 109,4
Burbuja, tiempo promedio en milisegundos: 296,4
QuickSort, tiempo promedio en milisegundos: 0,6

C:\Users\engin\source\repos\Ejemplo\Ejemplo\bin\Release\net8.0\Ejemplo.exe (proceso 17476) se cerró con el código 0.
Para cerrar automáticamente la consola cuando se detiene la depuración, habilite Herramientas ->Opciones ->Depuración ->Cerrar la consola automáticamente al detenerse la depuración.

```

Ilustración 95: Métrica de velocidad: Algoritmos de ordenación

```

namespace Ejemplo {
    internal class Program {
        static void Main() {
            int TotalFilas = 5;
            int TotalColumnas = 10;

            //Declara un arreglo bidimensional
            int[,] Tablero = new int[TotalFilas, TotalColumnas];

            /* Llena ese arreglo bidimensional
            Tablero.GetLength(0) Retorna el número de filas (la primera dimensión)
            Tablero.GetLength(1) Retorna el número de columnas (la segunda dimensión)
            Un arreglo bidimensional inicia en [0,0]
            Un arreglo bidimensional termina en [TotalFilas-1, TotalColumnas-1]
            */
            Random azar = new Random();
            for (int fila = 0; fila < Tablero.GetLength(0); fila++)
                for (int columna = 0; columna < Tablero.GetLength(1); columna++)
                    Tablero[fila, columna] = azar.Next(0, 9);

            //Imprime ese arreglo bidimensional
            for (int fila = 0; fila < Tablero.GetLength(0); fila++) {
                Console.WriteLine(" ");
                for (int columna = 0; columna < Tablero.GetLength(1); columna++)
                    Console.Write(Tablero[fila, columna].ToString() + " ");
            }
        }
    }
}

```

0 ; 7 ; 2 ; 7 ; 6 ; 2 ; 7 ; 8 ; 1 ; 4 ;
2 ; 1 ; 0 ; 0 ; 2 ; 2 ; 4 ; 0 ; 5 ; 2 ;
4 ; 2 ; 0 ; 5 ; 1 ; 7 ; 0 ; 1 ; 0 ; 2 ;
6 ; 6 ; 6 ; 7 ; 1 ; 4 ; 2 ; 7 ; 0 ; 5 ;
5 ; 7 ; 2 ; 5 ; 1 ; 8 ; 0 ; 8 ; 2 ; 6 ;
C:\Users\engin\source\repos\Ejemplo\Ejemplo\bin\Release\net8.0\Ejemplo.exe (proceso 15388) se cerró con el código 0.

Ilustración 96: Arreglo bidimensional

```

namespace Ejemplo {
    internal class Program {
        static void Main() {
            int DimensionX = 5;
            int DimensionY = 8;
            int DimensionZ = 3;

            //Declara un arreglo Tridimensional
            int[, ,] Tablero = new int[DimensionX, DimensionY, DimensionZ];

            /* Llena ese arreglo tridimensional
            Tablero.GetLength(0) Retorna la primera dimensión
            Tablero.GetLength(1) Retorna la segunda dimensión
            Tablero.GetLength(2) Retorna la tercera dimensión

            Un arreglo tridimensional inicia en [0,0,0]
            */
            Random azar = new Random();
            for (int posX = 0; posX < Tablero.GetLength(0); posX++)
                for (int posY = 0; posY < Tablero.GetLength(1); posY++)
                    for (int posZ = 0; posZ < Tablero.GetLength(2); posZ++)
                        Tablero[posX, posY, posZ] = azar.Next(0, 9);

            //Imprime ese arreglo tridimensional
            for (int posX = 0; posX < Tablero.GetLength(0); posX++) {
                Console.WriteLine(" ");
                for (int posY = 0; posY < Tablero.GetLength(1); posY++) {
                    Console.Write("[");
                    for (int posZ = 0; posZ < Tablero.GetLength(2); posZ++)
                        Console.Write(Tablero[posX, posY, posZ].ToString() + ";");
                    Console.Write("]   ");
                }
            }
        }
    }
}

```

The screenshot shows the 'Consola de depuración de Mi...' (Debug Console) window in Microsoft Visual Studio. The output displays a 3D array of integers. The array is 5x8x3, with dimensions explicitly shown in brackets. The values range from 0 to 9. The output is as follows:

```

[4;5;1;]  [3;8;3;]  [1;7;7;]  [6;0;5;]  [7;6;1;]  [2;2;3;]  [1;7;2;]  [0;1;1;]
[0;6;7;]  [3;2;1;]  [5;7;0;]  [1;2;6;]  [6;0;5;]  [6;1;4;]  [7;1;7;]  [8;7;6;]
[1;2;0;]  [8;2;1;]  [5;1;1;]  [0;4;6;]  [6;5;6;]  [5;4;1;]  [3;6;8;]  [3;5;5;]
[4;7;8;]  [3;5;2;]  [8;8;5;]  [2;4;2;]  [3;7;5;]  [4;2;6;]  [1;4;8;]  [0;8;3;]
[4;2;1;]  [3;6;2;]  [0;1;5;]  [6;4;6;]  [6;8;8;]  [2;6;4;]  [1;3;8;]  [1;5;5;]
C:\Users\engin\source\repos\Ejemplo\Ejemplo\bin\Release\net8.0\Ejemplo.exe (proceso 143
16) se cerró con el código 0.
Para cerrar automáticamente la consola cuando se detiene la depuración, habilite Herram
ientas ->Opciones ->Depuración ->Cerrar la consola automáticamente al detenerse la depu

```

Ilustración 97: Arreglo tridimensional

```

namespace Ejemplo {
    internal class Program {
        static void Main() {
            /* Arreglo de arreglos (NO confundirlos con arreglos bidimensionales).
             * Se entiende mejor haciendo analogía con un conjunto y subconjuntos */

            //Defino un arreglo (el conjunto)
            int[][] arreglo = new int[5][];

            //Defino los subconjuntos
            arreglo[0] = new int[7]; //Tendrá 7 elementos
            arreglo[1] = new int[3]; //Tendrá 3 elementos
            arreglo[2] = new int[9]; //Tendrá 9 elementos
            arreglo[3] = new int[4]; //Tendrá 4 elementos
            arreglo[4] = new int[8]; //Tendrá 8 elementos

            //Llenando un arreglo de arreglos
            Random azar = new Random();
            for (int conjunto = 0; conjunto < arreglo.Length; conjunto++)
                for (int subconjunto = 0; subconjunto < arreglo[conjunto].Length; subconjunto++)
                    arreglo[conjunto][subconjunto] = azar.Next(0, 9);

            //Imprime ese arreglo de arreglos
            for (int conjunto = 0; conjunto < arreglo.Length; conjunto++) {
                Console.WriteLine(" ");
                for (int subconjunto = 0; subconjunto < arreglo[conjunto].Length; subconjunto++)
                    Console.Write(arreglo[conjunto][subconjunto].ToString() + " ");
            }
        }
    }
}

```

6 ; 3 ; 5 ; 8 ; 6 ; 5 ; 7 ;
5 ; 1 ; 0 ;
5 ; 4 ; 4 ; 1 ; 0 ; 1 ; 5 ; 8 ; 8 ;
8 ; 5 ; 7 ; 0 ;
8 ; 6 ; 8 ; 0 ; 6 ; 8 ; 3 ; 1 ;
C:\Users\engin\source\repos\Ejemplo\Ejemplo\bin\Release\net8.0\Ejemplo.exe (proceso 18696) se cerró con el código 0.
Para cerrar automáticamente la consola cuando se

Ilustración 98: Arreglo de arreglos

```

using System.Diagnostics;

namespace Ejemplo {
    internal class Program {
        static void Main() {
            /* ¿Qué es más rápido? ¿Un arreglo bidimensional o un arreglo de arreglos */

            //Límite ancho*alto de ambos arreglos
            int Limite = 80;

            //Arreglo Bidimensional
            int[,] bidimensional = new int[Limite, Limite];

            //Arreglo de arreglos
            int[][] arreglo = new int[Limite][];
            for (int subconjunto = 0; subconjunto < arreglo.Length; subconjunto++)
                arreglo[subconjunto] = new int[Limite];

            //Medidor de tiempos
            Stopwatch cronometro = new Stopwatch();

            //Llenando un arreglo bidimensional
            int valor = 0;
            cronometro.Reset();
            cronometro.Start();
            for (int fila = 0; fila < bidimensional.GetLength(0); fila++)
                for (int columna = 0; columna < bidimensional.GetLength(1); columna++)
                    bidimensional[fila, columna] = valor++;
            long TBidim = cronometro.ElapsedTicks;

            //Llenando un arreglo de arreglos
            valor = 0;
            cronometro.Reset();
            cronometro.Start();
            for (int conjunto = 0; conjunto < arreglo.Length; conjunto++)
                for (int subconjunto = 0; subconjunto < arreglo[conjunto].Length; subconjunto++)
                    arreglo[conjunto][subconjunto] = valor++;
            long TArreglo = cronometro.ElapsedTicks;

            //Imprime los tiempos
            Console.WriteLine("Tiempo arreglo bidimensional: " + TBidim.ToString());
            Console.WriteLine("Tiempo arreglo de arreglos: " + TArreglo.ToString());
        }
    }
}

```

```

Tiempo arreglo bidimensional: 3145
Tiempo arreglo de arreglos: 345

C:\Users\engin\source\repos\Ejemplo\Ejemplo\bin\Release\
net8.0\Ejemplo.exe (proceso 18760) se cerró con el código 0.

Para cerrar automáticamente la consola cuando se detiene la depuración, habilite Herramientas ->Opciones ->Depuración ->Cerrar la consola automáticamente al detenerse la depuración.

Presione cualquier tecla para cerrar esta ventana. . .

```

Ilustración 99: Métrica de velocidad: arreglo bidimensional vs arreglo de arreglos

```

using System.Diagnostics;

namespace Ejemplo {
    internal class Program {
        static void Main() {
            Random Azar = new Random();

            //Límite de todos los arreglos
            int Limite = 10;

            //Arreglos
            int[,] bidimensional = new int[Limite, Limite];
            int[,,] tridimensional = new int[Limite, Limite, Limite];
            int[,,,] cuatrodimensiones = new int[Limite, Limite, Limite, Limite];
            int[,,,,] cincodimensiones = new int[Limite, Limite, Limite, Limite, Limite];

            //Medidor de tiempos
            Stopwatch cronometro = new Stopwatch();

            //Evitar optimización de ciclos, haciendo que acumule los valores
            long Acumula = 0;

            //Llenando un arreglo bidimensional
            int valor = 0;
            cronometro.Reset();
            cronometro.Start();
            for (int a = 0; a < bidimensional.GetLength(0); a++)
                for (int b = 0; b < bidimensional.GetLength(1); b++)
                    bidimensional[a, b] = valor++;
            long TBidim = cronometro.ElapsedTicks;

            int posA = Azar.Next(bidimensional.GetLength(0));
            int posB = Azar.Next(bidimensional.GetLength(1));
            Acumula += bidimensional[posA, posB];

            //Llenando un arreglo tridimensional
            valor = 0;
            cronometro.Reset();
            cronometro.Start();
            for (int a = 0; a < tridimensional.GetLength(0); a++)
                for (int b = 0; b < tridimensional.GetLength(1); b++)
                    for (int c = 0; c < tridimensional.GetLength(2); c++)
                        tridimensional[a, b, c] = valor++;
            long TTridim = cronometro.ElapsedTicks;

            posA = Azar.Next(tridimensional.GetLength(0));
            posB = Azar.Next(tridimensional.GetLength(1));
            int posC = Azar.Next(tridimensional.GetLength(2));
            Acumula += tridimensional[posA, posB, posC];

            //Llenando un arreglo de cuatro dimensiones
            valor = 0;
            cronometro.Reset();
            cronometro.Start();
            for (int a = 0; a < cuatrodimensiones.GetLength(0); a++)
                for (int b = 0; b < cuatrodimensiones.GetLength(1); b++)
                    for (int c = 0; c < cuatrodimensiones.GetLength(2); c++)
                        for (int d = 0; d < cuatrodimensiones.GetLength(3); d++)
                            cuatrodimensiones[a, b, c, d] = valor++;
            long TCuatrodim = cronometro.ElapsedTicks;

            posA = Azar.Next(cuatrodimensiones.GetLength(0));
            posB = Azar.Next(cuatrodimensiones.GetLength(1));
            posC = Azar.Next(cuatrodimensiones.GetLength(2));
            int posD = Azar.Next(cuatrodimensiones.GetLength(3));
            Acumula += cuatrodimensiones[posA, posB, posC, posD];

            //Llenando un arreglo de cinco dimensiones
            valor = 0;
            cronometro.Reset();
            cronometro.Start();
            for (int a = 0; a < cincodimensiones.GetLength(0); a++)
                for (int b = 0; b < cincodimensiones.GetLength(1); b++)
                    for (int c = 0; c < cincodimensiones.GetLength(2); c++)
                        for (int d = 0; d < cincodimensiones.GetLength(3); d++)
                            cincodimensiones[a, b, c, d, e] = valor++;
        }
    }
}

```

```

        for (int e = 0; e < cincodimensiones.GetLength(4); e++)
            cincodimensiones[a, b, c, d, e] = valor++;
    long TCincodim = cronometro.ElapsedTicks;

    posA = Azar.Next(cincodimensiones.GetLength(0));
    posB = Azar.Next(cincodimensiones.GetLength(1));
    posC = Azar.Next(cincodimensiones.GetLength(2));
    posD = Azar.Next(cincodimensiones.GetLength(3));
    int posE = Azar.Next(cincodimensiones.GetLength(4));
    Acumula += cincodimensiones[posA, posB, posC, posD, posE];

    //Imprime los tiempos
    Console.WriteLine("Acumulado: " + Acumula.ToString());
    Console.WriteLine("Tiempo arreglo bidimensional: " + TBidim.ToString());
    Console.WriteLine("Tiempo arreglo tridimensional: " + TTridim.ToString());
    Console.WriteLine("Tiempo arreglo cuatro dimensiones: " + TCuartrodim.ToString());
    Console.WriteLine("Tiempo arreglo cinco dimensiones: " + TCincodim.ToString());
}
}
}

```

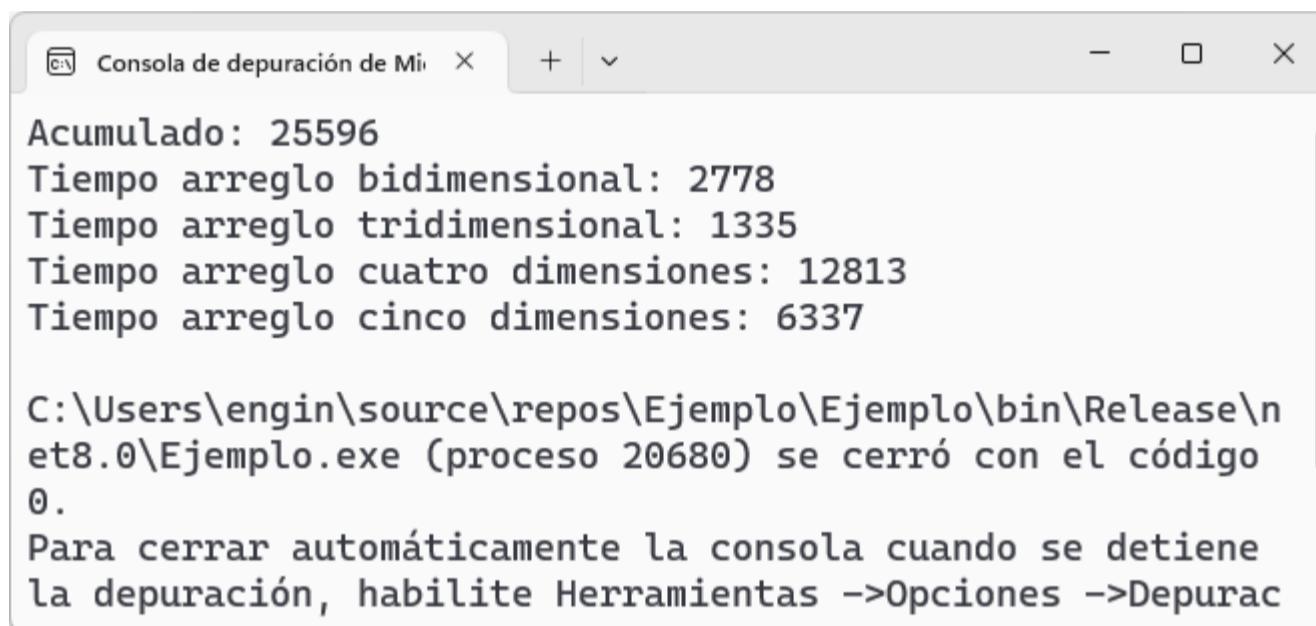


Ilustración 100: Métrica de velocidad: arreglos multidimensionales

Es interesante observar que un arreglo tridimensional es más rápido que un arreglo bidimensional, o un arreglo de cinco dimensiones es más rápido que uno de cuatro dimensiones. ¿Por qué pasa esto que desafía toda lógica? El sospechoso habitual es el compilador que optimiza los bloques de código en modo “release” de C#. Para evitar esto, se hizo uso de la variable Acumula y se suma alguna posición al azar de cada arreglo, pero, persiste esas métricas de tiempo extrañas. ¿Qué puede ser?

Cambiando a Limite = 30 en el código, estos son los valores obtenidos:

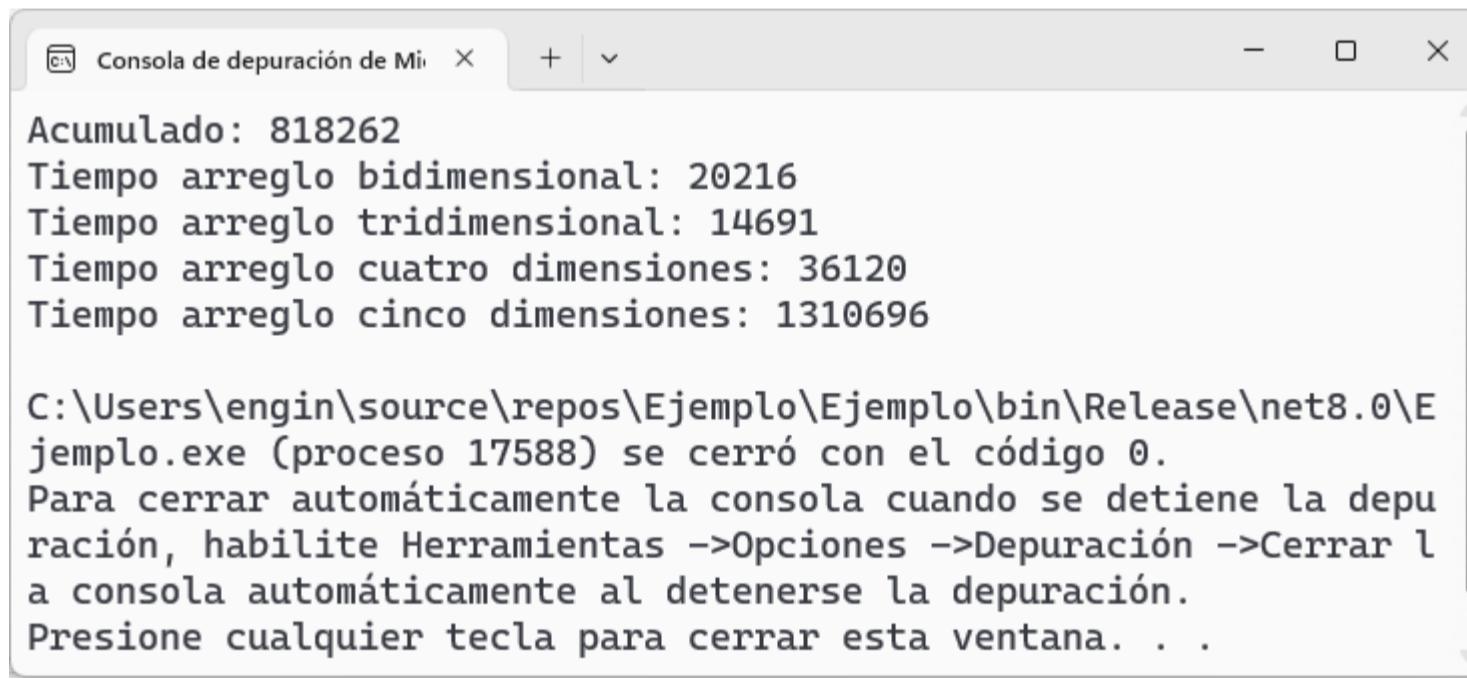


Ilustración 101: Métrica de velocidad: arreglos multidimensionales. Limite = 30

En este caso es más lógico que el arreglo de cinco dimensiones demore más que el arreglo de cuatro dimensiones. Pero el dato extraño de arreglo tridimensional más rápido que el arreglo bidimensional, es incómodo. ¿Qué puede estar pasando? Hay dos sospechosos, el primero es el Garbage Collector (recolector de basura, proceso independiente que se encarga de liberar la memoria no usada) y este proceso es recomendado no manipularlo; el segundo sospechoso es ver como se implementan los arreglos bidimensionales y tridimensionales en código precompilado (p-code). La diferencia en velocidad no es alta, pero si está haciendo una aplicación donde es prioritaria la velocidad, hay que hacer múltiples pruebas e implementaciones y no concluir (antes de probar) que algo es más rápido porque “suena lógico”.

Tipo implícito de arreglo

Al inicializar un arreglo que se ha declarado de tipo "var", este es capaz de determinar el tipo de dato al inicializarlo.

C/018.cs

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Tipo implícito de arreglo
            var arregloA = new[] { 16, 83, 29, 29 };
            var arregloB = new[] { 'R', 'a', 'f', 'a', 'e', 'l' };
            var arregloC = new[] { "Esta", "es", "una", "prueba" };
            var arregloD = new[] { true, false, false, true };
            var arregloE = new[] { 3.1, 8.9, 2.3 };

            //Imprime los tipos
            Console.WriteLine("Tipo ArregloA: " + arregloA.GetType().ToString());
            Console.WriteLine("Tipo ArregloB: " + arregloB.GetType().ToString());
            Console.WriteLine("Tipo ArregloC: " + arregloC.GetType().ToString());
            Console.WriteLine("Tipo ArregloD: " + arregloD.GetType().ToString());
            Console.WriteLine("Tipo ArregloE: " + arregloE.GetType().ToString());
        }
    }
}
```

```
Tipo ArregloA: System.Int32[]
Tipo ArregloB: System.Char[]
Tipo ArregloC: System.String[]
Tipo ArregloD: System.Boolean[]
Tipo ArregloE: System.Double[]

C:\Users\engin\source\repos\Ejemplo\Ejemplo\b
in\Release\net8.0\Ejemplo.exe (proceso 20352)
se cerró con el código 0.
Para cerrar automáticamente la consola cuando
```

Ilustración 102: Tipo implícito de arreglo

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Tipo implícito en arreglo de arreglos
            var arregloA = new[] {
                new[] { 16, 83, 29, 29 },
                new[] { 72, 6, 26 }
            };

            var arregloB = new[] {
                new[] { 'a', 'e', 'i', 'o' },
                new[] { 'q', 'w', 'e' },
                new[] { 'r', 't', 'y', 'u', 'o', 'p' }
            };

            //Imprime los tipos
            Console.WriteLine("Tipo ArregloA: " + arregloA.GetType().ToString());
            Console.WriteLine("Tipo ArregloB: " + arregloB.GetType().ToString());
        }
    }
}
```

```
Tipo ArregloA: System.Int32[][]  
Tipo ArregloB: System.Char[][]  
  
C:\Users\engin\source\repos\Ejemplo\Ejemplo\b...  
se cerró con el código 0.  
Para cerrar automáticamente la consola cuando se detiene la depuración, habilite Herramientas ->Opciones ->Depuración ->Cerrar la consola automáticamente al detenerse la depuración
```

Ilustración 103: Tipo implícito en arreglo de arreglos

Convertir una cadena en un arreglo de cadenas al dividirla

C/020.cs

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Una cadena
            string Cadena = "Esta es una frase para probar la división por palabras";

            //Se divide en un arreglo de palabras
            string[] Palabras = Cadena.Split(' ');

            foreach (string elemento in Palabras) {
                Console.WriteLine("[" + elemento + "]");
            }
        }
    }
}
```

```
[Esta]
[es]
[una]
[frase]
[para]
[probar]
[la]
[división]
[por]
[palabras]

C:\Users\engin\source\repos\Ejemplo\Ejemplo\bin\Release\net8.0\Ejemplo.exe (proceso 11856) se cerró con el código 0.
```

Ilustración 104: Convertir una cadena en un arreglo de cadenas al dividirla

Sin embargo, si hay más de un espacio intermedio.

C/021.cs

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Una cadena
            string Cadena = " abc def ghij klmn opqrstu vw x yz ";

            //Se divide en un arreglo de palabras
            string[] Palabras = Cadena.Split(' ');

            //¡OJO! que cuando hay más de un espacio intermedio, este se interpreta como una palabra
            foreach (string elemento in Palabras) {
                Console.WriteLine("[" + elemento + "]");
            }
        }
    }
}
```

```
Consola de depuración de Mi... X + - □ ×
[]
[]
[abc]
[]
[]
[def]
[]
[]
[ghij    klmn]
[]
[]
[opqrstuvwxyz]
[]
[]
[vw]
[]
[x]
[]
[yz]
[]
[]

C:\Users\engin\source\repos\Ejemplo\Ejemplo\bin\Release\net
8.0\Ejemplo.exe (proceso 19400) se cerró con el código 0.
```

Ilustración 105: Cuando hay más de un espacio intermedio

Parte 4: Programación Orientada a Objetos

Definir una clase

Al definir una clase en C#, se pueden hacer uso de atributos privados (con la palabra reservada private), atributos públicos (con la palabra reservada public, pero no es recomendado), métodos privados y públicos.

D/001.cs

```
namespace Ejemplo {
    //Esta es una clase propia con sus atributos y métodos (encapsulación)
    internal class MiClase {
        //Atributos privados
        private int Numero;
        private char Letra;
        private string Cadena;
        private double Valor;

        //Atributos públicos (no recomendado)
        public int Acumula;
        public char Caracter;

        //Métodos privados
        private double Maximo(double numA, double numB, double numC) {
            double max = numA;
            if (max < numB) max = numB;
            if (max < numC) max = numC;
            return max;
        }

        //Métodos públicos
        public double CalculaPromedio(double numA, double numB, double numC) {
            return (numA + numB + numC) / 3;
        }
    }

    //Inicia la aplicación aquí
    internal class Program {
        static void Main() {
            //Instancia o crea un objeto de MiClase
            MiClase Objeto = new MiClase();

            //Solo puede llamar al método público de MiClase
            double resultado = Objeto.CalculaPromedio(1, 7, 8);
            Console.WriteLine(resultado.ToString());
        }
    }
}
```

```
Consola de depuración de MiClase + - X
5,333333333333333333

C:\Users\engin\source\repos\Ejemplo\Ejemplo\bin\Release\net8.0\Ejemplo.exe (proceso 1152) se cerró con el código 0.

Para cerrar automáticamente la consola cuando se detiene la depuración, habilite Herramientas ->Opciones ->Depuración ->Cerrar la consola automáticamente al de tenerse la depuración.
```

Ilustración 106: Atributos/Métodos privados y públicos

Errores al tratar de acceder a atributos o métodos privados

Si se intenta hacer uso de un método privado o acceder a un atributo privado, se generará un error en tiempo de compilación.

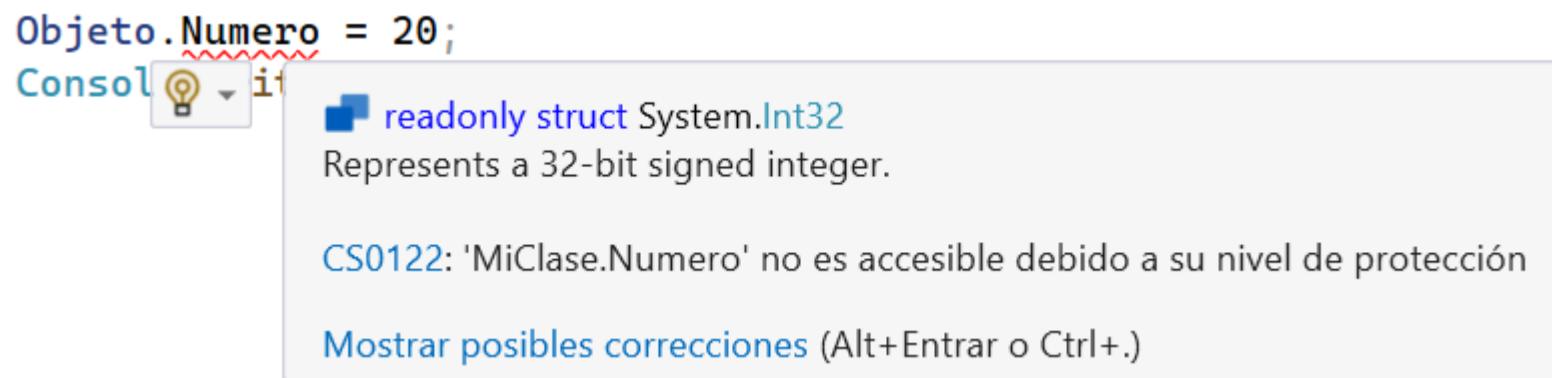


Ilustración 107: Intento de acceder a un atributo privado

Los atributos deben ser privados. Accediendo a ellos.

La recomendación es que los atributos de una clase siempre sean privados. Así que, para acceder a ellos desde una instancia, se debe hacer a través de métodos públicos de lectura y escritura. Esos métodos son conocidos en el medio como getters y setters.

Cada atributo se le crea un método que en su interior tiene el "get" (para leer) o el "set" (para darle valor)

D/002.cs

```
namespace Ejemplo {
    //Esta es una clase propia con sus atributos y métodos (encapsulación)
    class MiClase {
        //Atributos privados
        private int numero;
        private char letra;
        private string cadena;
        private double valor;

        //Los getters y setters
        public int Numero { get => numero; set => numero = value; }
        public char Letra { get => letra; set => letra = value; }
        public string Cadena { get => cadena; set => cadena = value; }
        public double Valor { get => valor; set => valor = value; }
    }

    //Inicia la aplicación aquí
    class Program {
        static void Main() {
            //Instancia o crea un objeto de MiClase
            MiClase Objeto = new MiClase();

            //Llama los setters
            Objeto.Cadena = "Suini, Capuchina, Grisú, Milú, Sally, Vikingo";
            Objeto.Numero = 7;
            Objeto.Letra = 'R';
            Objeto.Valor = 93.5;

            //Usa los getters
            Console.WriteLine(Objeto.Letra.ToString());
            Console.WriteLine(Objeto.Valor.ToString());
            Console.WriteLine(Objeto.Cadena);
            Console.WriteLine(Objeto.Numero.ToString());
        }
    }
}
```

```
R
93,5
Suini, Capuchina, Grisú, Milú, Sally, Vikingo
7

C:\Users\engin\source\repos\Ejemplo\Ejemplo\binn\Release\net8.0\Ejemplo.exe (proceso 6008) se
cerró con el código 0.
Para cerrar automáticamente la consola cuando
```

Ilustración 108: Los atributos deben ser privados. Accediendo a ellos.

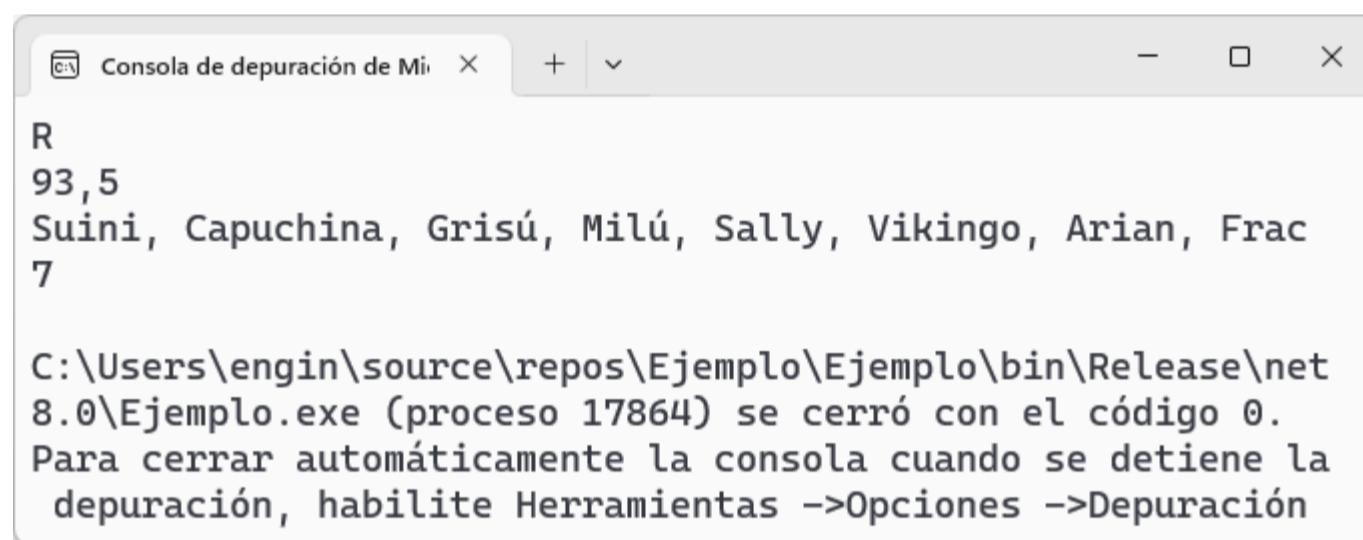
Más información en: <https://learn.microsoft.com/es-es/dotnet/csharp/programming-guide/classes-and-structs/properties>

```
namespace Ejemplo {
    //Esta es una clase propia con sus atributos y métodos (encapsulación)
    class MiClase {
        //Otra forma de definir atributos con los getters y setters
        public int Numero { get; set; }
        public char Letra { get; set; }
        public string Cadena { get; set; }
        public double Valor { get; set; }
    }

    //Inicia la aplicación aquí
    class Program {
        static void Main() {
            //Instancia o crea un objeto de MiClase
            MiClase Objeto = new MiClase();

            //Llama los setters
            Objeto.Cadena = "Suini, Capuchina, Grisú, Milú, Sally, Vikingo, Arian, Frac";
            Objeto.Numero = 7;
            Objeto.Letra = 'R';
            Objeto.Valor = 93.5;

            //Usa los getters
            Console.WriteLine(Objeto.Letra.ToString());
            Console.WriteLine(Objeto.Valor.ToString());
            Console.WriteLine(Objeto.Cadena);
            Console.WriteLine(Objeto.Numero.ToString());
        }
    }
}
```



```
R
93,5
Suini, Capuchina, Grisú, Milú, Sally, Vikingo, Arian, Frac
7
```

```
C:\Users\engin\source\repos\Ejemplo\Ejemplo\bin\Release\net
8.0\Ejemplo.exe (proceso 17864) se cerró con el código 0.
Para cerrar automáticamente la consola cuando se detiene la
depuración, habilite Herramientas ->Opciones ->Depuración
```

Ilustración 109: Forma reducida de los getters y setters

```

namespace Ejemplo {
    //Esta es una clase propia con sus atributos y métodos (encapsulación)
    class MiClase {
        //Atributos privados. Un uso de los getters y setters
        private int numero;
        private char letra;
        private string cadena;
        private double valor;

        //Puede auditar cuando se leyó o cambió el valor de un atributo
        public int Numero {
            get {
                Console.WriteLine("Lee numero: " + DateTime.Now.ToString("yyyy-MM-dd h:mm:ss tt"));
                return numero;
            }
            set {
                Console.WriteLine("Cambia numero: " + DateTime.Now.ToString("yyyy-MM-dd h:mm:ss tt"));
                numero = value;
            }
        }

        public char Letra {
            get {
                Console.WriteLine("Lee letra: " + DateTime.Now.ToString("yyyy-MM-dd h:mm:ss tt"));
                return letra;
            }
            set {
                Console.WriteLine("Cambia letra: " + DateTime.Now.ToString("yyyy-MM-dd h:mm:ss tt"));
                letra = value;
            }
        }

        public string Cadena {
            get {
                Console.WriteLine("Lee cadena: " + DateTime.Now.ToString("yyyy-MM-dd h:mm:ss tt"));
                return cadena;
            }
            set {
                Console.WriteLine("Cambia cadena: " + DateTime.Now.ToString("yyyy-MM-dd h:mm:ss tt"));
                cadena = value;
            }
        }

        public double Valor {
            get {
                Console.WriteLine("Lee valor: " + DateTime.Now.ToString("yyyy-MM-dd h:mm:ss tt"));
                return valor;
            }
            set {
                Console.WriteLine("Cambia valor: " + DateTime.Now.ToString("yyyy-MM-dd h:mm:ss tt"));
                valor = value;
            }
        }
    }

    //Inicia la aplicación aquí
    class Program {
        static void Main() {
            //Instancia o crea un objeto de MiClase
            MiClase Objeto = new MiClase();

            //Llama los setters
            Objeto.Cadena = "Suini, Capuchina, Grisú, Milú, Sally, Vikingo";
            Objeto.Numero = 7;
            Objeto.Letra = 'R';
            Objeto.Valor = 93.5;

            //Usa los getters
            char unaletra = Objeto.Letra;
            double unvalor = Objeto.Valor;
            string unacadena = Objeto.Cadena;
            int unnumero = Objeto.Numero;

            Console.WriteLine("Letra es: " + unaletra.ToString());
        }
    }
}

```

```
        Console.WriteLine("Valor es: " + unvalor.ToString());
        Console.WriteLine("Cadena es: " + unacadena);
        Console.WriteLine("Número es: " + unnumero.ToString());
    }
}
```



The screenshot shows the 'Consola de depuración de Mi...' (Debug Console) window. It displays the following text output:

```
Cambia cadena: 2023-12-22 10:15:09 a. m.
Cambia numero: 2023-12-22 10:15:09 a. m.
Cambia letra: 2023-12-22 10:15:09 a. m.
Cambia valor: 2023-12-22 10:15:09 a. m.
Lee letra: 2023-12-22 10:15:09 a. m.
Lee valor: 2023-12-22 10:15:09 a. m.
Lee cadena: 2023-12-22 10:15:09 a. m.
Lee numero: 2023-12-22 10:15:09 a. m.
Letra es: R
Valor es: 93,5
Cadena es: Suini, Capuchina, Grisú, Milú, Sally, Vikingo
Número es: 7
```

Ilustración 110: Uso de los getters/setters

Otro uso de los getters y setters

Para validar el dato de entrada

D/005.cs

```
namespace Ejemplo {
    //Esta es una clase propia con sus atributos y métodos (encapsulación)
    class MiClase {
        //Atributos privados. Un uso de los getters y setters
        private int edad;

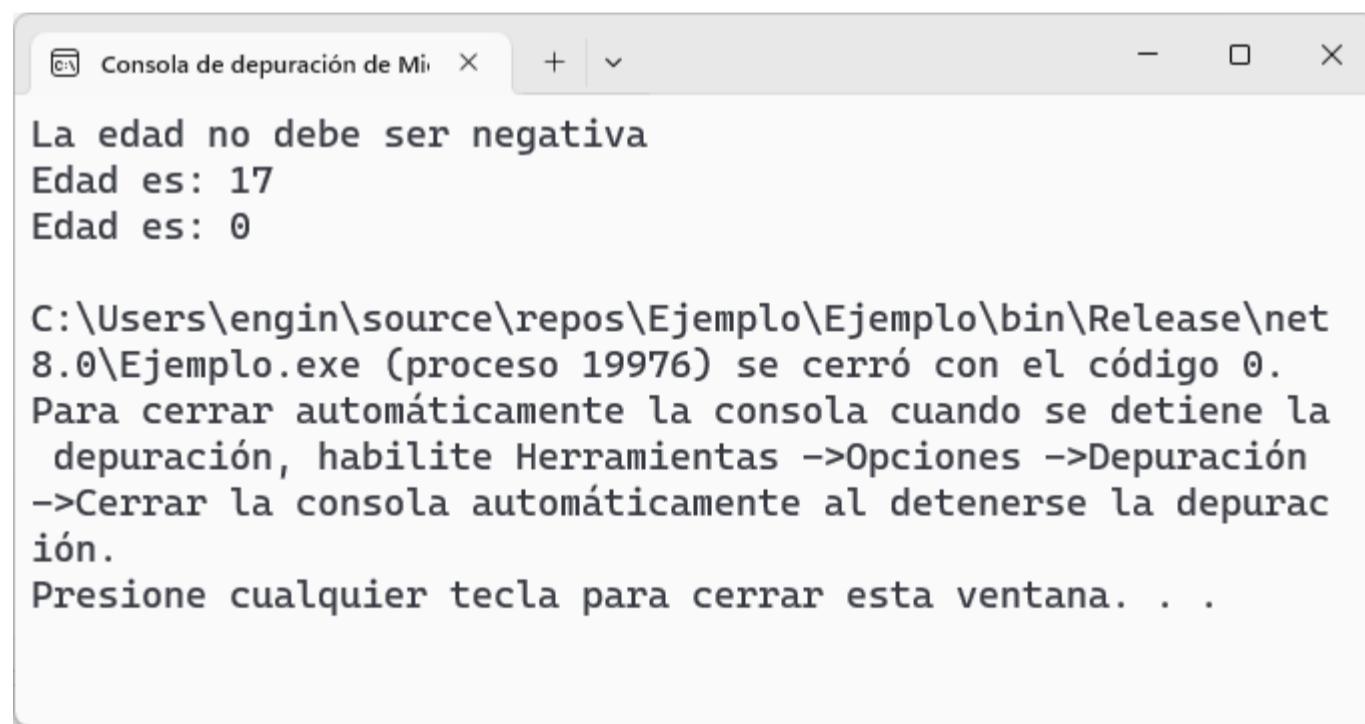
        //Puede validar el dato de inicialización
        public int Edad {
            get {
                return edad;
            }
            set {
                if (value < 0)
                    Console.WriteLine("La edad no debe ser negativa");
                else
                    edad = value;
            }
        }
    }

    //Inicia la aplicación aquí
    class Program {
        static void Main() {
            //Instancia o crea un objeto de MiClase
            MiClase Objeto = new MiClase();
            MiClase Otro = new MiClase();

            //Llama los setters
            Objeto.Edad = 17;
            Otro.Edad = -8;

            Console.WriteLine("Edad es: " + Objeto.Edad.ToString());
            Console.WriteLine("Edad es: " + Otro.Edad.ToString());
        }
    }
}
```

Al intentar dar un valor a un atributo el setter valida si ese dato es válido. Dado el caso, asigna el valor, de lo contrario, muestra un mensaje y el dato no es asignado.



```
Consola de depuración de Mi... + × - □ ×
La edad no debe ser negativa
Edad es: 17
Edad es: 0

C:\Users\engin\source\repos\Ejemplo\Ejemplo\bin\Release\net
8.0\Ejemplo.exe (proceso 19976) se cerró con el código 0.
Para cerrar automáticamente la consola cuando se detiene la
depuración, habilite Herramientas ->Opciones ->Depuración
->Cerrar la consola automáticamente al detenerse la depurac
ión.
Presione cualquier tecla para cerrar esta ventana. . .
```

Ilustración 111: Otro uso de los getters y setters

```

namespace Ejemplo {
    //Esta es una clase propia con sus atributos y métodos (encapsulación)
    class MiClase {
        //Un uso de los getters y setters
        private int numero;
        private char letra;
        private string cadena;
        private double valor;

        //Puede auditar cuando se leyó o cambió el valor de un atributo
        public int Numero {
            get {
                Console.WriteLine("Lee numero: " + DateTime.Now.ToString("yyyy-MM-dd h:mm:ss tt"));
                return numero;
            }
            set {
                Console.WriteLine("Cambia numero: " + DateTime.Now.ToString("yyyy-MM-dd h:mm:ss tt"));
                numero = value;
            }
        }

        public char Letra {
            get {
                Console.WriteLine("Lee letra: " + DateTime.Now.ToString("yyyy-MM-dd h:mm:ss tt"));
                return letra;
            }
            set {
                Console.WriteLine("Cambia letra: " + DateTime.Now.ToString("yyyy-MM-dd h:mm:ss tt"));
                letra = value;
            }
        }

        public string Cadena {
            get {
                Console.WriteLine("Lee cadena: " + DateTime.Now.ToString("yyyy-MM-dd h:mm:ss tt"));
                return cadena;
            }
            set {
                Console.WriteLine("Cambia cadena: " + DateTime.Now.ToString("yyyy-MM-dd h:mm:ss tt"));
                cadena = value;
            }
        }

        public double Valor {
            get {
                Console.WriteLine("Lee valor: " + DateTime.Now.ToString("yyyy-MM-dd h:mm:ss tt"));
                return valor;
            }
            set {
                Console.WriteLine("Cambia valor: " + DateTime.Now.ToString("yyyy-MM-dd h:mm:ss tt"));
                valor = value;
            }
        }
    }

    //Inicia la aplicación aquí
    class Program {
        static void Main() {
            //Instancia o crea un objeto de MiClase.
            //Otra forma de inicializar los atributos.
            MiClase Objeto = new MiClase {

                //Llama los setters
                Cadena = "Suini, Capuchina, Grisú, Milú, Sally, Vikingo",
                Numero = 7,
                Letra = 'R',
                Valor = 93.5
            };

            //Usa los getters
            char unaletra = Objeto.Letra;
            double unvalor = Objeto.Valor;
            string unacadena = Objeto.Cadena;
            int unnumero = Objeto.Numero;

            Console.WriteLine("Letra es: " + unaletra.ToString());
        }
    }
}

```

```
        Console.WriteLine("Valor es: " + unvalor.ToString());
        Console.WriteLine("Cadena es: " + unacadena);
        Console.WriteLine("Número es: " + unnumero.ToString());
    }
}
```



```
Cambia cadena: 2023-12-22 10:18:30 a. m.
Cambia numero: 2023-12-22 10:18:30 a. m.
Cambia letra: 2023-12-22 10:18:30 a. m.
Cambia valor: 2023-12-22 10:18:30 a. m.
Lee letra: 2023-12-22 10:18:30 a. m.
Lee valor: 2023-12-22 10:18:30 a. m.
Lee cadena: 2023-12-22 10:18:30 a. m.
Lee numero: 2023-12-22 10:18:30 a. m.
Letra es: R
Valor es: 93,5
Cadena es: Suini, Capuchina, Grisú, Milú, Sally, Vikingo
Número es: 7
```

Ilustración 112: Forma de inicializar los objetos llamando los setters

```

namespace Ejemplo {
    //Esta es una clase propia con sus atributos y métodos (encapsulación)
    class MiClase {
        //Otra forma de definir atributos con los getters y setters
        public int Numero { get; set; }
        public char Letra { get; set; }
        public string Cadena { get; set; }
        public double Valor { get; set; }
    }

    //Inicia la aplicación aquí
    class Program {
        static void Main() {
            //Instancia o crea un objeto de MiClase.
            MiClase Mascotas = new MiClase {
                Cadena = "Suini, Capuchina, Grisú, Milú, Sally, Vikingo",
                Numero = 7,
                Letra = 'R',
                Valor = 93.5
            };

            //Crea una variable de tipo MiClase
            MiClase otraVariable;

            //Asigna el primer objeto a esa variable
            otraVariable = Mascotas;

            //¿Qué sucede? Que hay dos variables apuntando al mismo objeto en memoria

            //Se imprimen los valores de ambas variables
            Console.WriteLine("Letra en Mascotas es: " + Mascotas.Letra.ToString());
            Console.WriteLine("Valor en Mascotas es: " + Mascotas.Valor.ToString());
            Console.WriteLine("Letra en otraVariable es: " + otraVariable.Letra.ToString());
            Console.WriteLine("Valor en otraVariable es: " + otraVariable.Valor.ToString());

            //Si se modifican los valores en otraVariable afecta a Mascotas
            //porque ambas apuntan al mismo objeto en memoria
            otraVariable.Valor = 12345.67;
            Console.WriteLine("Nuevo Valor en Mascotas es: " + Mascotas.Valor.ToString());
        }
    }
}

```

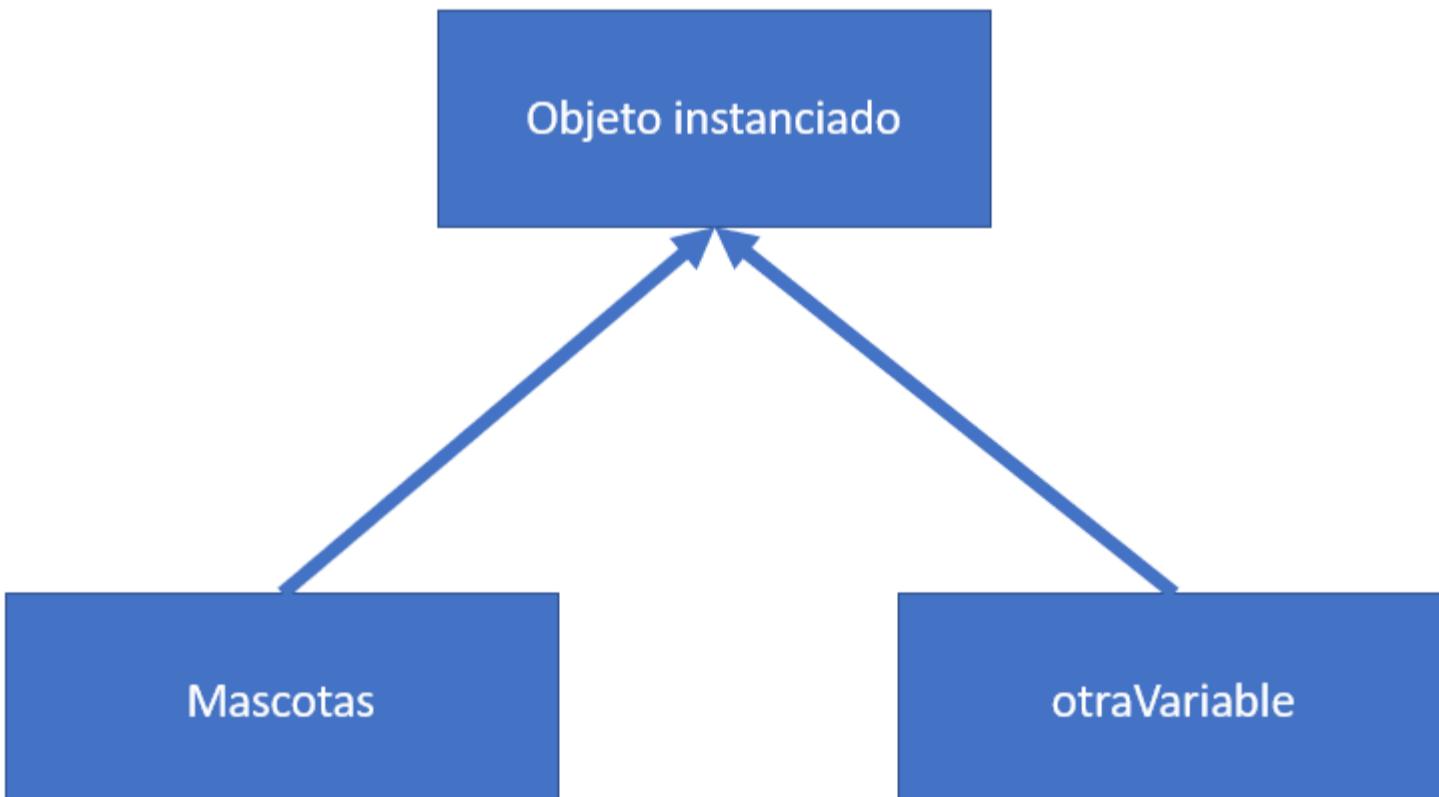


Ilustración 113: Ambas variables apuntan al mismo objeto instanciado

Dos variables apuntando al mismo objeto en memoria, eso es una copia superficial o "Shallow Copy".

```
Letra en Mascotas es: R
Valor en Mascotas es: 93,5
Letra en otraVariable es: R
Valor en otraVariable es: 93,5
Nuevo Valor en Mascotas es: 12345,67

C:\Users\engin\source\repos\Ejemplo\Ejemplo\bin\Release\net
8.0\Ejemplo.exe (proceso 20284) se cerró con el código 0.
Para cerrar automáticamente la consola cuando se detiene la
depuración, habilite Herramientas ->Opciones ->Depuración
->Cerrar la consola automáticamente al detenerse la depurac
ión.
Presione cualquier tecla para cerrar esta ventana. . .|
```

Ilustración 114: Dos variables refiriendo al mismo objeto

Sobrecarga de métodos

Dependiendo del número y tipo de parámetros C# sabe qué método usar así tenga el mismo nombre.

A continuación, una clase con tres métodos con el mismo nombre, pero cada uno de los métodos tiene diferente número de parámetros.

Por número de parámetros

D/008.cs

```
namespace Ejemplo {
    class Geometria {
        //Calcula el área del círculo
        public double Area(double radio) {
            return Math.PI * Math.Pow(radio, 2);
        }

        //Calcula el área del rectángulo
        public double Area(double baseR, double alturaR) {
            return baseR * alturaR;
        }

        //Calcula el área del triángulo
        public double Area(double ladoA, double ladoB, double ladoC) {
            double S = (ladoA + ladoB + ladoC) / 2;
            return Math.Sqrt(S * (S - ladoA) * (S - ladoB) * (S - ladoC));
        }
    }

    //Inicia la aplicación aquí
    class Program {
        static void Main() {
            //Instancia o crea un objeto Geometria
            Geometria geometria = new Geometria();

            //Dependiendo del número de parámetros llama a un método u otro
            double areaCirculo = geometria.Area(8);
            double areaTriangulo = geometria.Area(4, 5, 6);
            double areaRectangulo = geometria.Area(17, 19);

            Console.WriteLine("Área del círculo: " + areaCirculo.ToString());
            Console.WriteLine("Área del triángulo: " + areaTriangulo.ToString());
            Console.WriteLine("Área del rectángulo: " + areaRectangulo.ToString());
        }
    }
}
```

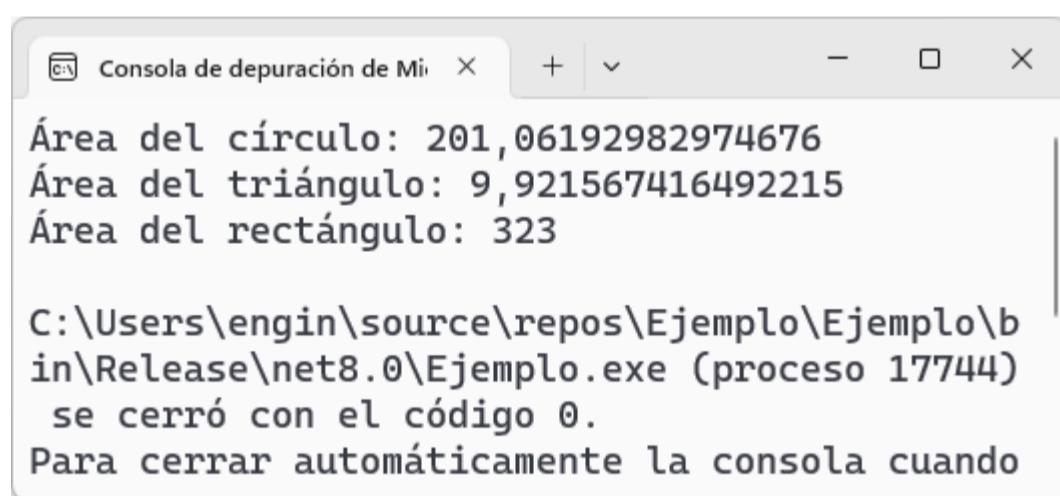


Ilustración 115: Sobrecarga de métodos

Por tipo de parámetros

C# selecciona el método dependiendo del tipo de parámetros. A continuación, una clase que implementa varios métodos con el mismo nombre, sólo que varía el tipo de parámetro.

D/009.cs

```
namespace Ejemplo {
    class MiClase {
        private int valor;
        private string cadena;
        private double costo;

        public void UnMetodo(int valor, string cadena) {
            this.valor = valor;
            this.cadena = cadena;
            Console.WriteLine("Un método B");
        }

        public void UnMetodo(string cadena, int valor) {
            this.cadena = cadena;
            this.valor = valor;
            Console.WriteLine("Segundo método");
        }

        public void UnMetodo(double costo, int valor) {
            this.costo = costo;
            this.valor = valor;
            Console.WriteLine("Tercer método");
        }

        public void UnMetodo(string cadena, double costo) {
            this.cadena = cadena;
            this.costo = costo;
            Console.WriteLine("Cuarto método");
        }
    }

    //Inicia la aplicación aquí
    class Program {
        static void Main() {
            //Instancia o crea un objeto
            MiClase objetoA = new MiClase();
            objetoA.UnMetodo(48, "Rafael");
            objetoA.UnMetodo("Alberto", 26);
            objetoA.UnMetodo(1994.06, 48);
            objetoA.UnMetodo("Moreno Parra", 1683.29);
        }
    }
}
```

```
Consola de depuración de Mi < X + - □ X
Un método B
Segundo método
Tercer método
Cuarto método

C:\Users\engin\source\repos\Ejemplo\Ejemplo\b
in\Release\net8.0\Ejemplo.exe (proceso 15188)
se cerró con el código 0.
```

Ilustración 116: Sobrecarga de métodos

Constructores

En C# los constructores se escriben con "public Nombre_de_la_clase". Los constructores tienen estas características:

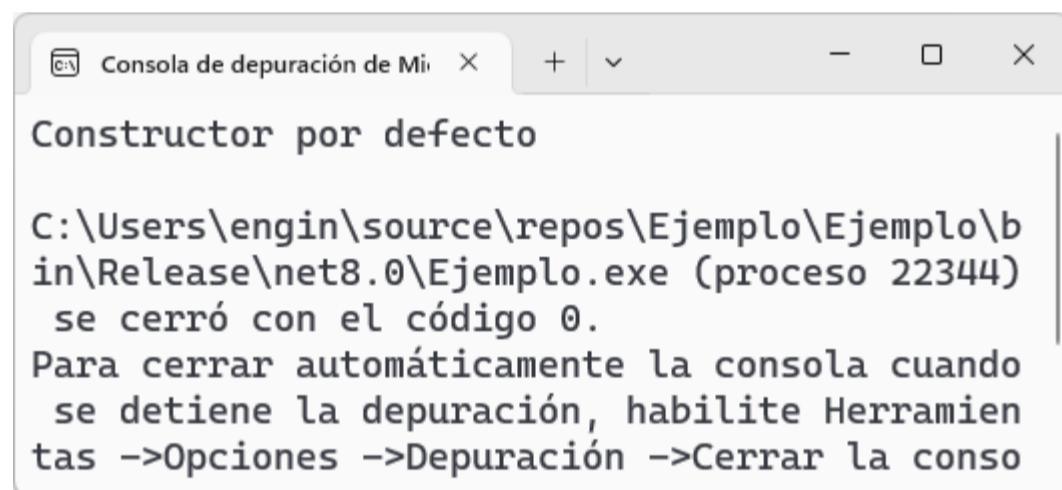
1. Deben tener el mismo nombre de la clase
2. Se ejecutan cuando el objeto es instanciado
3. No pueden retornar valores
4. Sólo ejecutan una sola vez (cuando el objeto se instancia)

Constructor sin parámetros

D/010.cs

```
namespace Ejemplo {
    class MiClase {
        public MiClase() {
            Console.WriteLine("Constructor por defecto");
        }
    }

    //Inicia la aplicación aquí
    class Program {
        static void Main() {
            //Instancia o crea un objeto
            MiClase instancia = new MiClase();
        }
    }
}
```



Constructor por defecto

C:\Users\engin\source\repos\Ejemplo\Ejemplo\b...
se cerró con el código 0.
Para cerrar automáticamente la consola cuando
se detiene la depuración, habilite Herramientas ->Opciones ->Depuración ->Cerrar la conso

Ilustración 117: Constructor sin parámetros

Constructor con parámetros

Se envía los datos del objeto al instanciarlo.

D/011.cs

```
namespace Ejemplo {
    //Esta es una clase propia con sus atributos y métodos (encapsulación)
    class MiClase {
        //Un constructor
        public MiClase(int Numero, char Letra, string Cadena, double Valor) {
            this.Numero = Numero; //Se asigna así this.atributo = valor parámetro
            this.Letra = Letra;
            this.Cadena = Cadena;
            this.Valor = Valor;
        }

        //Otra forma de definir atributos con los getters y setters
        public int Numero { get; set; }
        public char Letra { get; set; }
        public string Cadena { get; set; }
        public double Valor { get; set; }
    }

    //Inicia la aplicación aquí
    class Program {
        static void Main() {
            //Instancia o crea un objeto de MiClase llamando el constructor
            MiClase Mascotas = new MiClase(2016, 'T', "Tammy", 12.17);

            //Se imprimen los valores de ambas variables
            Console.WriteLine("Letra en Mascotas es: " + Mascotas.Letra.ToString());
            Console.WriteLine("Valor en Mascotas es: " + Mascotas.Valor.ToString());
        }
    }
}
```

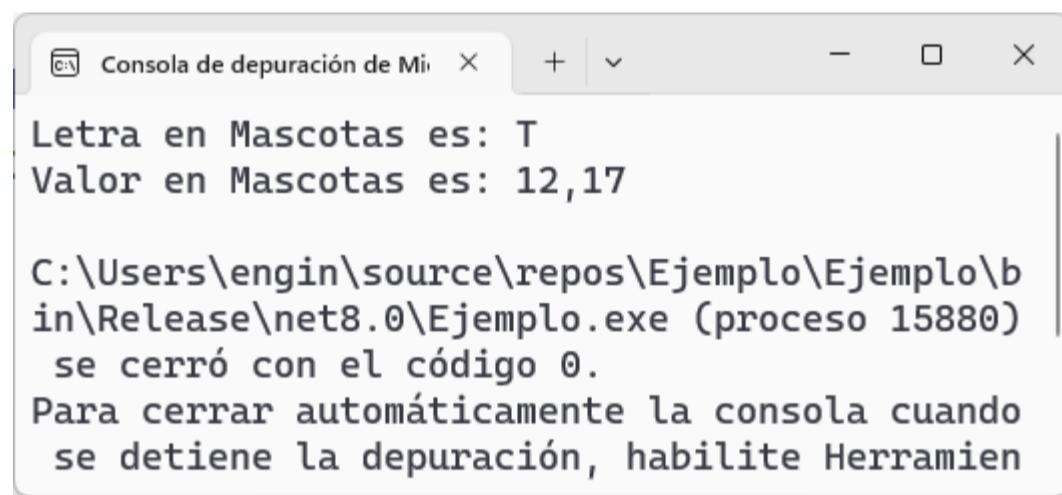


Ilustración 118: Constructor con parámetros

Sobrecarga de constructores

A continuación, una clase con varios constructores:

D/012.cs

```
namespace Ejemplo {
    class MiClase {
        private int valor;
        private string cadena;
        private double costo;
        public MiClase() {
            Console.WriteLine("Constructor por defecto");
        }

        public MiClase(int valor) {
            this.valor = valor;
            this.cadena = "por defecto";
            this.costo = 0;
            Console.WriteLine("Constructor B");
        }

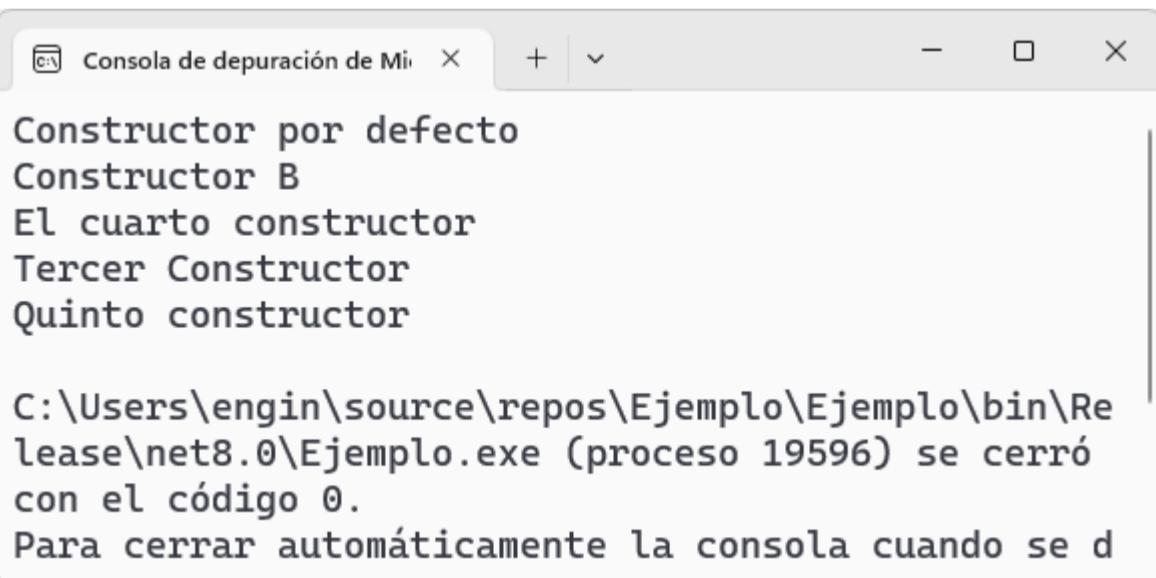
        public MiClase(string cadena, int valor) {
            this.cadena = cadena;
            this.valor = valor;
            this.costo = 0;
            Console.WriteLine("Tercer Constructor");
        }

        public MiClase(double costo, int valor) {
            this.cadena = "por defecto";
            this.costo = costo;
            this.valor = valor;
            Console.WriteLine("El cuarto constructor");
        }

        public MiClase(string cadena, double costo, int valor) {
            this.cadena = cadena;
            this.costo = costo;
            this.valor = valor;
            Console.WriteLine("Quinto constructor");
        }
    }

    //Inicia la aplicación aquí
    class Program {
        static void Main() {
            //Instancia o crea un objeto
            MiClase objetoA = new MiClase();
            MiClase objetoB = new MiClase(48);
            MiClase objetoC = new MiClase(1972.06, 26);
            MiClase objetoD = new MiClase("Ramp", 48);
            MiClase objetoE = new MiClase("Moreno Parra", 1683.29, 29);
        }
    }
}
```

Dependiendo del número de parámetros, se llama a un constructor o a otro



```
Constructor por defecto
Constructor B
El cuarto constructor
Tercer Constructor
Quinto constructor

C:\Users\engin\source\repos\Ejemplo\Ejemplo\bin\Release\net8.0\Ejemplo.exe (proceso 19596) se cerró con el código 0.

Para cerrar automáticamente la consola cuando se d
```

Ilustración 119: Sobrecarga de constructores

Usando el constructor para copiar objetos

La asignación de una variable objeto a otra variable objeto da como resultado que ambas variables apunten al mismo objeto en memoria, ese se le conoce como una copia superficial "Shallow Copy". ¿Cómo entonces generar una copia total del objeto, es decir, copiar los valores de los atributos también conocida como copia profunda o "Deep Copy"? Usualmente se busca el término "clonar el objeto", en el pasado, se usaba la instrucción ICloneable (considerada obsoleta o mejor no usarla: <https://stackoverflow.com/questions/536349/why-no-iclonable>). A continuación, se muestra una técnica para hacer una copia profunda:

Se pone un método al que se le puede llamar CopiarObjeto() que retorna una nueva instancia de la clase y se le envía por el **constructor** los valores que tienen en los atributos.

D/013.cs

```
namespace Ejemplo {
    //Esta es una clase propia con sus atributos y métodos (encapsulación)
    internal class MiClase {
        //Un constructor
        public MiClase(int Numero, char Letra, string Cadena, double Valor) {
            this.Numero = Numero; //Se asigna así this.atributo = valor parámetro
            this.Letra = Letra;
            this.Cadena = Cadena;
            this.Valor = Valor;
        }

        //Método que permite copiar un objeto
        public MiClase CopiarObjeto() {
            MiClase copia = new MiClase(Numero, Letra, Cadena, Valor);
            return copia;
        }

        //Otra forma de definir atributos con los getters y setters
        public int Numero { get; set; }
        public char Letra { get; set; }
        public string Cadena { get; set; }
        public double Valor { get; set; }
    }

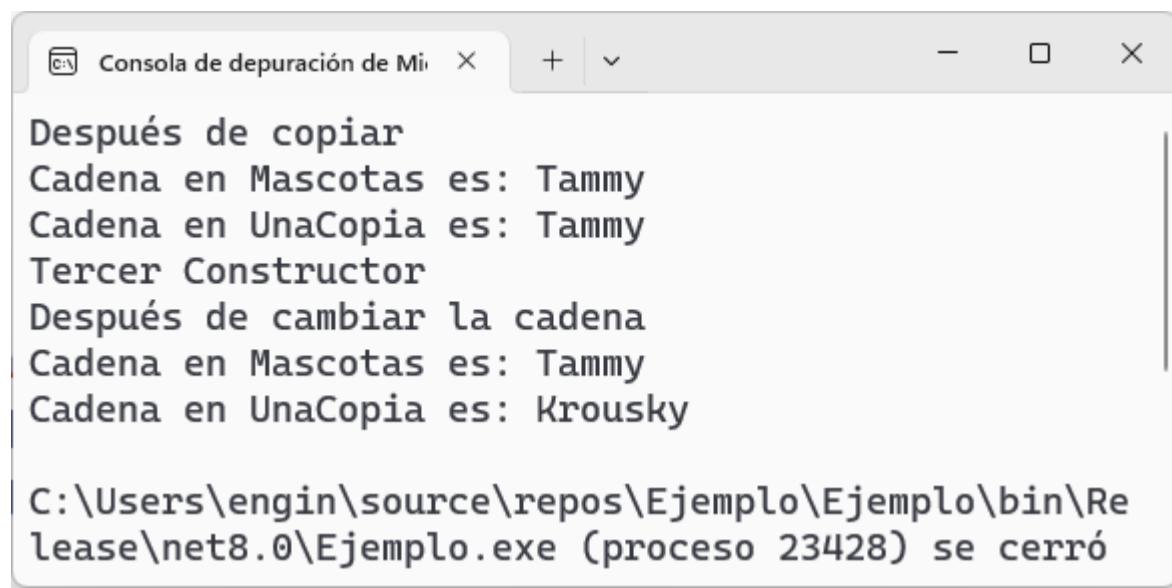
    //Inicia la aplicación aquí
    internal class Program {
        static void Main() {
            //Instancia o crea un objeto de MiClase llamando el constructor
            MiClase Mascotas = new MiClase(2016, 'T', "Tammy", 12.17);

            //Hace una copia de ese objeto
            MiClase UnaCopia = Mascotas.CopiarObjeto();

            //Se imprimen los valores de los dos objetos
            Console.WriteLine("Después de copiar");
            Console.WriteLine("Cadena en Mascotas es: " + Mascotas.Cadena);
            Console.WriteLine("Cadena en UnaCopia es: " + UnaCopia.Cadena);

            //Cambia el valor de cadena en UnaCopia
            UnaCopia.Cadena = "Krousky";

            //Imprime de nuevo los valores
            Console.WriteLine("\r\nDespués de cambiar la cadena");
            Console.WriteLine("Cadena en Mascotas es: " + Mascotas.Cadena);
            Console.WriteLine("Cadena en UnaCopia es: " + UnaCopia.Cadena);
        }
    }
}
```



```
Después de copiar
Cadena en Mascotas es: Tammy
Cadena en UnaCopia es: Tammy
Tercer Constructor
Después de cambiar la cadena
Cadena en Mascotas es: Tammy
Cadena en UnaCopia es: Krousky

C:\Users\engin\source\repos\Ejemplo\Ejemplo\bin\Release\net8.0\Ejemplo.exe (proceso 23428) se cerró
```

Ilustración 120: Usando el constructor para copiar objetos

Un constructor puede llamar a otros métodos

Al instanciar una clase, el constructor puede llamar a otros métodos.

D/014.cs

```
namespace Ejemplo {

    internal class MiClase {
        //Constructor
        public MiClase() {
            //Llama a otros métodos
            MetodoA();
            MetodoB();
        }

        public void MetodoA() {
            Console.WriteLine("Ha llamado el método A");
        }

        private void MetodoB() {
            Console.WriteLine("Ha llamado el método B");
        }
    }

    //Inicia la aplicación aquí
    internal class Program {
        static void Main() {
            MiClase objeto = new MiClase();
        }
    }
}
```

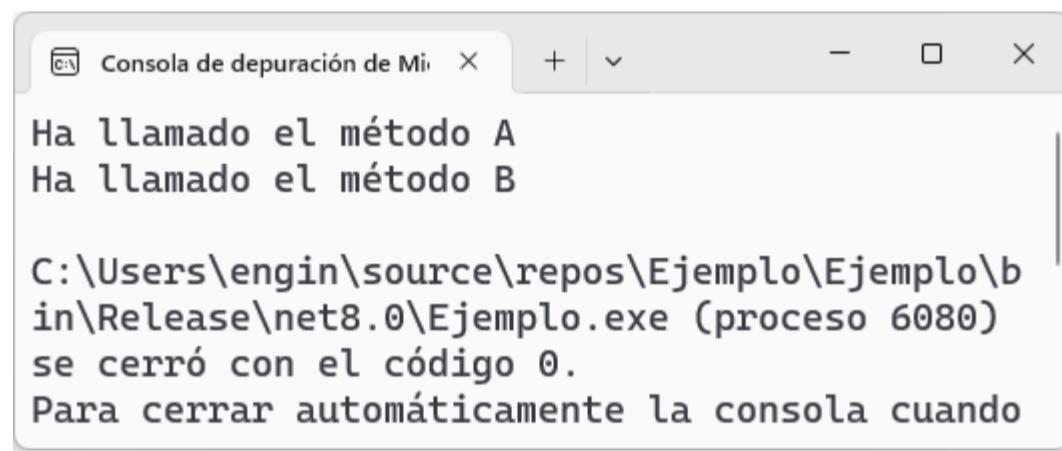


Ilustración 121: Un constructor puede llamar a otros métodos

En C# se implementa así: *nombre clase:clase madre*

D/015.cs

```

namespace Ejemplo {
    class Mascota {
        public string SerialChip { get; set; }
        public string Nombre { get; set; }
        public DateTime FechaNacimiento { get; set; }
        public char Sexo { get; set; } //Macho, Hembra
        public string Propietario { get; set; } //Nombre del propietario
        public string Telefono { get; set; } //Teléfono del propietario
        public string Correo { get; set; } //Correo electrónico del propietario
        public double Peso { get; set; }
        public int ColorOjos { get; set; } //0. Azul, 1. Verde, 2. Dorado, 3. Dispar
        public int EsperanzaVidaMinimo { get; set; }
        public int EsperanzaVidaMaximo { get; set; }
        public int NecesidadAtencion { get; set; } //0. Baja, 1. Media, 2. Alta
        public string Raza { get; set; }
    }

    class Gato:Mascota {
        public string PatronPelo { get; set; }
        public int TendenciaPerderPelo { get; set; }

        public string ReconocimientoCFA { get; set; } //Asociación de Criadores de Gatos
        public string ReconocimientoACFA { get; set; } //Asociación Americana de Criadores de Gatos
        public string ReconocimientoFIFe { get; set; } //Fédération Internationale Féline
        public string ReconocimientoTICA { get; set; } //Asociación Internacional de Gatos
    }

    class Perro:Mascota {
        //Real Sociedad Canina de España
        public string ReconocimientoRSCE { get; set; }

        //United Kennel Club
        public string ReconocimientoUKC { get; set; }

        //Crianza
        public string CriadoPara { get; set; }

        public double AlturaALaCruz { get; set; }
        public int TendenciaBabear { get; set; } //0. Ninguna, 1. Baja, 2. Moderada
        public int TendenciaRoncar { get; set; }
        public int TendenciaLadrar { get; set; }
        public int TendenciaExcavar { get; set; }
    }

    //Inicia la aplicación aquí
    class Program {
        static void Main() {
            Mascota objMascota = new Mascota();
            Gato objGato = new Gato();
            Perro objPerro = new Perro();

            //Da valores a la instancia de mascota
            objMascota.Correo = "enginelife@hotmail.com";
            objMascota.ColorOjos = 1;

            //Da valores a la instancia de gato
            objGato.Correo = "ramsoftware@gmail.com";
            objGato.Propietario = "Rafael Alberto Moreno Parra";
            objGato.Nombre = "Sally";
            objGato.Sexo = 'H';
            objGato.PatronPelo = "Tricolor";

            //Da valores a la instancia de perro
            objPerro.Raza = "Pastor Alemán";
            objPerro.Sexo = 'M';
            objPerro.Nombre = "Firuláis";
            objPerro.TendenciaLadrar = 1;
        }
    }
}

```

Clases abstractas y herencia

Una clase abstracta solo permite heredar, no se puede instanciar. Si se intenta instanciar dará un mensaje de error en tiempo de compilación. La palabra reservada "abstract" es para definir clases abstractas.

D/016.cs

```
namespace Ejemplo {

    //Esta clase solo puede heredar, no se puede instanciar
    abstract class Mascota {
        public string SerialChip { get; set; }
        public string Nombre { get; set; }
        public DateTime FechaNacimiento { get; set; }
        public char Sexo { get; set; } //Macho, Hembra
        public string Propietario { get; set; } //Nombre del propietario
        public string Telefono { get; set; } //Teléfono del propietario
        public string Correo { get; set; } //Correo electrónico del propietario
        public double Peso { get; set; }
        public int ColorOjos { get; set; } //0. Azul, 1. Verde, 2. Dorado, 3. Dispar
        public int EsperanzaVidaMinimo { get; set; }
        public int EsperanzaVidaMaximo { get; set; }
        public int NecesidadAtencion { get; set; } //0. Baja, 1. Media, 2. Alta
        public string Raza { get; set; }
    }

    class Perro : Mascota {
        //Real Sociedad Canina de España
        public string ReconocimientoRSCE { get; set; }

        //United Kennel Club
        public string ReconocimientoUKC { get; set; }

        //Crianza
        public string CriadoPara { get; set; }

        public double AlturaALaCruz { get; set; }
        public int TendenciaBabear { get; set; } //0. Ninguna, 1. Baja, 2. Moderada
        public int TendenciaRoncar { get; set; }
        public int TendenciaLadrar { get; set; }
        public int TendenciaExcavar { get; set; }
    }

    class Gato : Mascota {
        public string PatronPelo { get; set; }
        public int TendenciaPerderPelo { get; set; }

        public string ReconocimientoCFA { get; set; } //Asociación de Criadores de Gatos
        public string ReconocimientoACFA { get; set; } //Asociación Americana de Criadores de Gatos
        public string ReconocimientoFIFE { get; set; } //Fédération Internationale Féline
        public string ReconocimientoTICA { get; set; } //Asociación Internacional de Gatos
    }

    //Inicia la aplicación aquí
    internal class Program {
        static void Main() {
            Mascota objMascota = new Mascota();
            Gato objGato = new Gato();
            Perro objPerro = new Perro();

            //Da valores a la instancia de mascota
            objMascota.Correo = "enginelife@hotmail.com";
            objMascota.ColorOjos = 1;

            //Da valores a la instancia de gato
            objGato.Correo = "ramsoftware@gmail.com";
            objGato.Propietario = "Rafael Alberto Moreno Parra";
            objGato.Nombre = "Sally";
            objGato.Sexo = 'H';
            objGato.PatronPelo = "Tricolor";

            //Da valores a la instancia de perro
            objPerro.Raza = "Pastor Alemán";
            objPerro.Sexo = 'M';
            objPerro.Nombre = "Firuláis";
            objPerro.TendenciaLadrar = 1;
        }
    }
}
```

Nivel de protección en los métodos y atributos

Private

En la clase madre, los atributos o métodos con el atributo private no pueden ser usados por las clases hijas.

D/017.cs

```
namespace Ejemplo {

    //Clase madre con atributos privados
    class Mascota {
        private string Nombre { get; set; }
        private char Sexo { get; set; } //Macho, Hembra
        private string Propietario { get; set; } //Nombre del propietario
    }

    class Gato : Mascota {
        public string PatronPelo { get; set; }
        public int TendenciaPerderPelo { get; set; }
        public void DatosGato(string Nombre, char Sexo, string Propietario) {
            this.Nombre = Nombre;
            this.Sexo = Sexo;
            this.Propietario = Propietario;
        }
    }

    class Perro : Mascota {
        //Crianza
        public string CriadoPara { get; set; }
        public double AlturaALaCruz { get; set; }

        public void DatosPerro(string Nombre, char Sexo, string Propietario) {
            this.Nombre = Nombre;
            this.Sexo = Sexo;
            this.Propietario = Propietario;
        }
    }

    //Inicia la aplicación aquí
    internal class Program {
        static void Main() {
            Gato objGato = new Gato();
            Perro objPerro = new Perro();

            //Da valores a la instancia de gato
            objGato.DatosGato("Sally", 'H', "Rafael Moreno");

            //Da valores a la instancia de perro
            objPerro.DatosPerro("Kitty", 'H', "Chloe Perry");
        }
    }
}
```

```
public void DatosGato(string Nombre, char Sexo, string Propietario) {
    this.Nombre = Nombre;
    this.Sexo = Sexo;
    this.Propietario = Propietario;
}
```

 class System.String
Represents text as a sequence of UTF-16 code units.

CS0122: 'Mascota.Propietario' no es accesible debido a su nivel de protección

Ilustración 122: Los atributos o métodos con el atributo private no pueden ser usados por las clases hijas

Si la clase madre tiene atributos o métodos con la palabra "protected", significa que esos atributos pueden ser accedidos por las clases hijas, pero no pueden ser accedidos por instancias.

```
namespace Ejemplo {

    //Clase madre con atributos privados
    class Mascota {
        protected string Nombre { get; set; }
        protected char Sexo { get; set; } //Macho, Hembra
        protected string Propietario { get; set; } //Nombre del propietario
    }

    class Gato : Mascota {
        public string PatronPelo { get; set; }
        public int TendenciaPerderPelo { get; set; }
        public void DatosGato(string Nombre, char Sexo, string Propietario) {
            this.Nombre = Nombre;
            this.Sexo = Sexo;
            this.Propietario = Propietario;
        }
    }

    class Perro : Mascota {
        //Crianza
        public string CriadoPara { get; set; }
        public double AlturaALaCruz { get; set; }

        public void DatosPerro(string Nombre, char Sexo, string Propietario) {
            this.Nombre = Nombre;
            this.Sexo = Sexo;
            this.Propietario = Propietario;
        }
    }

    //Inicia la aplicación aquí
    internal class Program {
        static void Main() {
            Mascota objMascota = new Mascota();
            Gato objGato = new Gato();
            Perro objPerro = new Perro();

            //Da valores a la instancia de gato
            objGato.DatosGato("Sally", 'H', "Rafael Moreno");

            //Da valores a la instancia de perro
            objPerro.DatosPerro("Kitty", 'H', "Chloe Perry");

            //Intenta acceder a los métodos protegidos de Mascota
            objMascota.Nombre = "Milú";
        }
    }
}
```

//Intenta acceder a los métodos protegidos de Mascota
`objMascota.Nombre = "Milú";`

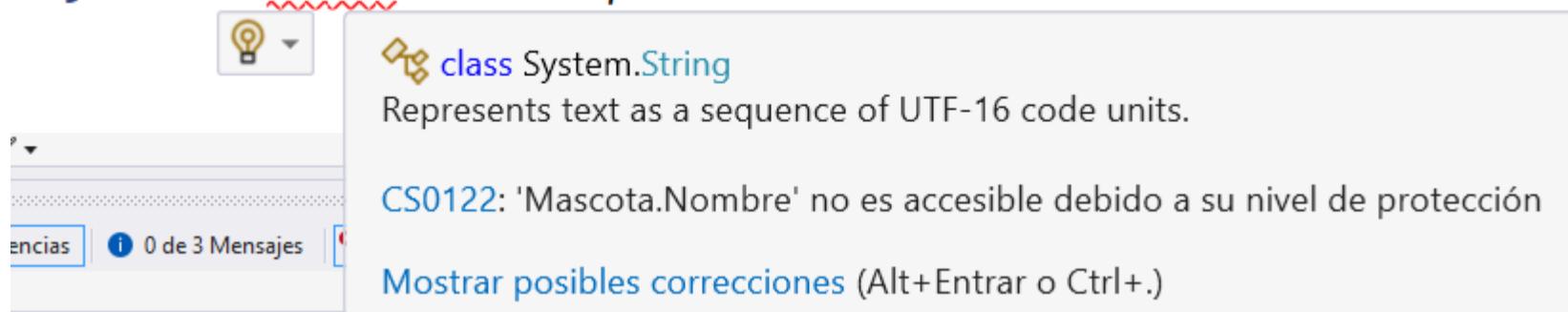


Ilustración 123: Atributos "protected" pueden ser accedidos por las clases hijas, pero no pueden ser accedidos por instancias

Cuando los atributos o métodos tienen la palabra “public”, entonces pueden ser accedidos por las clases hijas y también por las instancias.

```

namespace Ejemplo {

    //Clase madre con atributos privados
    class Mascota {
        public string Nombre { get; set; }
        public char Sexo { get; set; } //Macho, Hembra
        public string Propietario { get; set; } //Nombre del propietario
    }

    class Gato : Mascota {
        public string PatronPelo { get; set; }
        public int TendenciaPerderPelo { get; set; }
        public void DatosGato(string Nombre, char Sexo, string Propietario) {
            this.Nombre = Nombre;
            this.Sexo = Sexo;
            this.Propietario = Propietario;
        }
    }

    class Perro : Mascota {
        //Crianza
        public string CriadoPara { get; set; }
        public double AlturaALaCruz { get; set; }

        public void DatosPerro(string Nombre, char Sexo, string Propietario) {
            this.Nombre = Nombre;
            this.Sexo = Sexo;
            this.Propietario = Propietario;
        }
    }

    //Inicia la aplicación aquí
    internal class Program {
        static void Main() {
            Mascota objMascota = new Mascota();
            Gato objGato = new Gato();
            Perro objPerro = new Perro();

            //Da valores a la instancia de gato
            objGato.DatosGato("Sally", 'H', "Rafael Moreno");

            //Da valores a la instancia de perro
            objPerro.DatosPerro("Kitty", 'H', "Chloe Perry");

            //Intenta acceder a los métodos protegidos de Mascota
            objMascota.Nombre = "Milú";
        }
    }
}

```

Herencia y métodos iguales en clase madre e hija

¿Qué sucede si un método están en la clase madre y se escribe un método con el mismo nombre en la clase hija y luego se instancia la clase hija y se ejecuta ese método? El término se le conoce como polimorfismo.

D/020.cs

```
namespace Ejemplo {
    internal class Madre {
        public void Procedimiento() {
            Console.WriteLine("En la clase madre");
        }
    }

    internal class Hija:Madre {
        public new void Procedimiento() {
            Console.WriteLine("En la clase hija");
        }
    }

    //Inicia la aplicación aquí
    internal class Program {
        static void Main() {
            Hija objHija = new Hija();
            objHija.Procedimiento();
        }
    }
}
```

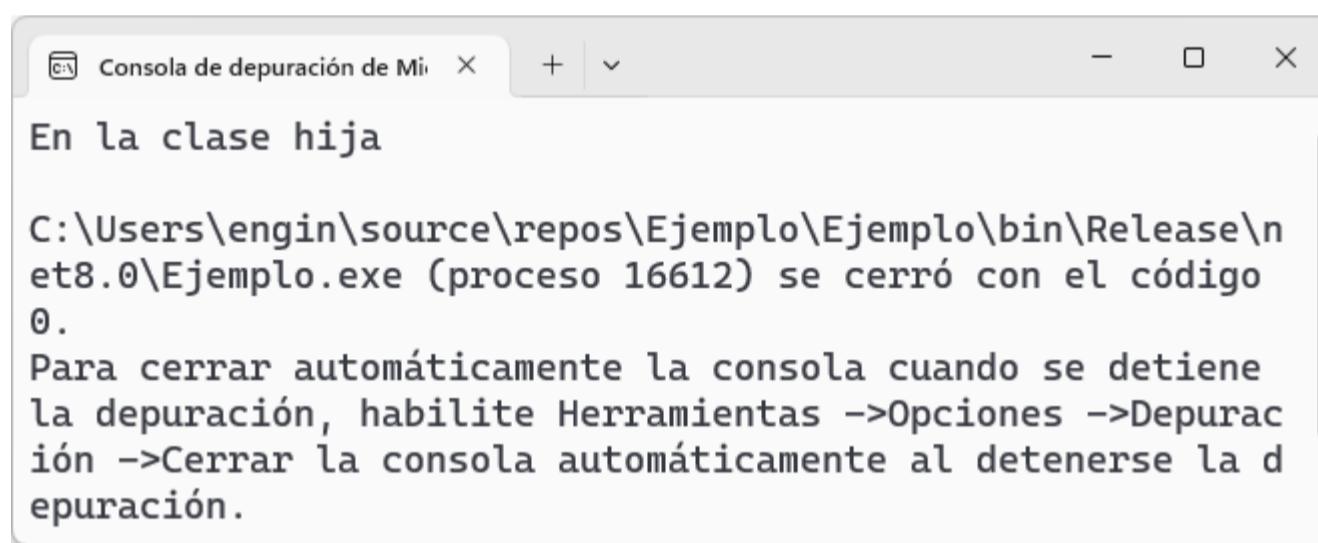


Ilustración 124: Herencia y métodos iguales en clase madre e hija

¿Qué sucede si la clase abuela o madre o hija tienen todos constructores? ¿Se ejecutan todos? ¿En qué orden?

D/021.cs

```
namespace Ejemplo {
    class Abuela {
        //Constructor
        public Abuela() {
            Console.WriteLine("Constructor de la clase abuela");
        }

        //Método
        public void Mostrar() {
            Console.WriteLine("Mostrar en Abuela");
        }
    }

    class Madre : Abuela {
        //Constructor
        public Madre() {
            Console.WriteLine("Constructor de la clase madre");
        }

        public new void Mostrar() {
            Console.WriteLine("Mostrar en Madre");
        }
    }

    class Hija : Madre {
        //Constructor
        public Hija() {
            Console.WriteLine("Constructor de la clase hija");
        }

        public new void Mostrar() {
            Console.WriteLine("Mostrar en Hija");
        }
    }

    //Inicia la aplicación aquí
    class Program {
        static void Main() {
            //Instancia a la hija
            Hija objHija = new Hija();

            //Ejecuta método
            objHija.Mostrar();
        }
    }
}
```

La ejecución del programa hace lo siguiente:

1. Ejecuta el constructor de la clase abuela
2. Ejecuta el constructor de la clase madre
3. Ejecuta el constructor de la clase hija
4. Si un método, diferente al constructor, tiene el mismo nombre en las clases abuela, madre e hija. Al ser ejecutado por la instancia de la clase hija, sólo ejecutará el método de la clase hija.

```
Constructor de la clase abuela
Constructor de la clase madre
Constructor de la clase hija
Mostrar en Hija

C:\Users\engin\source\repos\Ejemplo\Ejemplo\bin\Release\net8.0\Ejemplo.exe (proceso 13408) se cerró con el código 0.

Para cerrar automáticamente la consola cuando se detiene la depuración, habilite Herramientas ->Opciones ->Depuración ->Cerrar la consola automáticamente al de tenerse la depuración.
```

Ilustración 125: Constructores y herencia

Llamando a métodos de clases madres

Desde el método de la clase hija se hace uso de la instrucción "base" y así se llama al método de la clase madre

D/022.cs

```
namespace Ejemplo {
    internal class Abuela {
        //Constructor
        public Abuela() {
            Console.WriteLine("Constructor de la clase abuela");
        }

        //Método
        public void Mostrar() {
            Console.WriteLine("Mostrar en Abuela");
        }
    }

    internal class Madre : Abuela {
        //Constructor
        public Madre() {
            Console.WriteLine("Constructor de la clase madre");
        }

        public new void Mostrar() {
            base.Mostrar(); //Llama al método de la clase abuela
            Console.WriteLine("Mostrar en Madre");
        }
    }

    internal class Hija: Madre {
        //Constructor
        public Hija() {
            Console.WriteLine("Constructor de la clase hija");
        }

        public new void Mostrar() {
            base.Mostrar(); //Llama al método de la clase madre
            Console.WriteLine("Mostrar en Hija");
        }
    }

    //Inicia la aplicación aquí
    internal class Program {
        static void Main() {
            //Instancia a la hija
            Hija objHija = new Hija();

            //Ejecuta método
            objHija.Mostrar();
        }
    }
}
```

```
Consola de depuración de Mi... X + - □ ×
Constructor de la clase abuela
Constructor de la clase madre
Constructor de la clase hija
Mostrar en Abuela
Mostrar en Madre
Mostrar en Hija

C:\Users\engin\source\repos\Ejemplo\Ejemplo\bin
\Release\net8.0\Ejemplo.exe (proceso 19236) se
cerró con el código 0.
Para cerrar automáticamente la consola cuando s
e detiene la depuración, habilite Herramientas
->Opciones ->Depuración ->Cerrar la consola aut
```

Ilustración 126: Llamando a métodos de clases madres

Evitar la herencia

Con la palabra reservada "sealed" se evita que de esa clase se pueda heredar. Si se intenta habrá un error en tiempo de compilación.

D/023.cs

```
namespace Ejemplo {
    sealed class Madre {
        public void Aviso() {
            Console.WriteLine("Método en clase madre");
        }
    }

    class Hija : Madre {
        public void Mensaje() {
            Console.WriteLine("Método en clase hija");
        }
    }

    //Inicia la aplicación aquí
    class Program {
        static void Main() {
            Hija objHija = new Hija();
            objHija.Mensaje();
        }
    }
}
```

```
sealed class Madre {
    0 referencias
    public void Aviso() {
        Console.WriteLine("Método en clase madre");
    }
}
```

```
2 referencias
class Hija : Madre {
    1 referencia
    public void ...
    Console.
}
    class Ejemplo.Madre
    CS0509: 'Hija': no puede derivar del tipo sellado 'Madre'
    Mostrar posibles correcciones (Alt+Entrar o Ctrl+.)
```

Ilustración 127: Evitar la herencia

Clases estáticas

Una clase estática no requiere instanciarse para ser usada.

D/024.cs

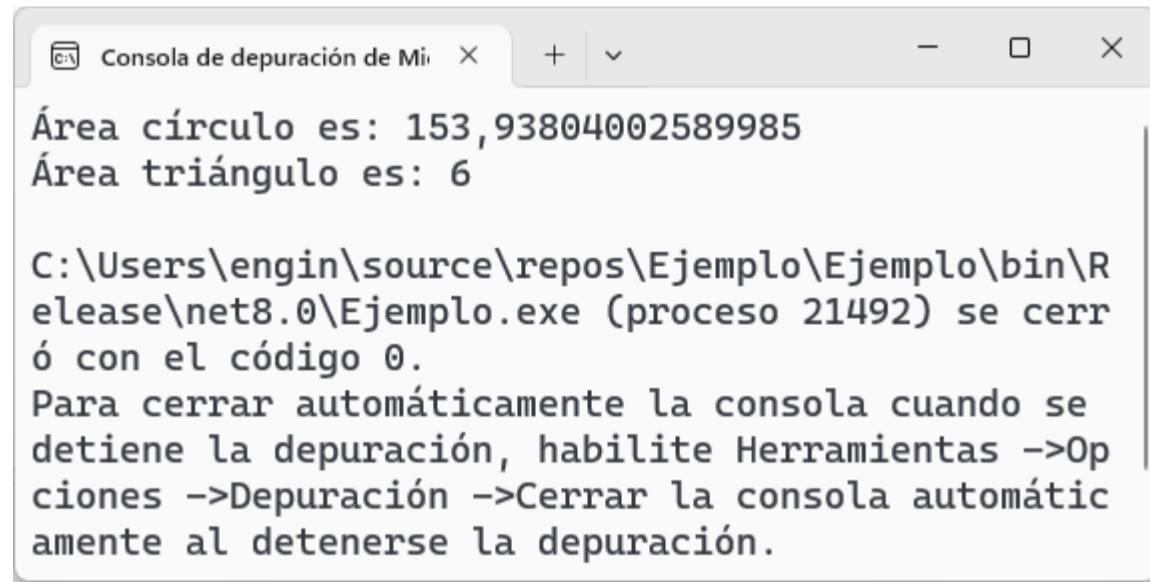
```
namespace Ejemplo {
    //Clase estática, no necesita instanciarse
    static class Geometria {
        //Todos los métodos deben ser static
        public static double AreaTriangulo(double baseT, double alturaT) {
            return baseT * alturaT / 2;
        }

        public static double AreaTriangulo(double ladoA, double ladoB, double ladoC) {
            double s = (ladoA + ladoB + ladoC) / 2;
            return Math.Sqrt(s * (s - ladoA) * (s - ladoB) * (s - ladoC));
        }

        public static double AreaCirculo(double radio) {
            return Math.PI*radio*radio;
        }
    }

    //Inicia la aplicación aquí
    internal class Program {
        static void Main() {
            //Hace uso de la clase sin instanciarla
            double unRadio = 7;
            double AreaUnCirculo = Geometria.AreaCirculo(unRadio);
            Console.WriteLine("Área círculo es: " + AreaUnCirculo.ToString());

            double AreaTri = Geometria.AreaTriangulo(3, 4, 5);
            Console.WriteLine("Área triángulo es: " + AreaTri.ToString());
        }
    }
}
```



The screenshot shows a command prompt window titled "Consola de depuración de Mi". The output displays the areas calculated by the static methods:

```
Área círculo es: 153,93804002589985
Área triángulo es: 6

C:\Users\engin\source\repos\Ejemplo\Ejemplo\bin\Release\net8.0\Ejemplo.exe (proceso 21492) se cerró con el código 0.
Para cerrar automáticamente la consola cuando se detiene la depuración, habilite Herramientas ->Opciones ->Depuración ->Cerrar la consola automáticamente al detenerse la depuración.
```

Ilustración 128: Clases estáticas

Métodos estáticos

Una clase tradicional puede tener métodos estáticos y estos métodos pueden ser accedidos sin necesidad de instanciar la clase, los otros métodos no estáticos si requieren que se instancie la clase.

D/025.cs

```
namespace Ejemplo {
    //Clase con métodos estáticos
    internal class Geometria {
        //Estos métodos estáticos pueden ser usados sin instanciar la clase
        public static double AreaTriangulo(double baseT, double alturaT) {
            return baseT * alturaT / 2;
        }

        public static double AreaTriangulo(double ladoA, double ladoB, double ladoC) {
            double s = (ladoA + ladoB + ladoC) / 2;
            return Math.Sqrt(s * (s - ladoA) * (s - ladoB) * (s - ladoC));
        }

        public static double AreaCirculo(double radio) {
            return Math.PI*radio*radio;
        }

        //Este método requiere instanciar la clase
        public double VolumenEsfera(double radio) {
            return 4 / 3 * Math.PI * Math.Pow(radio, 3);
        }
    }

    //Inicia la aplicación aquí
    internal class Program {
        static void Main() {
            double unRadio = 7;
            double AreaUnCirculo = Geometria.AreaCirculo(unRadio);
            Console.WriteLine("Área círculo es: " + AreaUnCirculo.ToString());

            double AreaTri = Geometria.AreaTriangulo(3, 4, 5);
            Console.WriteLine("Área triángulo es: " + AreaTri.ToString());

            //Instancio la clase
            Geometria objGeometria = new Geometria();
            double Esfera = objGeometria.VolumenEsfera(7);
            Console.WriteLine("Volumen Esfera: " + Esfera.ToString());
        }
    }
}
```

The screenshot shows a command prompt window titled 'Consola de depuración de Mi'. The output is as follows:

```
Área círculo es: 153,93804002589985
Área triángulo es: 6
Volumen Esfera: 1077,5662801812991

C:\Users\engin\source\repos\Ejemplo\Ejemplo\bin\Release\net8
.0\Ejemplo.exe (proceso 18036) se cerró con el código 0.
Para cerrar automáticamente la consola cuando se detiene la
depuración, habilite Herramientas ->Opciones ->Depuración ->
Cerrar la consola automáticamente al detenerse la depuración
.

Presione cualquier tecla para cerrar esta ventana. . .
```

Ilustración 129: Métodos estáticos

Constructor static

Para inicializar los atributos static de una clase, se hace uso de los constructores static. Una clase no static puede tener un constructor static y un constructor normal, el primero se ejecuta siempre al usarse la clase o instanciarse el objeto, en cambio, el constructor normal sólo se ejecuta al instanciarse la clase.

D/026.cs

```
namespace Ejemplo {
    //Clase con métodos estáticos
    class Geometria {
        public static int valorEntero;
        public static double valorReal;
        public static string unaCadena;

        //Constructor static (para inicializar atributos static)
        static Geometria() {
            valorEntero = 7;
            valorReal = 16.832;
            unaCadena = "Rafael";
            Console.WriteLine("Se ha ejecutado el constructor static");
        }

        //Constructor de clase
        public Geometria() {
            Console.WriteLine("Ejecuta el constructor de la clase");
        }

        //Este método estático puede ser usado sin instanciar la clase
        public static double AreaCirculo(double radio) {
            return Math.PI*radio*radio;
        }

        //Este método requiere instanciar la clase
        public double VolumenEsfera(double radio) {
            return 4 / 3 * Math.PI * Math.Pow(radio, 3);
        }
    }

    //Inicia la aplicación aquí
    class Program {
        static void Main() {
            //Accediendo a métodos estáticos
            double AreaUnCirculo = Geometria.AreaCirculo(7);
            Console.WriteLine("Área círculo es: " + AreaUnCirculo.ToString());

            //Accediendo a atributos estáticos
            Console.WriteLine("Cadena es: " + Geometria.unaCadena);
            Console.WriteLine("Valor entero es: " + Geometria.valorEntero);
            Console.WriteLine("Valor real es: " + Geometria.valorReal);

            //Se instancia la clase
            Geometria objGeometria = new Geometria();
            double Esfera = objGeometria.VolumenEsfera(7);
            Console.WriteLine("Volumen Esfera: " + Esfera.ToString());
        }
    }
}
```

```
Se ha ejecutado el constructor static
Área círculo es: 153,93804002589985
Cadena es: Rafael
Valor entero es: 7
Valor real es: 16,832
Ejecuta el constructor de la clase
Volumen Esfera: 1077,5662801812991

C:\Users\engin\source\repos\Ejemplo\Ejemplo\bin\Release\net8
.0\Ejemplo.exe (proceso 14476) se cerró con el código 0.
Para cerrar automáticamente la consola cuando se detiene la
depuración, habilite Herramientas ->Opciones ->Depuración ->
```

Ilustración 130: Constructor static

Cuidado con el constructor static

El constructor static se ejecuta en el momento que es usada la clase por primera vez (por ejemplo, cuando se instancia), no se vuelve a usar más.

D/027.cs

```
namespace Ejemplo {

    //Clase con métodos estáticos
    internal class Geometria {
        public static int valorEntero;
        public static double valorReal;
        public static string unaCadena;

        //Constructor static (para inicializar atributos static)
        static Geometria() {
            valorEntero = 7;
            valorReal = 16.832;
            unaCadena = "Rafael";
            Console.WriteLine("Se ha ejecutado el constructor static");
        }

        //Constructor de clase
        public Geometria() {
            Console.WriteLine("Ejecuta el constructor de la clase");
        }

        //Este método estático puede ser usado sin instanciar la clase
        public static double AreaCirculo(double radio) {
            return Math.PI * radio * radio;
        }

        //Este método requiere instanciar la clase
        public double VolumenEsfera(double radio) {
            return 4 / 3 * Math.PI * Math.Pow(radio, 3);
        }
    }

    //Inicia la aplicación aquí
    internal class Program {
        static void Main() {
            //Se instancia la clase la primera vez
            Geometria objGeometria = new Geometria();
            double Esfera = objGeometria.VolumenEsfera(7);
            Console.WriteLine("Volumen Esfera A: " + Esfera.ToString());

            //Se instancia la clase la segunda vez
            Geometria objOtro = new Geometria();
            double OtraEsfera = objOtro.VolumenEsfera(7);
            Console.WriteLine("Volumen Esfera B: " + OtraEsfera.ToString());
        }
    }
}
```

The screenshot shows a command prompt window titled "Consola de depuración de Mi". The output text is as follows:

```
Se ha ejecutado el constructor static
Ejecuta el constructor de la clase
Volumen Esfera A: 1077,5662801812991
Ejecuta el constructor de la clase
Volumen Esfera B: 1077,5662801812991

C:\Users\engin\source\repos\Ejemplo\Ejemplo\bin\Release\net8
.0\Ejemplo.exe (proceso 5920) se cerró con el código 0.
Para cerrar automáticamente la consola cuando se detiene la
depuración, habilite Herramientas ->Opciones ->Depuración ->
Cerrar la consola automáticamente al detenerse la depuración
```

Ilustración 131: Cuidado con el constructor static

Interface

Con la palabra reservada "interface" se crean las definiciones de métodos y propiedades que deben ser escritos en las clases que implementen esa "interface". El estándar solicita que el nombre de las "interface" empiece por I

D/028.cs

```
namespace Ejemplo {
    //Declara una interface (el estándar dice que debe empezar con la letra "I")
    interface IMetodosRequeridos {
        //Métodos requeridos en las clases que implementen esta interface
        double AreaFigura();
        double PerimetroFigura();

    }

    //Esta clase debe implementar lo que dice la interface
    class Circulo : IMetodosRequeridos {
        public double Radio { get; set; }

        public Circulo(double Radio) {
            this.Radio = Radio;
        }

        //Implementa los métodos señalados por la interface
        public double AreaFigura() {
            return Math.PI * Radio * Radio;
        }

        public double PerimetroFigura() {
            return 2 * Math.PI * Radio;
        }
    }

    //Esta clase debe implementar lo que dice la interface
    class Cuadrado : IMetodosRequeridos {
        public double Lado { get; set; }

        public Cuadrado(double Lado) {
            this.Lado = Lado;
        }

        //Implementa los métodos señalados por la interface
        public double AreaFigura() {
            return Lado * Lado;
        }

        public double PerimetroFigura() {
            return 4 * Lado;
        }
    }

    //Inicia la aplicación aquí
    class Program {
        static void Main() {
            //Instancia las clases
            Cuadrado objCuadrado = new Cuadrado(5);
            Circulo objCirculo = new Circulo(5);

            //Imprime los valores
            Console.WriteLine("Área del círculo: " + objCirculo.AreaFigura().ToString());
            Console.WriteLine("Área del cuadrado: " + objCuadrado.AreaFigura().ToString());
            Console.WriteLine("Perímetro del círculo: " + objCirculo.PerimetroFigura().ToString());
            Console.WriteLine("Perímetro del cuadrado: " + objCuadrado.PerimetroFigura().ToString());
        }
    }
}
```

A screenshot of a Windows Command Prompt window titled "Consola de depuración de Mi...". The window contains the following text:

```
Área del círculo: 78,53981633974483
Área del cuadrado: 25
Perímetro del círculo: 31,41592653589793
Perímetro del cuadrado: 20

C:\Users\engin\source\repos\Ejemplo\Ejemplo\bin\Release\net8.0\Ejemplo.exe (proceso 10640) se cerró con
el código 0.

Para cerrar automáticamente la consola cuando se detiene la depuración, habilite Herramientas ->Opciones
```

Ilustración 132: Interface

```
namespace Ejemplo {
    //Declara una interface (el estándar dice que debe empezar con la letra "I")
    interface ICalculos {
        //Métodos requeridos en las clases que implementen esta interface
        double AreaFigura();
        double PerimetroFigura();

    }

    //Otra interface para obligar a mostrar los resultados
    interface IMuestra {
        void ImprimeArea();
        void ImprimePerimetro();
    }

    //Implementa de varios Interface
    class Circulo : ICalculos, IMuestra {
        public double Radio { get; set; }

        public Circulo(double Radio) {
            this.Radio = Radio;
        }

        //Implementa los métodos señalados por la interface
        public double AreaFigura() {
            return Math.PI * Radio * Radio;
        }

        public double PerimetroFigura() {
            return 2 * Math.PI * Radio;
        }

        public void ImprimeArea() {
            Console.WriteLine("Área del círculo: " + AreaFigura().ToString());
        }

        public void ImprimePerimetro() {
            Console.WriteLine("Perímetro del círculo: " + PerimetroFigura().ToString());
        }
    }

    class Cuadrado : ICalculos, IMuestra {
        public double Lado { get; set; }

        public Cuadrado(double Lado) {
            this.Lado = Lado;
        }

        //Implementa los métodos señalados por la interface
        public double AreaFigura() {
            return Lado * Lado;
        }

        public double PerimetroFigura() {
            return 4 * Lado;
        }

        public void ImprimeArea() {
            Console.WriteLine("Área del cuadrado: " + AreaFigura().ToString());
        }

        public void ImprimePerimetro() {
            Console.WriteLine("Perímetro del cuadrado: " + PerimetroFigura().ToString());
        }
    }

    //Inicia la aplicación aquí
    class Program {
        static void Main() {
            //Instancia las clases
            Cuadrado objCuadrado = new Cuadrado(5);
            Circulo objCirculo = new Circulo(5);

            //Imprime los valores
        }
    }
}
```

```
        objCuadrado.ImprimeArea();
        objCuadrado.ImprimePerimetro();

        objCirculo.ImprimeArea();
        objCirculo.ImprimePerimetro();
    }
}
```

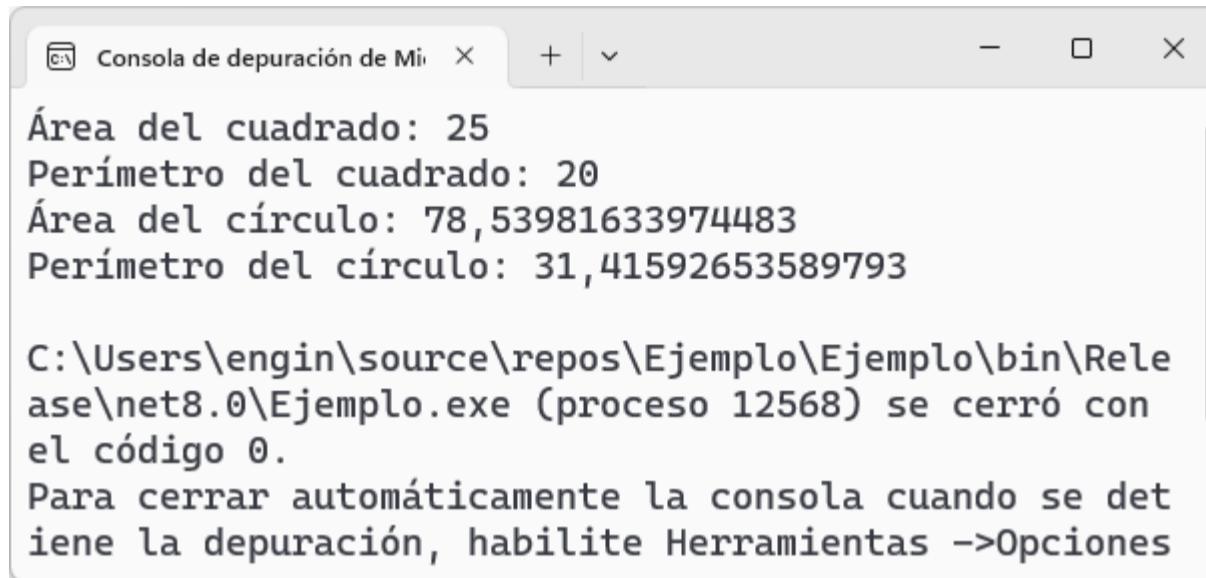


Ilustración 133: Interface múltiple

Se puede heredar de una clase e implementar de varios Interface

D/030.cs

```
namespace Ejemplo {

    interface IMetodos {
        void MetodoA();
        void MetodoB();
    }

    interface IProcedimientos {
        void ProcedimientoA();
        void ProcedimientoB();
    }

    class Madre {
        public void Aviso() {
            Console.WriteLine("Método de clase madre");
        }
    }

    //Hereda de Madre e implementa de IMetodos e IProcedimientos
    class Hija : Madre, IMetodos, IProcedimientos {
        public void Mensaje() {
            Console.WriteLine("En clase hija");
        }

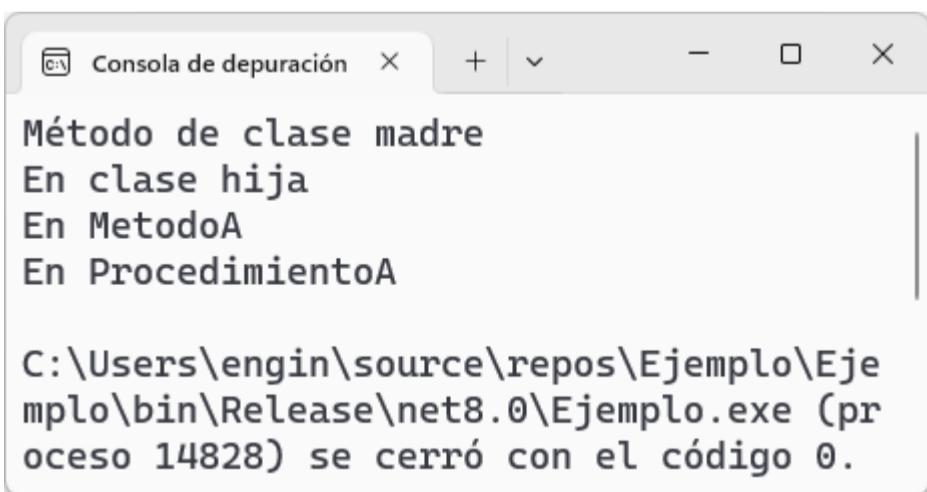
        public void MetodoA() {
            Console.WriteLine("En MetodoA");
        }

        public void MetodoB() {
            Console.WriteLine("En MetodoB");
        }

        public void ProcedimientoA() {
            Console.WriteLine("En ProcedimientoA");
        }

        public void ProcedimientoB() {
            Console.WriteLine("En ProcedimientoB");
        }
    }

    //Inicia la aplicación aquí
    class Program {
        static void Main() {
            Hija objHija = new Hija();
            objHija.Aviso();
            objHija.Mensaje();
            objHija.MetodoA();
            objHija.ProcedimientoA();
        }
    }
}
```



The screenshot shows a command prompt window titled "Consola de depuración". The output of the program is displayed:

```
Método de clase madre
En clase hija
En MetodoA
En ProcedimientoA
```

At the bottom of the window, the path and process information is shown:

```
C:\Users\engin\source\repos\Ejemplo\Eje
mple\bin\Release\net8.0\Ejemplo.exe (pr
oceso 14828) se cerró con el código 0.
```

Ilustración 134: Herencia e Interface

Es una “clase especial” para guardar constantes

D/031.cs

```
namespace Ejemplo {

    //Inicia la aplicación aquí
    internal class Program {

        //Una "clase especial" para almacenar constantes
        enum Meses {
            Enero, //0
            Febrero, //1
            Marzo, //2
            Abril, //3
            Mayo, //4
            Junio, //5
            Julio, //6
            Agosto, //7
            Septiembre, //8
            Octubre, //9
            Noviembre, //10
            Diciembre //11
        }

        static void Main() {
            Meses unMes = Meses.Junio;
            Console.WriteLine(unMes);
            Console.WriteLine((int) unMes);
        }
    }
}
```

Por defecto, las constantes de un enum comienzan a enumerarse desde cero, pero eso pueden cambiarse.

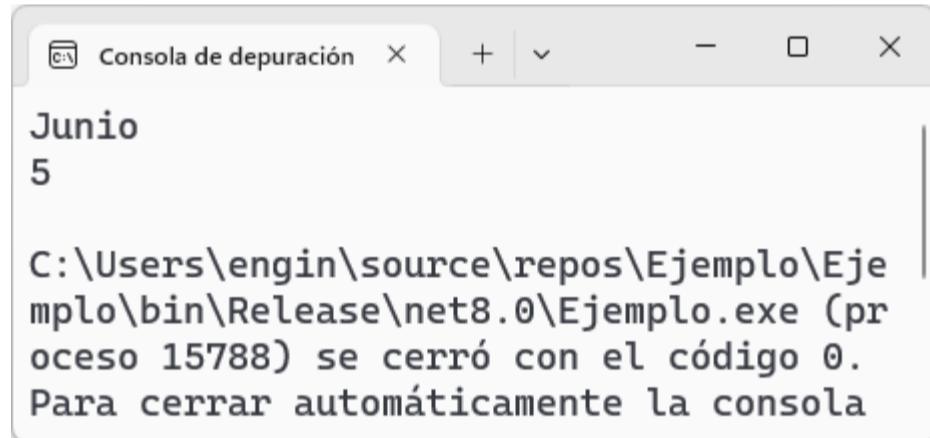


Ilustración 135: Enums

Cambiando los valores de las constantes en enums

Ejemplo 1

El método para que cada constante tenga su propio valor es constante = valor

D/032.cs

```
namespace Ejemplo {

    //Inicia la aplicación aquí
    internal class Program {

        //Una "clase especial" para almacenar constantes
        enum Meses {
            Enero = 1,
            Febrero = 2,
            Marzo = 3,
            Abril = 4,
            Mayo = 5,
            Junio = 6,
            Julio = 7,
            Agosto = 8,
            Septiembre = 9,
            Octubre = 10,
            Noviembre = 11,
            Diciembre = 12
        }

        static void Main() {
            Meses unMes = Meses.Junio;
            Console.WriteLine(unMes);
            Console.WriteLine((int) unMes);
        }
    }
}
```

```
Junio
6

C:\Users\engin\source\repos\Ejemplo\Ejemplo\bin\Release\net8.0\Ejemplo.exe (proceso 180) se cerró con el código 0.
Para cerrar automáticamente la consola
```

Ilustración 136: Cambiando los valores de las constantes en enums

```
namespace Ejemplo {

    //Inicia la aplicación aquí
    internal class Program {

        //Una "clase especial" para almacenar constantes
        enum Valores {
            valorA = 89,
            valorB = 12,
            valorC = 47,
            valorD = 63
        }

        static void Main() {
            Valores unosValores = Valores.valorC;
            Console.WriteLine(unosValores);
            Console.WriteLine((int) unosValores);
        }
    }
}
```

Nota 1: Los valores sólo pueden ser de tipo entero, long, byte, sbyte, short, ushort, uint, long, ulong

Nota 2: Los valores no pueden ser reales o cadenas o caracteres

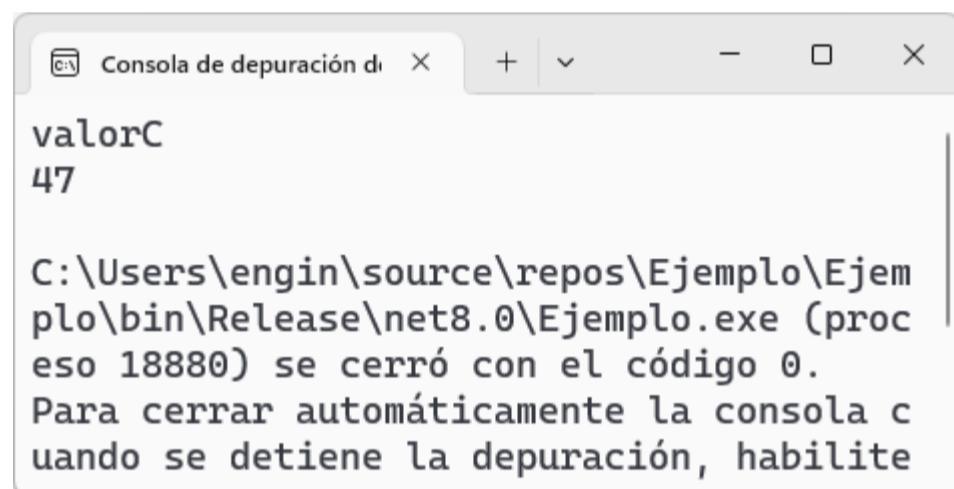


Ilustración 137: Cambiando los valores de las constantes en enums

Structs

C# trae las estructuras, que tienen un gran parecido a las clases, a tal punto que podrían ser su reemplazo en varias ocasiones porque tienen características interesantes como poder copiar el contenido de un struct en otro con el simple operador de asignación (algo que requiere un tratamiento si se usarán clases). Los structs tienen sus diferencias con respecto a las clases: sólo pueden definir constructores propios (con parámetros de entrada), pero no se puede crear un constructor simple, no pueden heredar, ni se puede heredar de estos, tampoco se pueden hacer structs abstractos.

D/034.cs

```
namespace Ejemplo {

    //Inicia la aplicación aquí
    internal class Program {

        //Una estructura
        struct Valores {
            public int valorA;
            public char valorB;
            public double valorC;
            public string valorD;
        }

        static void Main() {
            //Crea una variable de tipo struct
            Valores unosValores;
            unosValores.valorA = 13;
            unosValores.valorB = 'R';
            unosValores.valorC = 16.832;
            unosValores.valorD = "Milú";

            //Puede imprimir esos valores
            Console.WriteLine(unosValores.valorA);
            Console.WriteLine(unosValores.valorB);
            Console.WriteLine(unosValores.valorC);
            Console.WriteLine(unosValores.valorD);
        }
    }
}
```

```
13
R
16,832
Milú

C:\Users\engin\source\repos\Ejemplo\Ejemplo\b...
in\Release\net8.0\Ejemplo.exe (proceso 13004)
se cerró con el código 0.

Para cerrar automáticamente la consola cuando
se detiene la depuración, habilite Herramientas
```

Ilustración 138: Structs

Un struct se puede copiar fácilmente

Los valores de una variable struct se pueden copiar en otra variable struct de la misma estructura usando el operador de asignación (=). Si se modifica el original, no afecta a la copia.

D/035.cs

```
namespace Ejemplo {

    //Inicia la aplicación aquí
    internal class Program {

        //Una estructura
        struct Valores {
            public int valorA;
            public char valorB;
            public double valorC;
            public string valorD;
        }

        static void Main() {
            //Crea una variable de tipo struct
            Valores unosValores;
            unosValores.valorA = 13;
            unosValores.valorB = 'R';
            unosValores.valorC = 16.832;
            unosValores.valorD = "Milú";

            //Crea una segunda variable y le asigna la primera, creando una copia
            Valores otro;
            otro = unosValores;

            //Puede imprimir esos valores
            Console.WriteLine("Valores copiados");
            Console.WriteLine(otro.valorA);
            Console.WriteLine(otro.valorB);
            Console.WriteLine(otro.valorC);
            Console.WriteLine(otro.valorD);

            //Modifica la original
            unosValores.valorA = -9876;

            //Imprime la copia
            Console.WriteLine("\nValores después de modificar el original");
            Console.WriteLine(otro.valorA);
        }
    }
}
```

```
Consola de depuración de Mi X + - □ ×
Valores copiados
13
R
16,832
Milú

Valores después de modificar el original
13

C:\Users\engin\source\repos\Ejemplo\Ejemplo\bin\Release
\net8.0\Ejemplo.exe (proceso 7692) se cerró con el código 0.
```

Ilustración 139: Un struct se puede copiar fácilmente

Métodos en un struct

Un struct puede tener métodos.

D/036.cs

```
namespace Ejemplo {

    //Inicia la aplicación aquí
    internal class Program {

        //Una estructura
        struct Valores {
            private int valorA;
            private char valorB;
            private double valorC;
            private string valorD;

            public void DaValores(int valorA, char valorB, double valorC, string valorD) {
                this.valorA = valorA;
                this.valorB = valorB;
                this.valorC = valorC;
                this.valorD = valorD;
            }

            public void ImprimeValores() {
                Console.WriteLine(valorA);
                Console.WriteLine(valorB);
                Console.WriteLine(valorC);
                Console.WriteLine(valorD);
            }

            public double RetornaValor(double numero) {
                return valorA * valorC + numero;
            }
        }

        static void Main() {
            //Crea una variable de tipo struct y la inicializa por defecto
            Valores unosValores = default;

            //Llama a los métodos del struct
            unosValores.DaValores(13, 'R', 16.832, "Milú");
            unosValores.ImprimeValores();

            //Y a la función del struct
            Console.WriteLine(unosValores.RetornaValor(5));
        }
    }
}
```

```
13
R
16,832
Milú
223,816

C:\Users\engin\source\repos\Ejemplo\Ejemplo\bin\Release
\net8.0\Ejemplo.exe (proceso 16764) se cerró con el código 0.
```

Ilustración 140: Métodos en un struct

Structs y constructores

Un struct puede tener un constructor que tenga parámetros (no se puede generar un constructor sin parámetros). Ejemplo:

D/037.cs

```
namespace Ejemplo {

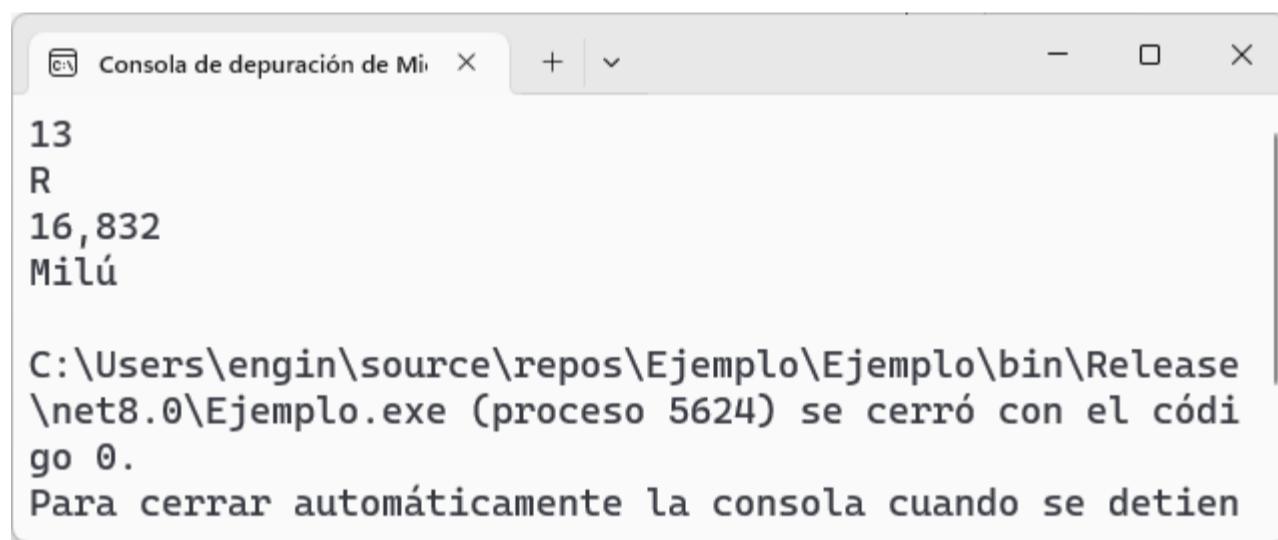
    //Inicia la aplicación aquí
    internal class Program {

        //Una estructura
        struct Valores {
            private int valorA;
            private char valorB;
            private double valorC;
            private string valorD;

            public Valores(int valorA, char valorB, double valorC, string valorD) {
                this.valorA = valorA;
                this.valorB = valorB;
                this.valorC = valorC;
                this.valorD = valorD;
            }

            public void ImprimeValores() {
                Console.WriteLine(valorA);
                Console.WriteLine(valorB);
                Console.WriteLine(valorC);
                Console.WriteLine(valorD);
            }
        }

        static void Main() {
            //Crea una variable de tipo struct y la inicializa con un constructor
            Valores unosValores = new Valores(13, 'R', 16.832, "Milú");
            unosValores.ImprimeValores();
        }
    }
}
```



13
R
16,832
Milú

C:\Users\engin\source\repos\Ejemplo\Ejemplo\bin\Release
\net8.0\Ejemplo.exe (proceso 5624) se cerró con el código 0.
Para cerrar automáticamente la consola cuando se detiene

Ilustración 141: Structs y constructores

Clases parciales

C# puede tener partes de una misma clase en diferentes archivos. Útil si dos o más programadores quieren trabajar en la misma clase al tiempo. En Visual Studio, se hace para separar la parte lógica de un formulario, de la parte de diseño GUI de ese mismo formulario.

D/038a.cs

```
namespace Ejemplo {
    partial class MiClase {
        public MiClase(int valorA, double valorB, char valorC, string valorD) {
            ValorA = valorA;
            ValorB = valorB;
            ValorC = valorC;
            ValorD = valorD;
        }

        public void Imprime() {
            Console.WriteLine("Valores");
            Console.WriteLine(ValorA);
            Console.WriteLine(ValorB);
            Console.WriteLine(ValorC);
            Console.WriteLine(ValorD);
        }
    }

    //Inicia la aplicación aquí
    class Program {
        public static void Main() {
            MiClase objClase = new MiClase(2010, 7.15, 'S', "Sally");
            objClase.Imprime();
        }
    }
}
```

D/038b.cs

```
namespace Ejemplo {
    partial class MiClase {
        public int ValorA { get; set; }
        public double ValorB { get; set; }
        public char ValorC { get; set; }
        public string ValorD { get; set; }
    }
}
```

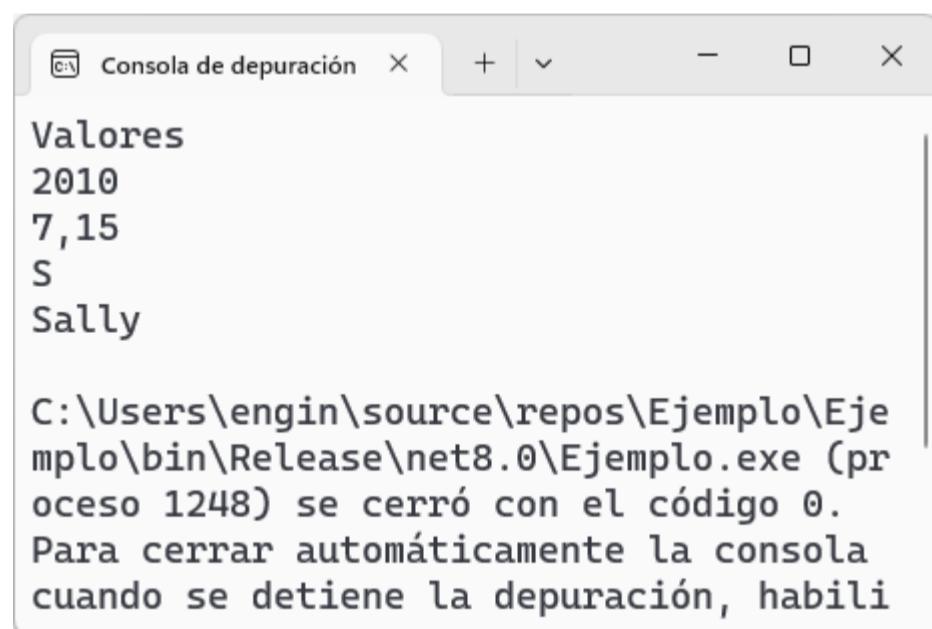


Ilustración 142: Clases parciales

Destructores

Por lo general, C# se encarga automáticamente de liberar la memoria de los objetos que ya no pueden ser usados (por ejemplo, cuando se crea un objeto dentro de una función con una variable local y luego termina la función). Aun así, en raras ocasiones, es necesario tener un método que se ejecuta cuando el objeto es eliminado, sería la contraparte del constructor y es conocido como el destructor. A pesar de su existencia, los destructores no pueden llamarse explícitamente. Eso sucede cuando el "Garbage Collector" lo considere oportuno. Entonces una forma de ejecutar el destructor es llamando explícitamente al Garbage Collector (con las siglas GC) para que haga limpieza y liberación de memoria.

Los destructores se nombran iniciando con el símbolo ~ seguido del nombre de la clase. No tienen parámetros.

D/039a.cs

```
namespace Ejemplo {

    //Inicia la aplicación aquí
    internal class Program {
        public static void Main() {
            Procedimiento();

            //Ejecuta el Garbage Collector
            GC.Collect(); //Limpia todo
            GC.WaitForPendingFinalizers(); //Espera que se limpie todo

            Console.WriteLine("Termina el programa");
        }

        public static void Procedimiento() {
            //Se instancia la clase con una variable local
            MiClase objClase = new MiClase(2010, 7.15, 'S', "Sally");
            objClase.Imprime();

            //Aquí debería ejecutarse el destructor de esa clase
        }
    }
}
```

D/039b.cs

```
namespace Ejemplo {
    partial class MiClase {
        public int ValorA { get; set; }
        public double ValorB { get; set; }
        public char ValorC { get; set; }
        public string ValorD { get; set; }

        public MiClase(int valorA, double valorB, char valorC, string valorD) {
            ValorA = valorA;
            ValorB = valorB;
            ValorC = valorC;
            ValorD = valorD;
        }

        public void Imprime() {
            Console.WriteLine("Valores");
            Console.WriteLine(ValorA);
            Console.WriteLine(ValorB);
            Console.WriteLine(ValorC);
            Console.WriteLine(ValorD);
        }

        //Destructor
        ~MiClase() {
            Console.WriteLine("Ejecuta el destructor");
        }
    }
}
```

Nota: Por lo publicado en diversos foros sobre los destructores y el uso del Garbage Collector, esto debe hacerlo con mucho cuidado, así que se recomienda no hacer uso de destructores, ni llamar al Garbage Collector.

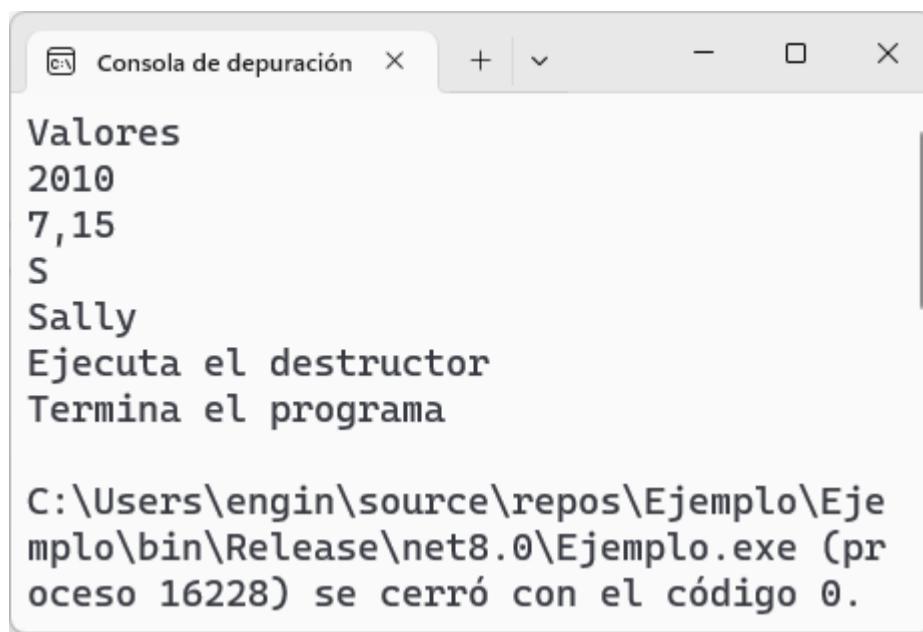


Ilustración 143: Destructores

Patrones de diseño

A continuación, algunos patrones de diseño implementados en C#:

Factory Method

El patrón de diseño Factory Method es un patrón creacional que se utiliza para crear objetos sin tener que especificar su clase exacta.

Se necesita crear objetos de una clase, pero no se sabe qué clase exacta necesita hasta que se ejecuta el programa. El patrón Factory Method permite crear objetos sin tener que especificar su clase exacta. En lugar de crear objetos directamente, se utiliza un método de fábrica para crear objetos. Este método de fábrica se encarga de crear el objeto correcto según los parámetros que se le pasen.

D/040.cs

```
//Patrón: Factory Method
namespace Ejemplo {
    //Interface que obliga a definir el método dibujar
    interface IFigura {
        void Dibujar();
    }

    class Circulo : IFigura {
        public void Dibujar() {
            Console.WriteLine("Se hace el dibujo de un círculo");
        }
    }

    class Rectangulo : IFigura {
        public void Dibujar() {
            Console.WriteLine("Estoy dibujando un rectángulo");
        }
    }

    class Triangulo : IFigura {
        public void Dibujar() {
            Console.WriteLine("Ahora se dibuja un triángulo");
        }
    }

    class FabricaFiguras {
        //Dependiendo del parámetro retorna uno u otro objeto
        public IFigura GetFigura(string TipoFigura) {
            if (TipoFigura.Equals("CIRCULO")) return new Circulo();
            if (TipoFigura.Equals("RECTANGULO")) return new Rectangulo();
            if (TipoFigura.Equals("TRIANGULO")) return new Triangulo();
            return null;
        }
    }

    class Program {
        static void Main() {
            FabricaFiguras objeto = new FabricaFiguras();

            //Obtiene un objeto círculo
            IFigura Figura1 = objeto.GetFigura("CIRCULO");

            //Llama el método de dibujar del objeto círculo
            Figura1.Dibujar();

            //Obtiene un objeto rectángulo
            IFigura Figura2 = objeto.GetFigura("RECTANGULO");

            //Llama el método de dibujar del objeto rectángulo
            Figura2.Dibujar();

            //Obtiene un objeto triángulo
            IFigura Figura3 = objeto.GetFigura("TRIANGULO");

            //Llama el método de dibujar del objeto triángulo
            Figura3.Dibujar();
        }
    }
}
```

```
Consola de depuración  X  +  -  □  X
Se hace el dibujo de un círculo
Estoy dibujando un rectángulo
Ahora se dibuja un triángulo

C:\Users\engin\source\repos\Ejemplo\Eje
mple\bin\Release\net8.0\Ejemplo.exe (pr
oceso 11932) se cerró con el código 0.
Para cerrar automáticamente la consola
cuando se detiene la depuración, habili
te Herramientas ->Opciones ->Depuración
```

Ilustración 144: Factory Method

Abstract Factory

El patrón de diseño Abstract Factory es un patrón creacional que se utiliza para crear familias de objetos relacionados o dependientes sin especificar su clase concreta.

Es necesario crear un conjunto de objetos que trabajen juntos, pero no se sabe qué objetos específicos se necesitan hasta que se ejecuta el programa: El patrón Abstract Factory permite crear una fábrica abstracta que define una interfaz para crear objetos relacionados o dependientes. Luego, puede crear fábricas concretas que implementan la fábrica abstracta y crean objetos específicos. De esta manera, puede crear diferentes familias de objetos relacionados o dependientes sin tener que cambiar el código del cliente.

D/041.cs

```
//Patrón: Abstract Factory
namespace Ejemplo {
    //Interface que obliga a definir el método dibujar
    public interface IFigura {
        void Dibujar();
    }

    class Rectangulo : IFigura {
        public void Dibujar() {
            Console.WriteLine("Estoy dibujando un rectángulo");
        }
    }

    class Triangulo : IFigura {
        public void Dibujar() {
            Console.WriteLine("Ahora se dibuja un triángulo");
        }
    }

    class Circulo : IFigura {
        public void Dibujar() {
            Console.WriteLine("Se hace el dibujo de un círculo");
        }
    }

    public interface IColor {
        void Rellenar();
    }

    class Rojo : IColor {
        public void Rellenar() {
            Console.WriteLine("Pinta de rojo");
        }
    }

    class Verde : IColor {
        public void Rellenar() {
            Console.WriteLine("Un verde es pintado");
        }
    }

    class Azul : IColor {
        public void Rellenar() {
            Console.WriteLine("Ahora de azul es relleno");
        }
    }

    public abstract class FabricaAbstracta {
        public abstract IFigura GetFigura(string TipoFigura);
        public abstract IColor GetColor(string color);
    }

    public class FabricaFiguras : FabricaAbstracta {
        //Dependiendo del parámetro retorna uno u otro objeto
        public override IFigura GetFigura(string TipoFigura) {
            if (TipoFigura.Equals("CIRCULO")) return new Circulo();
            if (TipoFigura.Equals("RECTANGULO")) return new Rectangulo();
            if (TipoFigura.Equals("TRIANGULO")) return new Triangulo();
            return null;
        }

        public override IColor GetColor(string color) {
            return null;
        }
    }

    class FabricaColores : FabricaAbstracta {
```

```

//Dependiendo del parámetro retorna uno u otro objeto
public override IFigura GetFigura(string TipoFigura) {
    return null;
}

public override IColor GetColor(string color) {
    if (color.Equals("ROJO")) return new Rojo();
    if (color.Equals("VERDE")) return new Verde();
    if (color.Equals("AZUL")) return new Azul();
    return null;
}
}

class ProductorDeFabricas {
    public static FabricaAbstracta GetFabrica(string seleccion) {
        if (seleccion.Equals("FIGURA")) return new FabricaFiguras();
        if (seleccion.Equals("COLOR")) return new FabricaColores();
        return null;
    }
}

class Program {
    static void Main() {
        //Trae una determinada fábrica en este caso de FIGURA
        FabricaAbstracta fabricaFiguras = ProductorDeFabricas.GetFabrica("FIGURA");

        //Obtenida la fábrica, se solicita un tipo de objeto de esa fábrica
        IFigura figural = fabricaFiguras.GetFigura("CIRCULO");

        //Llama un método de ese objeto dado por la fábrica en particular
        figural.Dibujar();

        //Obtenida la fábrica, se solicita un tipo de objeto de esa fábrica
        IFigura figura2 = fabricaFiguras.GetFigura("RECTANGULO");

        //Llama un método de ese objeto dado por la fábrica en particular
        figura2.Dibujar();

        //Obtenida la fábrica, se solicita un tipo de objeto de esa fábrica
        IFigura figura3 = fabricaFiguras.GetFigura("TRIANGULO");

        //Llama un método de ese objeto dado por la fábrica en particular
        figura3.Dibujar();

        //Trae una determinada fábrica en este caso de COLOR
        FabricaAbstracta FabricaColores = ProductorDeFabricas.GetFabrica("COLOR");

        //Obtenida la fábrica, se solicita un tipo de objeto de esa fábrica
        IColor color1 = FabricaColores.GetColor("ROJO");

        //Llama un método de ese objeto dado por la fábrica en particular
        color1.Rellenar();

        //Obtenida la fábrica, se solicita un tipo de objeto de esa fábrica
        IColor color2 = FabricaColores.GetColor("VERDE");

        //Llama un método de ese objeto dado por la fábrica en particular
        color2.Rellenar();

        //Obtenida la fábrica, se solicita un tipo de objeto de esa fábrica
        IColor color3 = FabricaColores.GetColor("AZUL");

        //Llama un método de ese objeto dado por la fábrica en particular
        color3.Rellenar();
    }
}
}

```

```
Se hace el dibujo de un círculo
Estoy dibujando un rectángulo
Ahora se dibuja un triángulo
Pinta de rojo
Un verde es pintado
Ahora de azul es rellenado

C:\Users\engin\source\repos\Ejemplo\Eje
mple\bin\Release\net8.0\Ejemplo.exe (pr
```

Ilustración 145: Abstract Factory

Singleton

El patrón de diseño Singleton es un patrón creacional que se utiliza para garantizar que una clase tenga exactamente una instancia y proporcionar un punto de acceso global a esta. En otras palabras, el patrón Singleton se utiliza para restringir la creación de objetos pertenecientes a una clase o el valor de un tipo a un único objeto.

D/042.cs

```
//Patrón: Singleton
namespace PatronDiseno {
    class ObjetoUnico {
        //Genera un objeto de ObjetoUnico
        private static ObjetoUnico instancia = new ObjetoUnico();

        //Hace el constructor privado por lo que no puede ser instanciado
        private ObjetoUnico() { }

        //Retorna la única instancia de esta clase
        public static ObjetoUnico GetInstancia() {
            return instancia;
        }

        public void Mensaje() {
            Console.WriteLine("Esta es una prueba");
        }
    }

    class Program {
        static void Main() {
            //Quite el comentario de esta instrucción y generará un error al compilar
            //ObjetoUnico pruebaObjeto = new ObjetoUnico();

            //Obtiene el único objeto instanciable
            ObjetoUnico miObjeto = ObjetoUnico.GetInstancia();

            //Muestra un mensaje
            miObjeto.Mensaje();
        }
    }
}
```

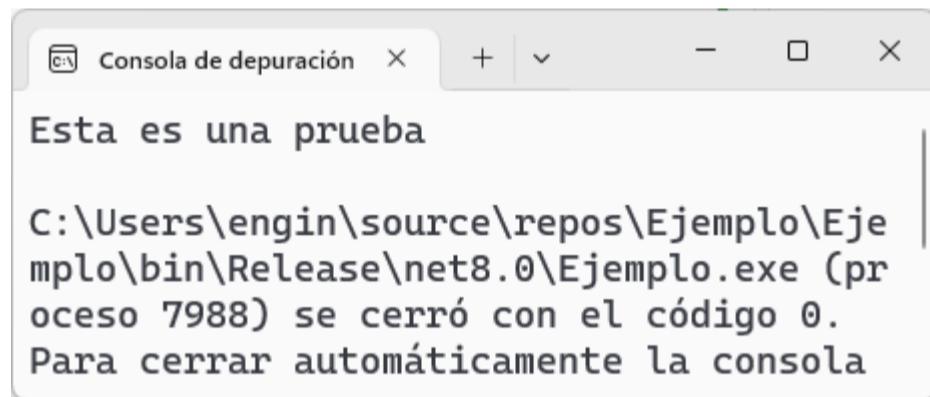


Ilustración 146: Singleton

Builder

El patrón de diseño Builder es un patrón creacional que se utiliza para crear objetos complejos paso a paso. Este patrón permite la creación de diferentes tipos y representaciones de un objeto utilizando el mismo proceso de construcción.

Cuando es necesario crear un objeto complejo que requiere una inicialización laboriosa, paso a paso, de muchos campos y objetos anidados: Si se intenta crear el objeto pasando argumentos a un constructor, puede que se termine con un constructor con muchos parámetros, muchos de los cuales no se usarían en la mayoría de los casos. El patrón Builder permite abstraer el proceso de creación de un objeto complejo, centralizando dicho proceso en un único punto, de tal forma que el mismo proceso de construcción pueda crear representaciones diferentes.

D/043.cs

```
//Patrón: Builder
namespace Ejemplo {
    public interface IEmpacado {
        string Empaque(); //Obligar a hacer el método Empaque()
    }

    class Envoltura : IEmpacado {
        public string Empaque() {
            return "Empaque Ecológico";
        }
    }

    public class Botella : IEmpacado {
        public string Empaque() {
            return "Botella biodegradable";
        }
    }

    //Todo producto en la comida tendrá estos ítems: Nombre, como se empaca, precio
    public interface Item {
        string Nombre();
        IEmpacado EmpacandoProducto();
        float Precio();
    }

    public abstract class Hamburguesa : Item {
        public IEmpacado EmpacandoProducto() {
            return new Envoltura();
        }
        public abstract float Precio();
        public abstract string Nombre();
    }

    public class HamburguesaPollo : Hamburguesa {
        public override float Precio() {
            return 7000;
        }

        public override string Nombre() {
            return "Hamburguesa de pollo";
        }
    }

    class HamburguesaVegetariana : Hamburguesa {
        public override float Precio() {
            return 5000;
        }

        public override string Nombre() {
            return "Hamburguesa vegetariana";
        }
    }

    public abstract class BebidaFria : Item {
        public IEmpacado EmpacandoProducto() {
            return new Botella();
        }
        public abstract float Precio();
        public abstract string Nombre();
    }

    class Malteada : BebidaFria {
        public override float Precio() {
            return 4700;
        }

        public override string Nombre() {
    }
```

```

        return "Malteada";
    }

}

class CaFeFrio : BebidaFria {
    public override float Precio() {
        return 4000;
    }

    public override string Nombre() {
        return "Café frio";
    }
}

class Comida {
    private List<Item> items = new List<Item>();

    public void AddItem(Item item) {
        items.Add(item);
    }

    public float GetCosto() {
        float costo = 0.0f;
        foreach (Item item in items) {
            costo += item.Precio();
        }
        return costo;
    }

    public void MostrarItems() {
        foreach (Item item in items) {
            Console.WriteLine("Item: " + item.Nombre());
            Console.WriteLine("Empaque: " + item.EmpacandoProducto().Empaque());
            Console.WriteLine("Precio: " + item.Precio());
        }
    }
}

//Prepara la comida dependiendo si es vegetariana o no
class FabricaComida {
    public Comida PrepararComidaVegetariana() {
        Comida miComida = new Comida();
        miComida.AddItem(new HamburguesaVegetariana());
        miComida.AddItem(new CaFeFrio());
        return miComida;
    }

    public Comida PrepararComidaNoVegetariana() {
        Comida miComida = new Comida();
        miComida.AddItem(new HamburguesaPollo());
        miComida.AddItem(new Malteada());
        return miComida;
    }
}

class Program {
    static void Main() {
        FabricaComida miComida = new FabricaComida();

        Comida vegetariano = miComida.PrepararComidaVegetariana();
        Console.WriteLine("Comida vegetariana");
        vegetariano.MostrarItems();
        Console.WriteLine("Costo total: " + vegetariano.GetCosto());

        Comida noVegetariano = miComida.PrepararComidaNoVegetariana();
        Console.WriteLine("\n\nComida No vegetariana");
        noVegetariano.MostrarItems();
        Console.WriteLine("Costo total: " + noVegetariano.GetCosto());
    }
}
}

```

```
Consola de depuración de I X + - □ X
Comida vegetariana
Item: Hamburguesa vegetariana
, Empaque: Empaque Ecológico
, Precio: 5000
Item: Café frío
, Empaque: Botella biodegradable
, Precio: 4000
Costo total: 9000

Comida No vegetariana
Item: Hamburguesa de pollo
, Empaque: Empaque Ecológico
, Precio: 7000
Item: Malteada
, Empaque: Botella biodegradable
, Precio: 4700
Costo total: 11700

C:\Users\engin\source\repos\Ejemplo\Ejemplo\bin\Release\net8.0\Ejemplo.exe (proces
```

Ilustración 147: Builder

Adapter

El patrón de diseño Adapter es un patrón estructural que se utiliza para adaptar una interfaz existente a otra interfaz. En otras palabras, el adaptador actúa como un intermediario entre dos interfaces incompatibles y proporciona una capa adicional de abstracción para permitir que los objetos trabajen juntos.

Se tienen dos clases con interfaces incompatibles, y es necesario que trabajen juntas. El patrón Adapter permite crear una clase intermedia que actúa como un traductor entre las dos interfaces. El adaptador implementa la interfaz del cliente y utiliza la interfaz del servicio para realizar la traducción.

D/044.cs

```
//Patrón de diseño: Adapter
namespace Ejemplo {
    public interface IEjecutorMultimedia {
        void Ejecutar(string TipoAudio, string NombreArchivo);
    }

    public interface IEjecutorAvanzadoArchivosMultimedia {
        void EjecutaVLC(string NombreArchivo);
        void EjecutaMP4(string NombreArchivo);
    }

    class EjecutorVLC : IEjecutorAvanzadoArchivosMultimedia {
        public void EjecutaVLC(string NombreArchivo) {
            Console.WriteLine("Ejecutando un archivo VLC. Nombre: " + NombreArchivo);
        }
        public void EjecutaMP4(string NombreArchivo) {
        }
    }

    class EjecutorMP4 : IEjecutorAvanzadoArchivosMultimedia {
        public void EjecutaVLC(string NombreArchivo) {
        }
        public void EjecutaMP4(string NombreArchivo) {
            Console.WriteLine("Ejecutando un archivo MP4. Nombre: " + NombreArchivo);
        }
    }

    class AdaptadorMultimedia : IEjecutorMultimedia {
        IEjecutorAvanzadoArchivosMultimedia ejecutorAvanzado;

        //Constructor
        public AdaptadorMultimedia(string TipoAudio) {
            if (TipoAudio.Equals("vlc")) {
                ejecutorAvanzado = new EjecutorVLC();
            }
            if (TipoAudio.Equals("mp4")) {
                ejecutorAvanzado = new EjecutorMP4();
            }
        }

        //Dependiendo del tipo de audio llama a VLC o MP4
        public void Ejecutar(string TipoAudio, string NombreArchivo) {
            if (TipoAudio.Equals("vlc")) {
                ejecutorAvanzado.EjecutaVLC(NombreArchivo);
            }
            else if (TipoAudio.Equals("mp4")) {
                ejecutorAvanzado.EjecutaMP4(NombreArchivo);
            }
        }
    }

    class EjecutorAudio : IEjecutorMultimedia {
        AdaptadorMultimedia adaptadorMultimedia;

        public void Ejecutar(string TipoAudio, string NombreArchivo) {
            //Archivos MP3
            if (TipoAudio.Equals("mp3")) {
                Console.WriteLine("Ejecutando archivo MP3. Nombre: " + NombreArchivo);
            } //Otros formatos
            else if (TipoAudio.Equals("vlc") || TipoAudio.Equals("mp4")) {
                adaptadorMultimedia = new AdaptadorMultimedia(TipoAudio);
                adaptadorMultimedia.Ejecutar(TipoAudio, NombreArchivo);
            }
            else {
                Console.WriteLine("Medio inválido. (" + TipoAudio + ") es un formato no soportado");
            }
        }
    }
}
```

```
}

class Program {
    static void Main() {
        EjecutorAudio Multimedia = new EjecutorAudio();

        Multimedia.Ejecutar("mp3", "MiMusica.mp3");
        Multimedia.Ejecutar("mp4", "unSonido.mp4");
        Multimedia.Ejecutar("vlc", "FondoMusical.vlc");
        Multimedia.Ejecutar("avi", "unAudio.avi");
    }
}
```

The screenshot shows a Windows Command Prompt window with the title bar 'Consola de depuración de Mi...'. The window contains the following text:

```
Ejecutando archivo MP3. Nombre: MiMusica.mp3
Ejecutando un archivo MP4. Nombre: unSonido.mp4
Ejecutando un archivo VLC. Nombre: FondoMusical.vlc
Medio inválido. (avi) es un formato no soportado

C:\Users\engin\source\repos\Ejemplo\Ejemplo\bin\Release\
net8.0\Ejemplo.exe (proceso 10816) se cerró con el código
```

Ilustración 148: Adapter

Composite

El patrón de diseño Composite es un patrón estructural que se utiliza para representar jerarquías parte-todo como un árbol. Este patrón permite a los clientes tratar objetos individuales y composiciones de objetos de manera uniforme.

Se tiene una estructura jerárquica de objetos, donde cada objeto puede ser un objeto simple o un objeto compuesto: el patrón Composite permite tratar tanto los objetos simples como los objetos compuestos de la misma manera, como si fueran una instancia única de un objeto.

D/045.cs

```
//Patrón de diseño: Composite
namespace Ejemplo {
    public class Empleado {
        private string nombre;
        private string departamento;
        private int salario;
        private List<Empleado> subordinados;

        //Constructor
        public Empleado(string nombre, string departamento, int salario) {
            this.nombre = nombre;
            this.departamento = departamento;
            this.salario = salario;
            subordinados = new List<Empleado>();
        }

        public void Adicionar(Empleado objEmpleado) {
            subordinados.Add(objEmpleado);
        }

        public void Quitar(Empleado objEmpleado) {
            subordinados.Remove(objEmpleado);
        }

        public List<Empleado> GetSubordinados() {
            return subordinados;
        }

        public new string ToString() {
            return "Empleado => Nombre: " + nombre + ", departamento: " + departamento + ", salario: " +
salario.ToString();
        }
    }

    internal class Program {
        static void Main() {
            Empleado Gerente = new Empleado("Laura", "Gerente", 5000000);
            Empleado jefeVentas = new Empleado("Patricia", "Jefa de Ventas", 3000000);
            Empleado jefeMercadeo = new Empleado("Adriana", "Jefa de Mercadeo", 3000000);
            Empleado disenador1 = new Empleado("Sandra", "Marketing", 2000000);
            Empleado disenador2 = new Empleado("Alejandra", "Marketing", 2000000);
            Empleado vendedor1 = new Empleado("Francisca", "Ventas", 200000);
            Empleado vendedor2 = new Empleado("Flor", "Ventas", 200000);

            Gerente.Adicionar(jefeVentas);
            Gerente.Adicionar(jefeMercadeo);

            jefeVentas.Adicionar(vendedor1);
            jefeVentas.Adicionar(vendedor2);

            jefeMercadeo.Adicionar(disenador1);
            jefeMercadeo.Adicionar(disenador2);

            //Imprime todos los empleados de la organización
            Console.WriteLine(Gerente.ToString());

            foreach (Empleado jefe in Gerente.GetSubordinados()) {
                Console.WriteLine(jefe.ToString());

                foreach (Empleado empleado in jefe.GetSubordinados()) {
                    Console.WriteLine(empleado.ToString());
                }
            }
        }
    }
}
```

```
Empleado => Nombre: Laura, departamento: Gerente, salario: 5000000
Empleado => Nombre: Patricia, departamento: Jefa de Ventas, salario: 3000000
Empleado => Nombre: Francisca, departamento: Ventas, salario: 200000
Empleado => Nombre: Flor, departamento: Ventas, salario: 2000000
Empleado => Nombre: Adriana, departamento: Jefa de Mercadeo, salario: 3000000
Empleado => Nombre: Sandra, departamento: Marketing, salario: 2000000
Empleado => Nombre: Alejandra, departamento: Marketing, salario: 2000000

C:\Users\engin\source\repos\Ejemplo\Ejemplo\bin\Release\net8.0\Ejemplo.exe (proceso 14276) se cerró con el código 0.
Para cerrar automáticamente la consola cuando se detiene la depuración, habilite
```

Ilustración 149: Composite

El patrón de diseño Facade es un patrón estructural que se utiliza para simplificar la complejidad de un sistema. Imaginarse tener un sistema complejo con muchos subsistemas, cada uno con su propia interfaz. Si un cliente quiere interactuar con el sistema, tendría que conocer todas las interfaces de los subsistemas, lo que puede ser muy complicado. El patrón Facade proporciona una interfaz unificada para un conjunto de interfaces en un subsistema. La idea detrás de este patrón es proporcionar una interfaz simple para un subsistema complejo, reduciendo así la complejidad del sistema y minimizando las comunicaciones y dependencias entre los subsistemas.

D/046.cs

```
//Patrón de diseño: Facade
namespace Ejemplo {
    interface IFigura {
        void Dibujar();
    }

    class Circulo : IFigura {
        public void Dibujar() {
            Console.WriteLine("Dibujando un círculo");
        }
    }

    class Rectangulo : IFigura {
        public void Dibujar() {
            Console.WriteLine("Traza un rectángulo");
        }
    }

    class Triangulo : IFigura {
        public void Dibujar() {
            Console.WriteLine("Delinea un triángulo");
        }
    }

    class HacerFigura {
        private IFigura circulo;
        private IFigura rectangulo;
        private IFigura triangulo;

        public HacerFigura() {
            circulo = new Circulo();
            rectangulo = new Rectangulo();
            triangulo = new Triangulo();
        }

        public void DibujaCirculo() {
            circulo.Dibujar();
        }

        public void DibujaRectangulo() {
            rectangulo.Dibujar();
        }

        public void DibujaTriangulo() {
            triangulo.Dibujar();
        }
    }

    class Program {
        static void Main() {
            HacerFigura hacefigura = new HacerFigura();

            hacefigura.DibujaCirculo();
            hacefigura.DibujaRectangulo();
            hacefigura.DibujaTriangulo();
        }
    }
}
```

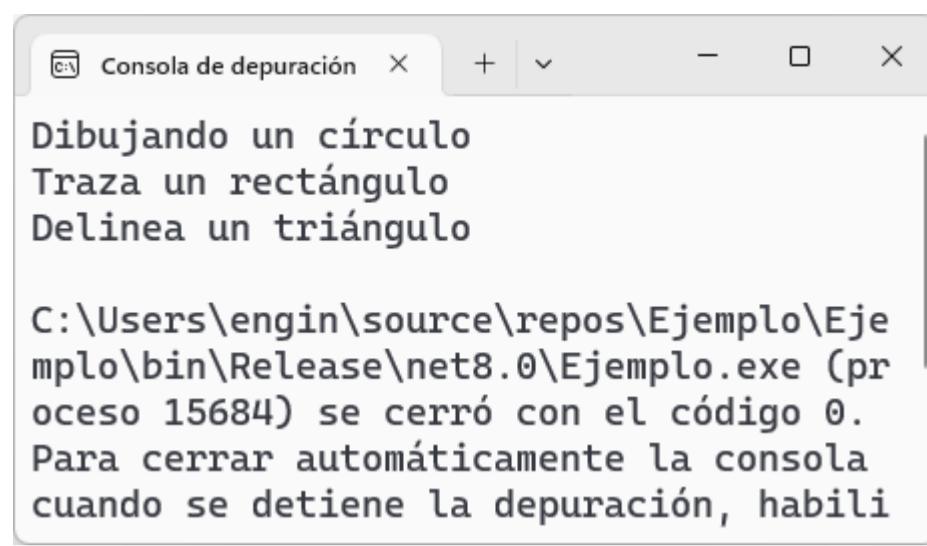


Ilustración 150: Facade

Modelo Vista Controlador

El patrón de diseño modelo vista controlador (MVC) es una forma de organizar una aplicación en tres componentes principales: el modelo, la vista y el controlador. Cada componente tiene una función específica y se comunica con los otros para lograr una interacción fluida entre el usuario y la aplicación. Una breve explicación de cada componente:

Modelo: Es el componente que gestiona los datos y la lógica de la aplicación. Se encarga de almacenar, manipular y validar los datos, así como de interactuar con fuentes externas como bases de datos o APIs. El modelo no tiene conocimiento de la interfaz de usuario, solo se ocupa de los datos.

Vista: Es el componente que muestra los datos al usuario. Se encarga de generar la interfaz gráfica de usuario (GUI). La vista recibe los datos del modelo a través del controlador y los presenta de forma atractiva y comprensible. La vista también puede capturar las acciones del usuario, como hacer clic en un botón o introducir un texto, y enviarlas al controlador.

Controlador: Es el componente que coordina la comunicación entre el modelo y la vista. Se encarga de procesar las peticiones del usuario, como solicitar una página o enviar un formulario, y de invocar al modelo para obtener o modificar los datos necesarios. El controlador también decide qué vista mostrar al usuario según el resultado del modelo.

D/047.cs

```
//Patrón de diseño: Modelo Vista Controlador
namespace Ejemplo {
    class Estudiante {
        public string Codigo { get; set; }
        public string Nombre { get; set; }
    }

    class ControladorEstudiante {
        private Estudiante modelo;
        private VisorEstudiante vista;

        public ControladorEstudiante(Estudiante modelo, VisorEstudiante vista) {
            this.modelo = modelo;
            this.vista = vista;
        }

        public void setNombreEstudiante(string nombre) {
            modelo.Nombre = nombre;
        }

        public string getNombreEstudiante() {
            return modelo.Nombre;
        }

        public void setCodigoEstudiante(string codigo) {
            modelo.Codigo = codigo;
        }

        public string getCodigoEstudiante() {
            return modelo.Codigo;
        }

        public void ActualizarVista() {
            vista.ImprimeDetallesEstudiante(modelo.Nombre, modelo.Codigo);
        }
    }

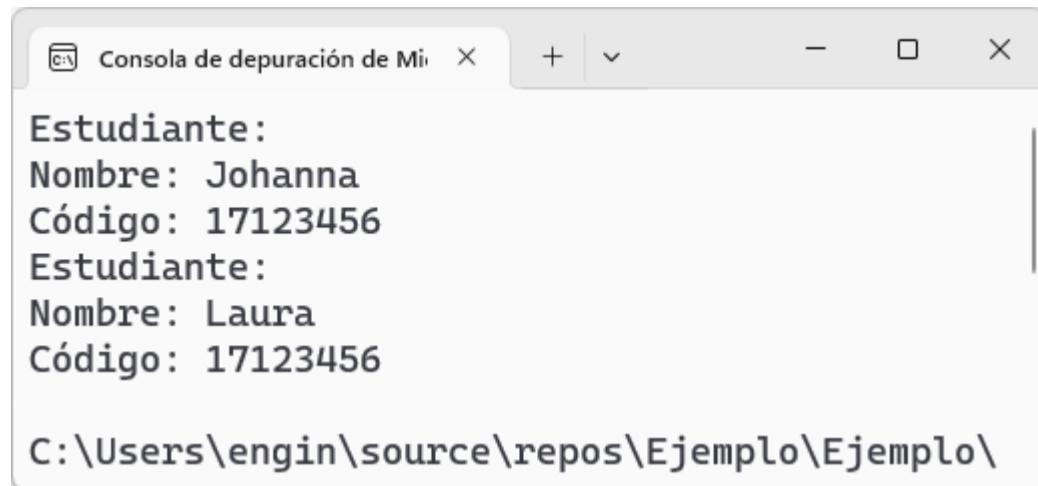
    class VisorEstudiante {
        public void ImprimeDetallesEstudiante(string NombreEstudiante, string CodigoEstudiante) {
            Console.WriteLine("Estudiante: ");
            Console.WriteLine("Nombre: " + NombreEstudiante);
            Console.WriteLine("Código: " + CodigoEstudiante);
        }
    }

    class Program {
        static void Main() {
            Estudiante modelo = TraeEstudianteBaseDatos();
            VisorEstudiante vista = new VisorEstudiante();
            ControladorEstudiante controlador = new ControladorEstudiante(modelo, vista);

            controlador.ActualizarVista();
            controlador.setNombreEstudiante("Laura");
            controlador.ActualizarVista();
        }

        private static Estudiante TraeEstudianteBaseDatos() {
            Estudiante estudiante = new Estudiante();
            estudiante.Nombre = "Johanna";
            estudiante.Codigo = "17123456";
        }
    }
}
```

```
        return estudiante;
    }
}
```



```
Consola de depuración de Mi... + - X
Estudiante:
Nombre: Johanna
Código: 17123456
Estudiante:
Nombre: Laura
Código: 17123456
```

C:\Users\engin\source\repos\Ejemplo\Ejemplo\

Ilustración 151: Modelo Vista Controlador

Parte 5: Estructuras de datos dinámicas

El ArrayList

Adicionar, tamaño, buscar e imprimir

En C# está el ArrayList, que es un contenedor de objetos de cualquier tipo.

E/001.cs

```
using System.Collections;

namespace Ejemplo {
    class Program {
        static void Main() {
            //Declara la lista que almacenará cadenas
            ArrayList ListaAnimales = new ArrayList();

            //Adiciona elementos a la lista
            ListaAnimales.Add("Ballena");
            ListaAnimales.Add("Tortuga marina");
            ListaAnimales.Add("Tiburón");
            ListaAnimales.Add("Estrella de mar");
            ListaAnimales.Add("Hipocampo");
            ListaAnimales.Add("Serpiente marina");
            ListaAnimales.Add("Delfín");
            ListaAnimales.Add("Pulpo");
            ListaAnimales.Add("Caballito de mar");
            ListaAnimales.Add("Coral");
            ListaAnimales.Add("Pingüinos");
            ListaAnimales.Add("Calamar");
            ListaAnimales.Add("Gaviota");
            ListaAnimales.Add("Foca");
            ListaAnimales.Add("Manatíes");
            ListaAnimales.Add("Ballena con barba");
            ListaAnimales.Add("Peces Guppy");
            ListaAnimales.Add("Orca");
            ListaAnimales.Add("Medusas");
            ListaAnimales.Add("Mejillones");
            ListaAnimales.Add("Caracoles");

            //Tamaño la lista
            int tamano = ListaAnimales.Count;
            Console.WriteLine("Tamaño de la lista: " + tamano);

            //Traer un determinado elemento de la lista
            int posicion = 7;
            string texto = ListaAnimales[posicion].ToString();
            Console.WriteLine("Elemento en la posición " + posicion + " es: " + texto);

            //Nos dice si existe un determinado elemento en la lista
            string buscar = "Foca";
            bool Existe = ListaAnimales.Contains(buscar);
            Console.WriteLine("Busca: " + buscar + " Resultado: " + Existe);

            //Nos dice la posición donde encontró el elemento en la lista
            int posBusca = ListaAnimales.IndexOf(buscar);
            Console.WriteLine("Busca: " + buscar + " Posición: " + posBusca.ToString());

            //Imprime la lista
            foreach (object elemento in ListaAnimales)
                Console.Write("{0}{1}", ";", elemento);
        }
    }
}
```

The screenshot shows the 'Consola de depuración de Mi...' (Debug Console) window from Visual Studio. It displays the output of the program's execution:

```
Tamaño de la lista: 21
Elemento en la posición 7 es: Pulpo
Busca: Foca Resultado: True
Busca: Foca Posición: 13
;Ballena;Tortuga marina;Tiburón;Estrella de mar;Hipocampo;Serpiente m
arina;Delfín;Pulpo;Caballito de mar;Coral;Pingüinos;Calamar;Gaviota;F
oca;Manatíes;Ballena con barba;Peces Guppy;Orca;Medusas;Mejillones;Ca
racoles
```

Ilustración 152: ArrayList Adicionar, tamaño, buscar e imprimir

Los ArrayLists empiezan en cero. En el código anterior se muestran las funciones para adicionar al ArrayList, determinar el tamaño (número de elementos que tiene), decir si existe un determinado elemento y mostrar la lista usando foreach.

```

using System.Collections;

namespace Ejemplo {
    class Program {
        static void Main() {
            //Declara la lista
            ArrayList ListaAnimales = new ArrayList();

            //Adiciona elementos a la lista
            ListaAnimales.Add("Ballena");
            ListaAnimales.Add("Tortuga marina");
            ListaAnimales.Add("Tiburón");
            ListaAnimales.Add("Hipocampo");
            ListaAnimales.Add("Delfín");
            ListaAnimales.Add("Pulpo");
            ListaAnimales.Add("Caballito de mar");
            ListaAnimales.Add("Coral");
            ListaAnimales.Add("Pingüinos");

            //Imprime la lista
            foreach (object elemento in ListaAnimales) Console.Write("{0}{1}", ";", elemento);
            Console.WriteLine(" ");

            //Retira elemento de la lista
            ListaAnimales.Remove("Hipocampo");

            //Imprime de nuevo la lista
            foreach (object elemento in ListaAnimales) Console.Write("{0}{1}", ";", elemento);
            Console.WriteLine(" ");

            //Elimina el objeto de determinada posición.
            ListaAnimales.RemoveAt(5);

            //Imprime de nuevo la lista
            foreach (object elemento in ListaAnimales) Console.Write("{0}{1}", ";", elemento);
        }
    }
}

```

```

;Ballena;Tortuga marina;Tiburón;Hipocampo;Delfín;Pulpo;Caballito de mar;Coral;Pingüinos
;Ballena;Tortuga marina;Tiburón;Delfín;Pulpo;Caballito de mar;Coral;Pingüinos
;Ballena;Tortuga marina;Tiburón;Delfín;Pulpo;Coral;Pingüinos

```

Ilustración 153: ArrayList, borrar elemento

Dos técnicas para eliminar elementos de un ArrayList, buscando el elemento y eliminándolo, o dada una posición se elimina lo que hay allí.

Cambiar Elemento

Con la posición de la cadena en la lista, se puede cambiar directamente.

E/003.cs

```
using System.Collections;

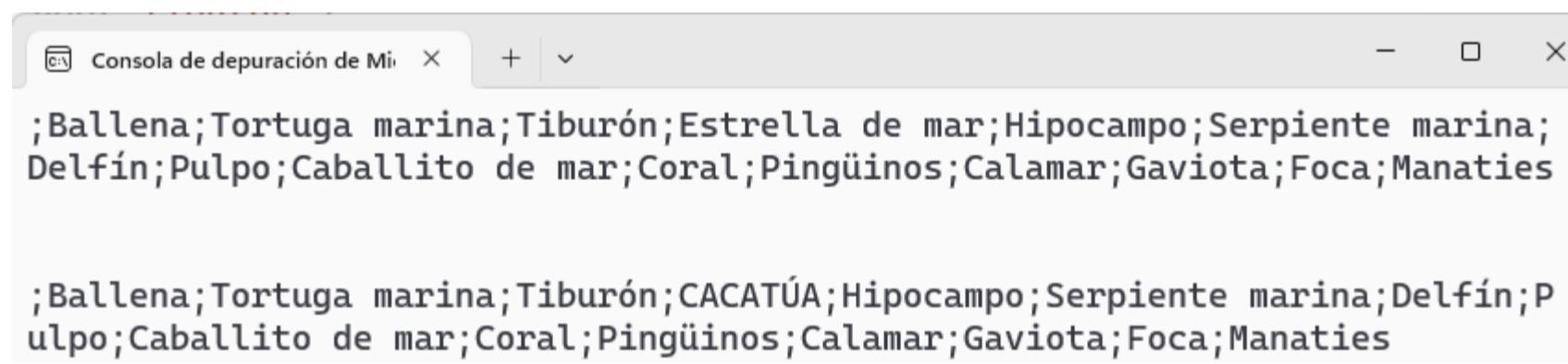
namespace Ejemplo {
    class Program {
        public static void Main() {
            //Declara la lista que almacenará cadenas
            ArrayList Ejemplo = new ArrayList();

            //Adiciona elementos a la lista
            Ejemplo.Add("Ballena");
            Ejemplo.Add("Tortuga marina");
            Ejemplo.Add("Tiburón");
            Ejemplo.Add("Estrella de mar");
            Ejemplo.Add("Hipocampo");
            Ejemplo.Add("Serpiente marina");
            Ejemplo.Add("Delfín");
            Ejemplo.Add("Pulpo");
            Ejemplo.Add("Caballito de mar");
            Ejemplo.Add("Coral");
            Ejemplo.Add("Pingüinos");
            Ejemplo.Add("Calamar");
            Ejemplo.Add("Gaviota");
            Ejemplo.Add("Foca");
            Ejemplo.Add("Manatíes");

            //Imprime valores
            foreach (Object objeto in Ejemplo) Console.Write("{0}{1}", ";", objeto);
            Console.WriteLine("\r\n");

            //Cambia una cadena en la lista
            Ejemplo[3] = "CACATÚA";

            //Imprime valores
            foreach (Object objeto in Ejemplo) Console.Write("{0}{1}", ";", objeto);
        }
    }
}
```



```
;Ballena;Tortuga marina;Tiburón;Estrella de mar;Hipocampo;Serpiente marina;
Delfín;Pulpo;Caballito de mar;Coral;Pingüinos;Calamar;Gaviota;Foca;Manatíes

;Ballena;Tortuga marina;Tiburón;CACATÚA;Hipocampo;Serpiente marina;Delfín;P
ulpo;Caballito de mar;Coral;Pingüinos;Calamar;Gaviota;Foca;Manatíes
```

Ilustración 154: ArrayList, cambiar elemento

Insertar Elemento

Se puede insertar un elemento en la lista simplemente señalando su posición.

E/004.cs

```
using System.Collections;

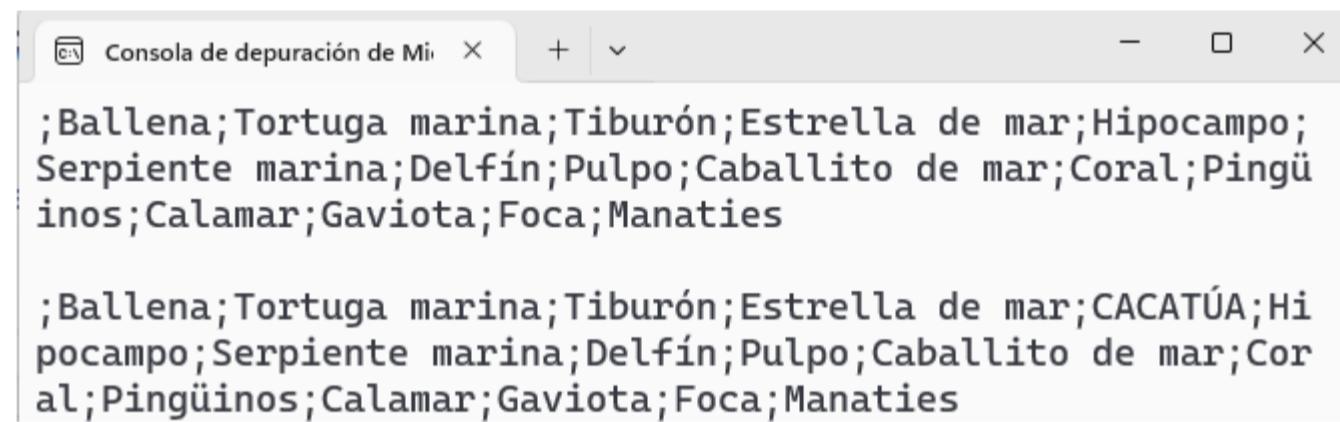
namespace Ejemplo {
    class Program {
        public static void Main() {
            //Declara la lista que almacenará cadenas
            ArrayList Ejemplo = new ArrayList();

            //Adiciona elementos a la lista
            Ejemplo.Add("Ballena");
            Ejemplo.Add("Tortuga marina");
            Ejemplo.Add("Tiburón");
            Ejemplo.Add("Estrella de mar");
            Ejemplo.Add("Hipocampo");
            Ejemplo.Add("Serpiente marina");
            Ejemplo.Add("Delfín");
            Ejemplo.Add("Pulpo");
            Ejemplo.Add("Caballito de mar");
            Ejemplo.Add("Coral");
            Ejemplo.Add("Pingüinos");
            Ejemplo.Add("Calamar");
            Ejemplo.Add("Gaviota");
            Ejemplo.Add("Foca");
            Ejemplo.Add("Manatíes");

            //Imprime valores
            foreach (Object objeto in Ejemplo) Console.Write("{0}{1}", ";", objeto);
            Console.WriteLine("\r\n");

            //Inserta una cadena en la posición 4 de la lista
            Ejemplo.Insert(4, "CACATÚA");

            //Imprime valores
            foreach (Object objeto in Ejemplo) Console.Write("{0}{1}", ";", objeto);
        }
    }
}
```



```
;Ballena;Tortuga marina;Tiburón;Estrella de mar;Hipocampo;
Serpiente marina;Delfín;Pulpo;Caballito de mar;Coral;Pingü
inos;Calamar;Gaviota;Foca;Manatíes

;Ballena;Tortuga marina;Tiburón;Estrella de mar;CACATÚA;Hi
pocampo;Serpiente marina;Delfín;Pulpo;Caballito de mar;Cor
al;Pingüinos;Calamar;Gaviota;Foca;Manatíes
```

Ilustración 155: ArrayList, insertar elemento

Referenciar con una variable, un rango de la lista

Dada una lista A, se crea una variable B de tipo ArrayList que haga referencia a un rango de esa lista A. Como es una referencia, si se modifica un elemento de B, ese cambio ocurrirá en A.

E/005.cs

```
using System.Collections;

namespace Ejemplo {
    class Program {
        public static void Main() {
            //Declara la lista que almacenará cadenas
            ArrayList Ejemplo = new ArrayList();

            //Adiciona elementos a la lista
            Ejemplo.Add("Ballena");
            Ejemplo.Add("Tortuga marina");
            Ejemplo.Add("Tiburón");
            Ejemplo.Add("Estrella de mar");
            Ejemplo.Add("Hipocampo");
            Ejemplo.Add("Serpiente marina");
            Ejemplo.Add("Delfín");
            Ejemplo.Add("Pulpo");
            Ejemplo.Add("Caballito de mar");
            Ejemplo.Add("Coral");
            Ejemplo.Add("Pingüinos");

            //Imprime valores
            Console.WriteLine("Lista original");
            foreach (Object objeto in Ejemplo) Console.Write("{0}{1}", ";", objeto);
            Console.WriteLine("\r\n");

            //Genera nueva lista
            int posicionInicial = 5;
            int cantidad = 3;
            ArrayList nuevaLista = Ejemplo.GetRange(posicionInicial, cantidad);

            //Imprime valores de esa nueva lista
            Console.WriteLine("Nueva lista");
            foreach (Object objeto in nuevaLista) Console.Write("{0}{1}", ";", objeto);
            Console.WriteLine("\r\n");

            //Modifica un valor de la nueva lista
            nuevaLista[0] = "CACATÚA";

            //Imprime la lista nueva con el valor alterado
            Console.WriteLine("Nueva lista con primer valor alterado");
            foreach (Object objeto in nuevaLista) Console.Write("{0}{1}", ";", objeto);
            Console.WriteLine("\r\n");

            //Imprime de nuevo la lista original
            Console.WriteLine("Lista original");
            foreach (Object objeto in Ejemplo) Console.Write("{0}{1}", ";", objeto);
        }
    }
}
```

```
Consola de depuración de Mi... + ▾ - □ ×
Lista original
;Ballena;Tortuga marina;Tiburón;Estrella de mar;Hipocampo;Serpiente marina;Delfín;Pulpo;Caballito de mar;Coral;Pingüinos

Nueva lista
;Serpiente marina;Delfín;Pulpo

Nueva lista con primer valor alterado
;CACATÚA;Delfín;Pulpo

Lista original
;Ballena;Tortuga marina;Tiburón;Estrella de mar;Hipocampo;CACATÚA;Delfín;Pulpo;Caballito de mar;Coral;Pingüinos
```

Ilustración 156: ArrayList, referenciar con una variable, un rango de la lista

```

using System.Collections;

namespace Ejemplo {
    class Program {
        public static void Main() {
            //Declara la lista que almacenará cadenas
            ArrayList Ejemplo = new ArrayList();

            //Adiciona elementos a la lista
            Ejemplo.Add("Ballena");
            Ejemplo.Add("Tortuga marina");
            Ejemplo.Add("Tiburón");
            Ejemplo.Add("Estrella de mar");
            Ejemplo.Add("Hipocampo");
            Ejemplo.Add("Serpiente marina");
            Ejemplo.Add("Delfín");
            Ejemplo.Add("Pulpo");
            Ejemplo.Add("Caballito de mar");
            Ejemplo.Add("Coral");
            Ejemplo.Add("Pingüinos");

            //Recorrido con foreach
            Console.WriteLine("Recorrido con foreach");
            foreach (Object objeto in Ejemplo) Console.Write("{0}{1}", ";", objeto);
            Console.WriteLine("\r\n");

            //Recorrido con for
            Console.WriteLine("Recorrido con for");
            for (int cont = 0; cont < Ejemplo.Count; cont++) Console.Write("{0}{1}", ";",
                Ejemplo[cont]);
            Console.WriteLine("\r\n");

            //Recorrido con un IEnumarator
            Console.WriteLine("Recorrido con un IEnumarator");
            IEnumarator elemento = Ejemplo.GetEnumerator();
            while (elemento.MoveNext()) Console.Write("{0}{1}", ";", elemento.Current);
        }
    }
}

```

```

Consola de depuración de Mi... + - X
Recorrido con foreach
;Ballena;Tortuga marina;Tiburón;Estrella de mar;Hipocampo;Serpiente marina;Delfín;Pulpo;Caballito de mar;Coral;Pingüinos

Recorrido con for
;Ballena;Tortuga marina;Tiburón;Estrella de mar;Hipocampo;Serpiente marina;Delfín;Pulpo;Caballito de mar;Coral;Pingüinos

Recorrido con un IEnumarator
;Ballena;Tortuga marina;Tiburón;Estrella de mar;Hipocampo;Serpiente marina;Delfín;Pulpo;Caballito de mar;Coral;Pingüinos

```

Ilustración 157: ArrayList, tres formas de recorrerlo

Borrar completamente un ArrayList

Se utiliza el método Clear()

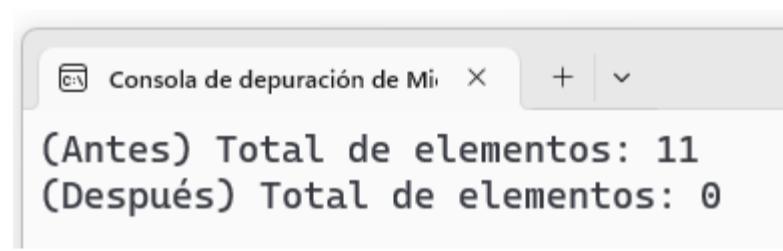
E/007.cs

```
using System.Collections;

namespace Ejemplo {
    class Program {
        public static void Main() {
            //Declara la lista que almacenará cadenas
            ArrayList Ejemplo = new ArrayList();

            //Adiciona elementos a la lista
            Ejemplo.Add("Ballena");
            Ejemplo.Add("Tortuga marina");
            Ejemplo.Add("Tiburón");
            Ejemplo.Add("Estrella de mar");
            Ejemplo.Add("Hipocampo");
            Ejemplo.Add("Serpiente marina");
            Ejemplo.Add("Delfín");
            Ejemplo.Add("Pulpo");
            Ejemplo.Add("Caballito de mar");
            Ejemplo.Add("Coral");
            Ejemplo.Add("Pingüinos");

            //Limpia el ArrayList
            Console.WriteLine("(Antes) Total de elementos: " + Ejemplo.Count);
            Ejemplo.Clear();
            Console.WriteLine("(Después) Total de elementos: " + Ejemplo.Count);
        }
    }
}
```



```
Consola de depuración de Mi... X + ▾
(Antes) Total de elementos: 11
(Después) Total de elementos: 0
```

Ilustración 158: Borrar completamente un ArrayList

Borrar un rango en un ArrayList

Se utiliza el método RemoveRange()

E/008.cs

```
using System.Collections;

namespace Ejemplo {
    class Program {
        public static void Main() {
            //Declara la lista que almacenará cadenas
            ArrayList Ejemplo = new ArrayList();

            //Adiciona elementos a la lista
            Ejemplo.Add("Ballena");
            Ejemplo.Add("Tortuga marina");
            Ejemplo.Add("Tiburón");
            Ejemplo.Add("Estrella de mar");
            Ejemplo.Add("Hipocampo");
            Ejemplo.Add("Serpiente marina");
            Ejemplo.Add("Delfín");
            Ejemplo.Add("Pulpo");
            Ejemplo.Add("Caballito de mar");
            Ejemplo.Add("Coral");
            Ejemplo.Add("Pingüinos");

            //Elimina un rango de elementos del ArrayList
            Console.WriteLine("Antes");
            foreach (Object objeto in Ejemplo) Console.Write("{0}{1}", ";", objeto);
            Console.WriteLine("\r\n");

            int posicion = 1;
            int cantidad = 4;
            Ejemplo.RemoveRange(posicion, cantidad);

            Console.WriteLine("Después");
            foreach (Object objeto in Ejemplo) Console.Write("{0}{1}", ";", objeto);
        }
    }
}
```

The screenshot shows the 'Consola de depuración de Mi...' (Debug Console) window. It displays two lines of output: 'Antes' (Before) and 'Después' (After). The 'Antes' line shows all ten items from the ArrayList separated by semicolons. The 'Después' line shows the same items, but the elements at indices 1 through 4 have been removed, resulting in only six items.

```
Antes
;Ballena;Tortuga marina;Tiburón;Estrella de mar;Hipocampo;Serpiente marina;Delfín;Pulpo;Caballito de mar;Coral;Pingüinos

Después
;Ballena;Serpiente marina;Delfín;Pulpo;Caballito de mar;Coral;Pingüinos
```

Ilustración 159: Borrar un rango en un ArrayList

Guardar el ArrayList en un arreglo estático

Todos los elementos del ArrayList pasan a un arreglo estático, sea de tipo object() o de algún tipo de dato como string.

E/009.cs

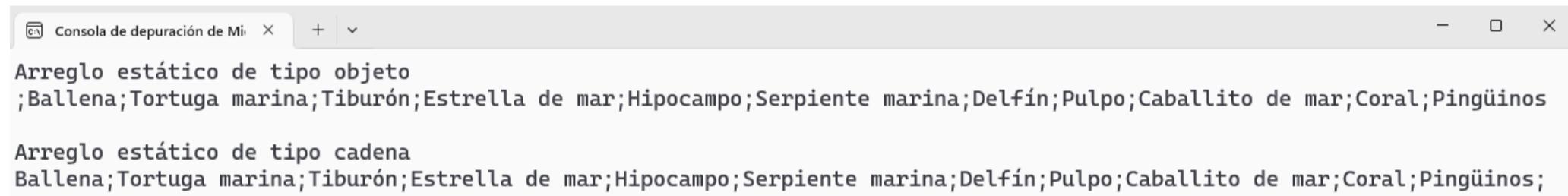
```
using System.Collections;

namespace Ejemplo {
    class Program {
        public static void Main() {
            //Declara la lista que almacenará cadenas
            ArrayList Ejemplo = new ArrayList();

            //Adiciona elementos a la lista
            Ejemplo.Add("Ballena");
            Ejemplo.Add("Tortuga marina");
            Ejemplo.Add("Tiburón");
            Ejemplo.Add("Estrella de mar");
            Ejemplo.Add("Hipocampo");
            Ejemplo.Add("Serpiente marina");
            Ejemplo.Add("Delfín");
            Ejemplo.Add("Pulpo");
            Ejemplo.Add("Caballito de mar");
            Ejemplo.Add("Coral");
            Ejemplo.Add("Pingüinos");

            //Guarda el ArrayList en un arreglo estático de tipo objeto
            Console.WriteLine("Arreglo estático de tipo objeto");
            object[] arregloEstatico = Ejemplo.ToArray();
            foreach (Object objeto in arregloEstatico) Console.Write("{0}{1}", ";", objeto);
            Console.WriteLine("\r\n");

            //Guarda el ArrayList en un arreglo estático de tipo string
            Console.WriteLine("Arreglo estático de tipo cadena");
            string[] cadenas = Ejemplo.ToArray(typeof(string)) as string[];
            for (int cont = 0; cont < cadenas.Length; cont++)
                Console.Write(cadenas[cont] + ";");
        }
    }
}
```



```
Consola de depuración de Mi... X + ▾ - □ ×
Arreglo estático de tipo objeto
;Ballena;Tortuga marina;Tiburón;Estrella de mar;Hipocampo;Serpiente marina;Delfín;Pulpo;Caballito de mar;Coral;Pingüinos

Arreglo estático de tipo cadena
Ballena;Tortuga marina;Tiburón;Estrella de mar;Hipocampo;Serpiente marina;Delfín;Pulpo;Caballito de mar;Coral;Pingüinos;
```

Ilustración 160: Guardar el ArrayList en un arreglo estático

Agregar un arreglo estático a un ArrayList

Toma el contenido del arreglo estático y lo copia en el ArrayList usando el método AddRange()

E/010.cs

```
using System.Collections;

namespace Ejemplo {
    class Program {
        public static void Main() {
            //Declara la lista que almacenará cadenas
            ArrayList Ejemplo = new ArrayList();

            //Adiciona elementos a la lista
            Ejemplo.Add("Ballena");
            Ejemplo.Add("Tortuga marina");
            Ejemplo.Add("Tiburón");
            Ejemplo.Add("Estrella de mar");
            Ejemplo.Add("Hipocampo");
            Ejemplo.Add("Serpiente marina");
            Ejemplo.Add("Delfín");
            Ejemplo.Add("Pulpo");
            Ejemplo.Add("Caballito de mar");
            Ejemplo.Add("Coral");
            Ejemplo.Add("Pingüinos");

            //Un arreglo estático
            string[] Cadenas = { "Gato", "Perro", "Conejo", "Liebre", "Oveja" };

            //Adiciona el arreglo estático al ArrayList
            Ejemplo.AddRange(Cadenas);

            //Imprime el ArrayList
            for (int cont = 0; cont < Ejemplo.Count; cont++) Console.Write(Ejemplo[cont] + "; ");
        }
    }
}
```



Ilustración 161: Agregar un arreglo estático a un ArrayList

Inserta un arreglo estático en una determinada posición del ArrayList

Toma el contenido del arreglo estático, lo copia y lo inserta en alguna posición del ArrayList usando el método InsertRange()

E/011.cs

```
using System.Collections;

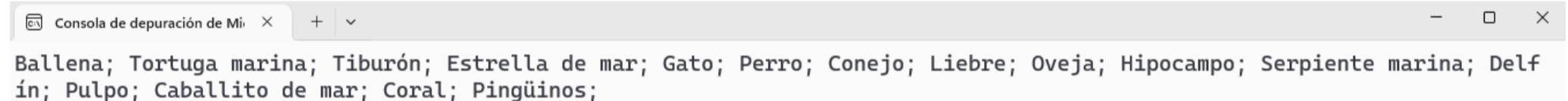
namespace Ejemplo {
    class Program {
        public static void Main() {
            //Declara la lista que almacenará cadenas
            ArrayList Ejemplo = new ArrayList();

            //Adiciona elementos a la lista
            Ejemplo.Add("Ballena");
            Ejemplo.Add("Tortuga marina");
            Ejemplo.Add("Tiburón");
            Ejemplo.Add("Estrella de mar");
            Ejemplo.Add("Hipocampo");
            Ejemplo.Add("Serpiente marina");
            Ejemplo.Add("Delfín");
            Ejemplo.Add("Pulpo");
            Ejemplo.Add("Caballito de mar");
            Ejemplo.Add("Coral");
            Ejemplo.Add("Pingüinos");

            //Un arreglo estático
            string[] Cadenas = { "Gato", "Perro", "Conejo", "Liebre", "Oveja" };

            //Inserta el arreglo estático en una determinada posición del ArrayList
            int posicionInserta = 4;
            Ejemplo.InsertRange(posicionInserta, Cadenas);

            //Imprime el ArrayList
            for (int cont = 0; cont < Ejemplo.Count; cont++) Console.Write(Ejemplo[cont] + "; ");
        }
    }
}
```



```
Ballena; Tortuga marina; Tiburón; Estrella de mar; Gato; Perro; Conejo; Liebre; Oveja; Hipocampo; Serpiente marina; Delfín; Pulpo; Caballito de mar; Coral; Pingüinos;
```

Ilustración 162: Inserta un arreglo estático en una determinada posición del ArrayList

Insertar varios ArrayList dentro de un ArrayList

El contenido de varios ArrayList es copiado dentro de otro ArrayList. Como es una copia, no importa si se modifica el ArrayList fuente.

E/012.cs

```
using System.Collections;

namespace Ejemplo {
    class Program {
        public static void Main() {
            //Declara tres ArrayList
            ArrayList ListaA = new ArrayList();
            ArrayList ListaB = new ArrayList();
            ArrayList ListaC = new ArrayList();

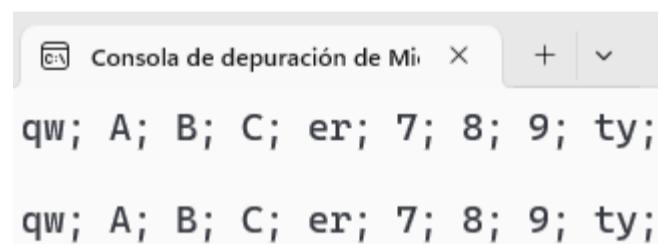
            //Adiciona elementos a esos ArrayList
            ListaA.Add("A");
            ListaA.Add("B");
            ListaA.Add("C");
            ListaB.Add("7");
            ListaB.Add("8");
            ListaB.Add("9");
            ListaC.Add("qw");
            ListaC.Add("er");
            ListaC.Add("ty");

            //Inserta los dos primeros ArrayList en el tercero
            int posicionInserta = 1;
            ListaC.InsertRange(posicionInserta, ListaA);

            posicionInserta = 5;
            ListaC.InsertRange(posicionInserta, ListaB);

            //Imprime el ArrayList
            for (int cont = 0; cont < ListaC.Count; cont++) Console.Write(ListaC[cont] + "; ");
            Console.WriteLine("\r\n");

            //Modifica ListaA y se chequea si eso afectó a ListaC
            ListaA[0] = "CACATÚA";
            for (int cont = 0; cont < ListaC.Count; cont++) Console.Write(ListaC[cont] + "; ");
        }
    }
}
```



```
Consola de depuración de Mi... X + ▾
qw; A; B; C; er; 7; 8; 9; ty;
qw; A; B; C; er; 7; 8; 9; ty;
```

Ilustración 163: Insertar varios ArrayList dentro de un ArrayList

Invertir un ArrayList

Le da la vuelta al ArrayList

E/013.cs

```
using System.Collections;

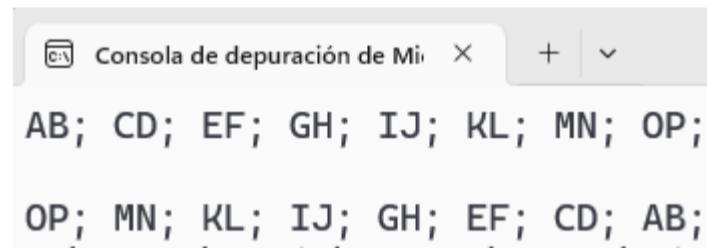
namespace Ejemplo {
    class Program {
        public static void Main() {
            //Declara la lista que almacenará cadenas
            ArrayList Ejemplo = new ArrayList();

            //Adiciona elementos a la lista
            Ejemplo.Add("AB");
            Ejemplo.Add("CD");
            Ejemplo.Add("EF");
            Ejemplo.Add("GH");
            Ejemplo.Add("IJ");
            Ejemplo.Add("KL");
            Ejemplo.Add("MN");
            Ejemplo.Add("OP");

            //Imprime el ArrayList
            for (int cont = 0; cont < Ejemplo.Count; cont++) Console.Write(Ejemplo[cont] + "; ");
            Console.WriteLine("\r\n");

            //Aplica Reverse()
            Ejemplo.Reverse();

            //Imprime de nuevo el ArrayList
            for (int cont = 0; cont < Ejemplo.Count; cont++) Console.Write(Ejemplo[cont] + "; ");
        }
    }
}
```



```
AB; CD; EF; GH; IJ; KL; MN; OP;
OP; MN; KL; IJ; GH; EF; CD; AB;
```

Ilustración 164: Invertir un ArrayList

```
using System.Collections;

namespace Ejemplo {
    class Program {
        public static void Main() {
            //Declara la lista que almacenará cadenas
            ArrayList Ejemplo = new ArrayList();

            //Adiciona elementos a la lista
            Ejemplo.Add("AB");
            Ejemplo.Add("CD");
            Ejemplo.Add("EF");
            Ejemplo.Add("GH");
            Ejemplo.Add("IJ");
            Ejemplo.Add("KL");
            Ejemplo.Add("MN");
            Ejemplo.Add("OP");

            //Imprime el ArrayList
            for (int cont = 0; cont < Ejemplo.Count; cont++) Console.Write(Ejemplo[cont] + "; ");
            Console.WriteLine("\r\n");

            //Aplica Reverse(posicion, cantidad)
            int posicion = 2;
            int cantidad = 4;
            Ejemplo.Reverse(posicion, cantidad);

            //Imprime de nuevo el ArrayList
            for (int cont = 0; cont < Ejemplo.Count; cont++) Console.Write(Ejemplo[cont] + "; ");
        }
    }
}
```



AB; CD; EF; GH; IJ; KL; MN; OP;
AB; CD; KL; IJ; GH; EF; MN; OP;

Ilustración 165: Invertir un rango de datos en el ArrayList

```
using System.Collections;

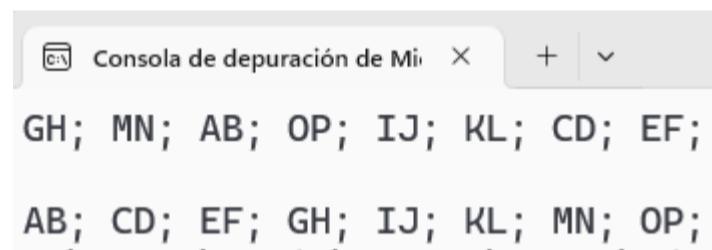
namespace Ejemplo {
    class Program {
        public static void Main() {
            //Declara la lista que almacenará cadenas
            ArrayList Ejemplo = new ArrayList();

            //Adiciona elementos a la lista
            Ejemplo.Add("GH");
            Ejemplo.Add("MN");
            Ejemplo.Add("AB");
            Ejemplo.Add("OP");
            Ejemplo.Add("IJ");
            Ejemplo.Add("KL");
            Ejemplo.Add("CD");
            Ejemplo.Add("EF");

            //Imprime el ArrayList
            for (int cont = 0; cont < Ejemplo.Count; cont++) Console.Write(Ejemplo[cont] + "; ");
            Console.WriteLine("\r\n");

            //Ordena el ArrayList
            Ejemplo.Sort();

            //Imprime de nuevo el ArrayList
            for (int cont = 0; cont < Ejemplo.Count; cont++) Console.Write(Ejemplo[cont] + "; ");
        }
    }
}
```



```
GH; MN; AB; OP; IJ; KL; CD; EF;
AB; CD; EF; GH; IJ; KL; MN; OP;
```

Ilustración 166: Ordena un ArrayList

Búsqueda Binaria

Una vez el ArrayList es ordenado, se puede hacer una búsqueda binaria.

E/016.cs

```
using System.Collections;

namespace Ejemplo {
    class Program {
        public static void Main() {
            //Declara la lista que almacenará cadenas
            ArrayList Ejemplo = new ArrayList();

            //Adiciona elementos a la lista
            Ejemplo.Add("GH");
            Ejemplo.Add("MN");
            Ejemplo.Add("AB");
            Ejemplo.Add("KL");
            Ejemplo.Add("OP");
            Ejemplo.Add("IJ");
            Ejemplo.Add("CD");
            Ejemplo.Add("EF");

            //Imprime el ArrayList
            Console.WriteLine("ArrayList Original");
            for (int cont = 0; cont < Ejemplo.Count; cont++) Console.Write(Ejemplo[cont] + "; ");
            Console.WriteLine("\r\n");

            //Ordena el ArrayList
            Ejemplo.Sort();

            //Imprime de nuevo el ArrayList
            Console.WriteLine("ArrayList Ordenado");
            for (int cont = 0; cont < Ejemplo.Count; cont++) Console.Write(Ejemplo[cont] + "; ");
            Console.WriteLine("\r\n");

            //Busca en forma binaria en el ArrayList
            string Buscar = "KL";
            int posicion = Ejemplo.BinarySearch(Buscar);
            Console.WriteLine("Buscando a: " + Buscar + " encontrada en: " + posicion.ToString());
        }
    }
}
```

```
Consola de depuración de Mi... X + ▾
ArrayList Original
GH; MN; AB; KL; OP; IJ; CD; EF;

ArrayList Ordenado
AB; CD; EF; GH; IJ; KL; MN; OP;

Buscando a: KL encontrada en: 5
```

Ilustración 167: Búsqueda Binaria

Capacidad del ArrayList

A medida que el ArrayList se le van adicionando elementos, su capacidad va aumentando siempre por encima del número de elementos almacenados.

E/017.cs

```
using System.Collections;

namespace Ejemplo {
    class Program {
        public static void Main() {
            //Declara la lista que almacenará cadenas
            ArrayList Ejemplo = new ArrayList();

            //Para agregar elementos al azar
            Random azar = new Random();

            //Va agregando elementos al azar, imprime el tamaño y la capacidad
            for (int veces = 1; veces <= 50; veces++) {
                Console.WriteLine("Iteración: " + veces.ToString());
                Console.WriteLine("Tamaño del ArrayList: " + Ejemplo.Count);
                Console.WriteLine("Capacidad del ArrayList: " + Ejemplo.Capacity + "\r\n");
                for (int cont = 1; cont <= 30; cont++) {
                    Ejemplo.Add(azar.NextDouble());
                }
            }
        }
    }
}
```

```
Consola de depuración de Mi... + ▾
Iteración: 1
Tamaño del ArrayList: 0
Capacidad del ArrayList: 0

Iteración: 2
Tamaño del ArrayList: 30
Capacidad del ArrayList: 32

Iteración: 3
Tamaño del ArrayList: 60
Capacidad del ArrayList: 64

Iteración: 4
Tamaño del ArrayList: 90
Capacidad del ArrayList: 128

Iteración: 5
Tamaño del ArrayList: 120
Capacidad del ArrayList: 128

Iteración: 6
Tamaño del ArrayList: 150
Capacidad del ArrayList: 256

Iteración: 7
Tamaño del ArrayList: 180
Capacidad del ArrayList: 256
```

Ilustración 168: Capacidad del ArrayList

Detectar el tipo de dato del elemento del ArrayList

Un ArrayList puede almacenar cualquier tipo de dato, luego si no se está seguro del tipo de dato, se hace uso de GetType()

E/018.cs

```
using System.Collections;

namespace Ejemplo {
    class Program {
        public static void Main() {
            //Declara la lista que almacenará cadenas
            ArrayList Ejemplo = new ArrayList();

            //Agrega diferentes tipos de datos
            Ejemplo.Add("Rafael Alberto Moreno Parra");
            Ejemplo.Add(720626);
            Ejemplo.Add(1.6832929);
            Ejemplo.Add('J');
            Ejemplo.Add(true);

            //Muestra el contenido y el tipo de cada elemento
            for (int cont=0; cont < Ejemplo.Count; cont++) {
                Console.WriteLine(Ejemplo[cont] + " tipo es: " + Ejemplo[cont].GetType());
            }
            Console.WriteLine("\r\n");

            //Y compara
            for (int cont = 0; cont < Ejemplo.Count; cont++) {
                Console.Write(Ejemplo[cont]);
                if (Ejemplo[cont].GetType() == typeof(int)) Console.WriteLine(" es un entero");
                if (Ejemplo[cont].GetType() == typeof(char)) Console.WriteLine(" es un caracter");
                if (Ejemplo[cont].GetType() == typeof(double)) Console.WriteLine(" es un real");
                if (Ejemplo[cont].GetType() == typeof(string)) Console.WriteLine(" es una cadena");
                if (Ejemplo[cont].GetType() == typeof(bool)) Console.WriteLine(" es un booleano");
            }
        }
    }
}
```

```
Rafael Alberto Moreno Parra tipo es: System.String
720626 tipo es: System.Int32
1,6832929 tipo es: System.Double
J tipo es: System.Char
True tipo es: System.Boolean

Rafael Alberto Moreno Parra es una cadena
720626 es un entero
1,6832929 es un real
J es un caracter
True es un booleano
```

Ilustración 169: Detectar el tipo de dato del elemento del ArrayList

List

List es similar a ArrayList. La diferencia es que List exige un mismo tipo de datos para todos los elementos. Los métodos de List son similares a ArrayList. La ventaja es que List es mucho más rápido que ArrayList.

La sintaxis para definir un List es:

```
List<tipo de dato> Nombre = new List<tipo de dato>();
```

Métodos similares a los de ArrayList

Los métodos que se documentaron anteriormente para ArrayList (excepto el de detectar el tipo de dato) se utilizan en List. Se muestra su uso a continuación:

E/019.cs

```
using System.Collections;

namespace Ejemplo {
    class Program {
        public static void Main() {
            //Declara la lista que almacenará cadenas
            List<string> ListaAnimales = new List<string>();

            //Adiciona elementos a la lista
            ListaAnimales.Add("Ballena");
            ListaAnimales.Add("Tortuga");
            ListaAnimales.Add("Tiburón");
            ListaAnimales.Add("Hipocampo");
            ListaAnimales.Add("Delfín");
            ListaAnimales.Add("Pulpo");
            ListaAnimales.Add("Coral");
            ListaAnimales.Add("Pingüinos");
            ListaAnimales.Add("Calamar");
            ListaAnimales.Add("Gaviota");
            ListaAnimales.Add("Foca");
            ListaAnimales.Add("Manatíes");
            ListaAnimales.Add("Orca");
            ListaAnimales.Add("Medusas");
            ListaAnimales.Add("Mejillones");
            ListaAnimales.Add("Caracoles");

            //Tamaño de la lista
            int tamano = ListaAnimales.Count;
            Console.WriteLine("Tamaño de la lista: " + tamano);

            //Traer un determinado elemento de la lista
            int posicion = 7;
            string texto = ListaAnimales[posicion].ToString();
            Console.WriteLine("Elemento en la posición " + posicion + " es: " + texto);

            //Nos dice si existe un determinado elemento en la lista
            string buscar = "Foca";
            bool Existe = ListaAnimales.Contains(buscar);
            Console.WriteLine("Busca: " + buscar + " Resultado: " + Existe);

            //Nos dice la posición donde encontró el elemento en la lista
            int posBusca = ListaAnimales.IndexOf(buscar);
            Console.WriteLine("Busca: " + buscar + " Posición: " + posBusca.ToString() + "\r\n");

            //Imprime la lista
            Console.WriteLine("Lista Original");
            ImprimeLista(ListaAnimales);

            //Retira elemento de la lista
            Console.WriteLine("Retira HipoCampo");
            ListaAnimales.Remove("Hipocampo");
            ImprimeLista(ListaAnimales);

            //Elimina el objeto de determinada posición.
            Console.WriteLine("Retira Elemento posición 5");
            ListaAnimales.RemoveAt(5);
            ImprimeLista(ListaAnimales);

            //Cambia una cadena en la lista
            Console.WriteLine("Modifica elemento posición 3");
            ListaAnimales[3] = "ORNITORRINCO";
            ImprimeLista(ListaAnimales);
        }
    }
}
```

```

//Inserta una cadena en la posición 4 de la lista
Console.WriteLine("Inserta elemento posición 4");
ListaAnimales.Insert(4, "CACATÚA");
ImprimeLista(ListaAnimales);

//Genera nueva lista
int posicionInicial = 5;
int cantidad = 3;
List<string> nuevaLista = ListaAnimales.GetRange(posicionInicial, cantidad);
Console.WriteLine("Nueva lista");
ImprimeLista(nuevaLista);

//Recorrido con foreach
Console.WriteLine("Recorrido con foreach");
foreach (Object objeto in ListaAnimales) Console.Write("{0}{1}", ";", objeto);
Console.WriteLine("\r\n");

//Recorrido con for
Console.WriteLine("Recorrido con for");
for (int cont = 0; cont < ListaAnimales.Count; cont++)
    Console.Write("{0}{1}", ";", ListaAnimales[cont]);
Console.WriteLine("\r\n");

//Recorrido con un IEnumerator
Console.WriteLine("Recorrido con un IEnumerator");
IEnumerator Iobjeto = ListaAnimales.GetEnumerator();
while (Iobjeto.MoveNext()) Console.Write("{0}{1}", ";", Iobjeto.Current);
Console.WriteLine("\r\n");

//Guarda el List en un arreglo estático de tipo string
Console.WriteLine("Pasa el List a un arreglo estático de tipo cadena");
string[] cadenas = ListaAnimales.ToArray();
for (int cont = 0; cont < cadenas.Length; cont++) Console.Write(cadenas[cont] + ";");
Console.WriteLine("\r\n");

//Adiciona un arreglo estático al List
Console.WriteLine("Adiciona un arreglo estático al List");
string[] Cadenas = { "Gato", "Perro", "Conejo", "Liebre", "Oveja" };
ListaAnimales.AddRange(Cadenas);
ImprimeLista(ListaAnimales);

//Inserta un arreglo estático al List
Console.WriteLine("Inserta un arreglo estático al List");
string[] Aves = { "Azulejo", "Bichofue", "Paloma", "Gavilán", "Halcón" };
int posicionInserta = 4;
ListaAnimales.InsertRange(posicionInserta, Aves);
ImprimeLista(ListaAnimales);

//Invierte la lista
Console.WriteLine("Invierte la lista");
ListaAnimales.Reverse();
ImprimeLista(ListaAnimales);

//Ordena el List
Console.WriteLine("Ordena la lista");
ListaAnimales.Sort();
ImprimeLista(ListaAnimales);

//Busca en forma binaria en el List
Console.WriteLine("Búsqueda binaria en la lista");
string Buscar = "Gato";
int buscado = ListaAnimales.BinarySearch(Buscar);
Console.WriteLine("Buscando a: " + Buscar + " encontrado en: " + buscado.ToString() +
"\r\n");

//Elimina un rango de elementos del List
Console.WriteLine("Elimina un rango de elementos");
int PosBorra = 1;
int CantidadBorra = 4;
ListaAnimales.RemoveRange(PosBorra, CantidadBorra);
ImprimeLista(ListaAnimales);

//Limpia el List
Console.WriteLine("Borra el List");
Console.WriteLine("(Antes) Total de elementos: " + ListaAnimales.Count);
ListaAnimales.Clear();
Console.WriteLine("(Después) Total de elementos: " + ListaAnimales.Count);
}

```

```

        static void ImprimeLista(List<string> listado) {
            for (int cont = 0; cont < listado.Count; cont++)
                Console.Write("{0}{1}", ";", listado[cont]);
            Console.WriteLine("\r\n");
        }
    }
}

```

Consola de depuración de Mi... X + ▾

Tamaño de la lista: 16
Elemento en la posición 7 es: Pingüinos
Busca: Foca Resultado: True
Busca: Foca Posición: 10

Lista Original
;Ballena;Tortuga;Tiburón;Hipocampo;Delfín;Pulpo;Coral;Pingüinos;Calamar;Gaviota;Foca;Manaties;Orca;Medusas;Mejillones;Caracoles

Retira HipoCampos
;Ballena;Tortuga;Tiburón;Delfín;Pulpo;Coral;Pingüinos;Calamar;Gaviota;Foca;Manaties;Orca;Medusas;Mejillones;Caracoles

Retira Elemento posición 5
;Ballena;Tortuga;Tiburón;Delfín;Pulpo;Pingüinos;Calamar;Gaviota;Foca;Manaties;Orca;Medusas;Mejillones;Caracoles

Modifica elemento posición 3
;Ballena;Tortuga;Tiburón;ORNITORRINCO;Pulpo;Pingüinos;Calamar;Gaviota;Foca;Manaties;Orca;Medusas;Mejillones;Caracoles

Inserta elemento posición 4
;Ballena;Tortuga;Tiburón;ORNITORRINCO;CACATÚA;Pulpo;Pingüinos;Calamar;Gaviota;Foca;Manaties;Orca;Medusas;Mejillones;Caracoles

Nueva lista
;Pulpo;Pingüinos;Calamar

Recorrido con foreach
;Ballena;Tortuga;Tiburón;ORNITORRINCO;CACATÚA;Pulpo;Pingüinos;Calamar;Gaviota;Foca;Manaties;Orca;Medusas;Mejillones;Caracoles

Recorrido con for
;Ballena;Tortuga;Tiburón;ORNITORRINCO;CACATÚA;Pulpo;Pingüinos;Calamar;Gaviota;Foca;Manaties;Orca;Medusas;Mejillones;Caracoles

Ilustración 170: List, métodos

Consola de depuración de Mi... X + ▾

Recorrido con un IEnumerator
;Ballena;Tortuga;Tiburón;ORNITORRINCO;CACATÚA;Pulpo;Pingüinos;Calamar;Gaviota;Foca;Manaties;Orca;Medusas;Mejillones;Caracoles

Pasa el List a un arreglo estático de tipo cadena
Ballena;Tortuga;Tiburón;ORNITORRINCO;CACATÚA;Pulpo;Pingüinos;Calamar;Gaviota;Foca;Manaties;Orca;Medusas;Mejillones;Caracoles;

Adiciona un arreglo estático al List
;Ballena;Tortuga;Tiburón;ORNITORRINCO;CACATÚA;Pulpo;Pingüinos;Calamar;Gaviota;Foca;Manaties;Orca;Medusas;Mejillones;Caracoles;Gato;Perro;Conejo;Liebre;Oveja

Inserta un arreglo estático al List
;Ballena;Tortuga;Tiburón;ORNITORRINCO;Azulejo;Bichofue;Paloma;Gavilán;Halcón;CACATÚA;Pulpo;Pingüinos;Calamar;Gaviota;Foca;Manaties;Orca;Medusas;Mejillones;Caracoles;Gato;Perro;Conejo;Liebre;Oveja

Invierte la lista
;Oveja;Liebre;Conejo;Perro;Gato;Caracoles;Mejillones;Medusas;Orca;Manaties;Foca;Gaviota;Calamar;Pingüinos;Pulpo;CACATÚA;Halcón;Gavilán;Paloma;Bichofue;Azulejo;ORNITORRINCO;Tiburón;Tortuga;Ballena

Ordena la lista
;Azulejo;Ballena;Bichofue;CACATÚA;Calamar;Caracoles;Conejo;Foca;Gato;Gavilán;Gaviota;Halcón;Liebre;Manaties;Medusas;Mejillones;Orca;ORNITORRINCO;Oveja;Paloma;Perro;Pingüinos;Pulpo;Tiburón;Tortuga

Búsqueda binaria en la lista
Buscando a: Gato encontrado en: 8

Elimina un rango de elementos
;Azulejo;Caracoles;Conejo;Foca;Gato;Gavilán;Gaviota;Halcón;Liebre;Manaties;Medusas;Mejillones;Orca;ORNITORRINCO;Oveja;Paloma;Perro;Pingüinos;Pulpo;Tiburón;Tortuga

Borra el List
(Antes) Total de elementos: 21
(Después) Total de elementos: 0

Ilustración 171: List, métodos

Métricas: Comparativa de desempeño de ArrayList vs List vs Arreglo estático

¿Cuál estructura tiene mejor desempeño? Para lograr esta comparativa, se utilizó el algoritmo de ordenación de burbuja, el cual hace uso intensivo de operaciones de lectura y cambio de valores en la estructura de memoria (por eso es tan lento ese algoritmo), pero será útil para hacer la comparativa. El programa a continuación:

Ordenando con tipo char

E/020a.cs

```
using System.Collections;
using System.Collections.Generic;
using System.Diagnostics;

namespace Ejemplo {
    class Program {
        static void Main() {
            /* Prueba de velocidad de los diferentes tipos de estructuras:
             * arreglo estático, ArrayList, List
             * Se usará el método de ordenación de burbuja en el que
             * hace una gran cantidad de lectura y escritura sobre la estructura
             * (por eso es el más lento pero muy bueno para hacer esta comparativa)
             * */

            int minimoOrdene = 1000; //Mínimo número de elementos a ordenar
            int maximoOrdene = 9000; //Máximo número de elementos a ordenar
            int avanceOrdene = 500; //Avance de elementos a ordenar

            /* Número de pruebas por ordenamiento
             * Luego el tiempo de ordenar N elementos es el promedio de esas pruebas
             * así se evita que por algún motivo los tiempos tengan picos o valles */
            int numPruebas = 30;

            //Limite es el tamaño de datos que se van a ordenar
            for (int Limite = minimoOrdene; Limite <= maximoOrdene; Limite += avanceOrdene)
                Ordenamiento(Limite, numPruebas);

            Console.WriteLine("\r\nFinal de la prueba");
        }

        static void Ordenamiento(int Limite, int numPruebas) {
            Random azar = new Random();

            //Las estructuras usadas: arreglo estático, ArrayList, List
            char[] numerosA = new char[Limite];
            char[] numerosB = new char[Limite];
            ArrayList arraylist = new ArrayList();
            List<char> list = new List<char>();

            //Medidor de tiempos
            Stopwatch temporizador = new Stopwatch();

            //Almacena los tiempos de cada método de ordenación
            long TParreglo = 0, TParraylist = 0, TPlist = 0;

            //Para evitar una optimización agresiva del compilador, se acumula el valor de diferentes
            //posiciones del arreglo ordenado.
            string valor = "";

            //Para disminuir picos o valles en el tiempo, se hacen varias pruebas
            for (int prueba = 1; prueba <= numPruebas; prueba++) {

                //Llena con valores al azar el arreglo
                LlenaAzar(numerosA, azar);

                //Ordenación por Burbuja ArrayList
                arraylist.Clear();
                arraylist.AddRange(numerosA);
                temporizador.Reset();
                temporizador.Start();
                BurbujaArrayList(arraylist);
                TParraylist += temporizador.ElapsedMilliseconds;
                valor += Convert.ToChar(arraylist[0]);

                //Ordenación por Burbuja List
                list.Clear();
                list.AddRange(numerosA);
                temporizador.Reset();
            }
        }
    }
}
```

```

temporizador.Start();
BurbujaList(list);
TPlist += temporizador.ElapsedMilliseconds;
valor += list[1];

//Ordenación por Burbuja Arreglo estático unidimensional
Array.Copy(numeroA, 0, numeroB, 0, numeroA.Length);
temporizador.Reset();
temporizador.Start();
BurbujaArreglo(numeroB);
TParreglo += temporizador.ElapsedMilliseconds;
valor += numeroB[2];
}

double Tarreglo = (double)TParreglo / numPruebas;
double Tarraylist = (double)TParraylist / numPruebas;
double Tlist = (double)TPlist / numPruebas;

Console.WriteLine("=====");
Console.WriteLine("\r\nValor de control (contra optimización agresiva): " +
valor.ToString());
Console.WriteLine("Número de elementos: " + Limite.ToString());
Console.WriteLine("Burbuja arreglo estático unidimensional, tiempo promedio en milisegundos:
" + Tarreglo.ToString());
Console.WriteLine("Burbuja ArrayList, tiempo promedio en milisegundos: " +
Tarraylist.ToString());
Console.WriteLine("Burbuja List, tiempo promedio en milisegundos: " + Tlist.ToString());
}

//Llena el arreglo unidimensional con valores aleatorios
static void LlenaAzar(char[] numeroA, Random azar) {
    string Permitido = "abcdefghijklmnñopqrstuvwxyzABCDEFGHIJKLMNÑOPQRSTUVWXYZ";
    for (int cont = 0; cont < numeroA.Length; cont++)
        numeroA[cont] = Permitido[azar.Next(Permitido.Length)];
}

//Ordenamiento por burbuja usando ArrayList
static void BurbujaArrayList(ArrayList arraylist) {
    int tamano = arraylist.Count;
    object tmp;
    for (int i = 0; i < tamano - 1; i++) {
        for (int j = 0; j < tamano - 1; j++) {
            if (Convert.ToChar(arraylist[j]) > Convert.ToChar(arraylist[j + 1])) {
                tmp = arraylist[j];
                arraylist[j] = arraylist[j + 1];
                arraylist[j + 1] = tmp;
            }
        }
    }
}

//Ordenamiento por burbuja usando List
static void BurbujaList(List<char> list) {
    int tamano = list.Count;
    char tmp;
    for (int i = 0; i < tamano - 1; i++) {
        for (int j = 0; j < tamano - 1; j++) {
            if (list[j] > list[j + 1]) {
                tmp = list[j];
                list[j] = list[j + 1];
                list[j + 1] = tmp;
            }
        }
    }
}

//Ordenamiento por burbuja usando arreglo unidimensional estático
static void BurbujaArreglo(char[] arregloestatico) {
    int tamano = arregloestatico.Length;
    char tmp;
    for (int i = 0; i < tamano - 1; i++) {
        for (int j = 0; j < tamano - 1; j++) {
            if (arregloestatico[j] > arregloestatico[j + 1]) {
                tmp = arregloestatico[j];
                arregloestatico[j] = arregloestatico[j + 1];
                arregloestatico[j + 1] = tmp;
            }
        }
    }
}

```

}

Valor de control (contra optimización agresiva): AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA/AAAAAA/AAAAAA

Número de elementos: 6000

Burbuja arreglo estático unidimensional, tiempo promedio en milisegundos: 38,5666666666667

Burbuja ArrayList, tiempo promedio en milisegundos: 281,8

Burbuja List, tiempo promedio en milisegundos: 50,5

=====

Valor de control (contra optimización agresiva): AAAAAAAAAAAAAAAAAAAAAAAA/AAAAAA/AAAAAA

Número de elementos: 6500

Burbuja arreglo estático unidimensional, tiempo promedio en milisegundos: 45,7333333333334

Burbuja ArrayList, tiempo promedio en milisegundos: 331,5

Burbuja List, tiempo promedio en milisegundos: 60,1

=====

Valor de control (contra optimización agresiva): AAAAAAAAAAAAAAAAAAAAAAAA/AAAAAA/AAAAAA

Número de elementos: 7000

Burbuja arreglo estático unidimensional, tiempo promedio en milisegundos: 53,2

Burbuja ArrayList, tiempo promedio en milisegundos: 383,2

Burbuja List, tiempo promedio en milisegundos: 69,3

Ilustración 172: Métrica: Ordenando con tipo char

```

using System.Collections;
using System.Collections.Generic;
using System.Diagnostics;

namespace Ejemplo {
    class Program {
        static void Main() {
            /* Prueba de velocidad de los diferentes tipos de estructuras:
             * arreglo estático, ArrayList, List
             * Se usará el método de ordenación de burbuja en el que
             * hace una gran cantidad de lectura y escritura sobre la estructura
             * (por eso es el más lento pero muy bueno para hacer esta comparativa)
             * */

            int minimoOrdene = 1000; //Mínimo número de elementos a ordenar
            int maximoOrdene = 9000; //Máximo número de elementos a ordenar
            int avanceOrdene = 500; //Avance de elementos a ordenar

            /* Número de pruebas por ordenamiento
             * Luego el tiempo de ordenar N elementos es el promedio de esas pruebas
             * así se evita que por algún motivo los tiempos tengan picos o valles */
            int numPruebas = 30;

            //Limite es el tamaño de datos que se van a ordenar
            for (int Limite = minimoOrdene; Limite <= maximoOrdene; Limite += avanceOrdene)
                Ordenamiento(Limite, numPruebas);

            Console.WriteLine("\r\nFinal de la prueba");
        }

        static void Ordenamiento(int Limite, int numPruebas) {
            Random azar = new Random();

            //Las estructuras usadas: arreglo estático, ArrayList, List
            double[] numerosA = new double[Limite];
            double[] numerosB = new double[Limite];
            ArrayList arraylist = new ArrayList();
            List<double> list = new List<double>();

            //Medidor de tiempos
            Stopwatch temporizador = new Stopwatch();

            //Almacena los tiempos de cada método de ordenación
            long TParreglo = 0, TParraylist = 0, TPlist = 0;

            //Para evitar una optimización agresiva del compilador, se acumula el valor de diferentes
            //posiciones del arreglo ordenado.
            double valor = 0;

            //Para disminuir picos o valles en el tiempo, se hacen varias pruebas
            for (int prueba = 1; prueba <= numPruebas; prueba++) {

                //Llena con valores al azar el arreglo
                LlenaAzar(numerosA, azar);

                //Ordenación por Burbuja ArrayList
                arraylist.Clear();
                arraylist.AddRange(numerosA);
                temporizador.Reset();
                temporizador.Start();
                BurbujaArrayList(arraylist);
                TParraylist += temporizador.ElapsedMilliseconds;
                valor += Convert.ToDouble(arraylist[0]);

                //Ordenación por Burbuja List
                list.Clear();
                list.AddRange(numerosA);
                temporizador.Reset();
                temporizador.Start();
                BurbujaList(list);
                TPlist += temporizador.ElapsedMilliseconds;
                valor += list[1];

                //Ordenación por Burbuja Arreglo estático unidimensional
                Array.Copy(numerosA, 0, numerosB, 0, numerosA.Length);
            }
        }
    }
}

```

```

temporizador.Reset();
temporizador.Start();
BurbujaArreglo(numerosB);
TParreglo += temporizador.ElapsedMilliseconds;
valor += numerosB[2];
}

double Tarreglo = (double)TParreglo / numPruebas;
double Tarraylist = (double)TParreglo / numPruebas;
double Tlist = (double)TList / numPruebas;

Console.WriteLine("=====");
Console.WriteLine("\r\nValor de control (contra optimización agresiva): " +
valor.ToString());
Console.WriteLine("Número de elementos: " + Limite.ToString());
Console.WriteLine("Burbuja arreglo estático unidimensional, tiempo promedio en milisegundos:
" + Tarreglo.ToString());
Console.WriteLine("Burbuja ArrayList, tiempo promedio en milisegundos: " + 
Tarraylist.ToString());
Console.WriteLine("Burbuja List, tiempo promedio en milisegundos: " + Tlist.ToString());
}

//Llena el arreglo unidimensional con valores aleatorios
static void LlenaAzar(double[] numerosA, Random azar) {
    for (int cont = 0; cont < numerosA.Length; cont++)
        numerosA[cont] = azar.NextDouble();
}

//Ordenamiento por burbuja usando ArrayList
static void BurbujaArrayList(ArrayList arraylist) {
    int tamano = arraylist.Count;
    object tmp;
    for (int i = 0; i < tamano - 1; i++) {
        for (int j = 0; j < tamano - 1; j++) {
            if (Convert.ToDouble(arraylist[j]) > Convert.ToDouble(arraylist[j + 1])) {
                tmp = arraylist[j];
                arraylist[j] = arraylist[j + 1];
                arraylist[j + 1] = tmp;
            }
        }
    }
}

//Ordenamiento por burbuja usando List
static void BurbujaList(List<double> list) {
    int tamano = list.Count;
    double tmp;
    for (int i = 0; i < tamano - 1; i++) {
        for (int j = 0; j < tamano - 1; j++) {
            if (list[j] > list[j + 1]) {
                tmp = list[j];
                list[j] = list[j + 1];
                list[j + 1] = tmp;
            }
        }
    }
}

//Ordenamiento por burbuja usando arreglo unidimensional estático
static void BurbujaArreglo(double[] arregloestatico) {
    int tamano = arregloestatico.Length;
    double tmp;
    for (int i = 0; i < tamano - 1; i++) {
        for (int j = 0; j < tamano - 1; j++) {
            if (arregloestatico[j] > arregloestatico[j + 1]) {
                tmp = arregloestatico[j];
                arregloestatico[j] = arregloestatico[j + 1];
                arregloestatico[j + 1] = tmp;
            }
        }
    }
}
}

```

Valor de control (contra optimización agresiva): 0,0470277561469592
Número de elementos: 5000
Burbuja arreglo estático unidimensional, tiempo promedio en milisegundos: 24,1
Burbuja ArrayList, tiempo promedio en milisegundos: 197,8
Burbuja List, tiempo promedio en milisegundos: 33,7

=====

Valor de control (contra optimización agresiva): 0,03263250121835315
Número de elementos: 5500
Burbuja arreglo estático unidimensional, tiempo promedio en milisegundos: 31,3
Burbuja ArrayList, tiempo promedio en milisegundos: 228,4666666666667
Burbuja List, tiempo promedio en milisegundos: 42,4333333333333

=====

Valor de control (contra optimización agresiva): 0,03364929442034292
Número de elementos: 6000
Burbuja arreglo estático unidimensional, tiempo promedio en milisegundos: 35,9
Burbuja ArrayList, tiempo promedio en milisegundos: 258,633333333333
Burbuja List, tiempo promedio en milisegundos: 50,5333333333333

=====

Ilustración 173: Métrica: Ordenando con tipo double

```

using System.Collections;
using System.Collections.Generic;
using System.Diagnostics;

namespace Ejemplo {
    class Program {
        static void Main() {
            /* Prueba de velocidad de los diferentes tipos de estructuras:
             * arreglo estático, ArrayList, List
             * Se usará el método de ordenación de burbuja en el que
             * hace una gran cantidad de lectura y escritura sobre la estructura
             * (por eso es el más lento pero muy bueno para hacer esta comparativa)
             * */

            int minimoOrdene = 1000; //Mínimo número de elementos a ordenar
            int maximoOrdene = 9000; //Máximo número de elementos a ordenar
            int avanceOrdene = 500; //Avance de elementos a ordenar

            /* Número de pruebas por ordenamiento
             * Luego el tiempo de ordenar N elementos es el promedio de esas pruebas
             * así se evita que por algún motivo los tiempos tengan picos o valles */
            int numPruebas = 30;

            //Límite es el tamaño de datos que se van a ordenar
            for (int Limite = minimoOrdene; Limite <= maximoOrdene; Limite += avanceOrdene)
                Ordenamiento(Limite, numPruebas);

            Console.WriteLine("\r\nFinal de la prueba");
        }

        static void Ordenamiento(int Limite, int numPruebas) {
            Random azar = new Random();

            //Las estructuras usadas: arreglo estático, ArrayList, List
            int[] numerosA = new int[Limite];
            int[] numerosB = new int[Limite];
            ArrayList arraylist = new ArrayList();
            List<int> list = new List<int>();

            //Medidor de tiempos
            Stopwatch temporizador = new Stopwatch();

            //Almacena los tiempos de cada método de ordenación
            long TParreglo = 0, TParraylist = 0, TPlist = 0;

            //Para evitar una optimización agresiva del compilador, se acumula el valor de diferentes
            //posiciones del arreglo ordenado.
            long valor = 0;

            //Para disminuir picos o valles en el tiempo, se hacen varias pruebas
            for (int prueba = 1; prueba <= numPruebas; prueba++) {

                //Llena con valores al azar el arreglo
                LlenaAzar(numerosA, azar);

                //Ordenación por Burbuja ArrayList
                arraylist.Clear();
                arraylist.AddRange(numerosA);
                temporizador.Reset();
                temporizador.Start();
                BurbujaArrayList(arraylist);
                TParraylist += temporizador.ElapsedMilliseconds;
                valor += Convert.ToInt32(arraylist[0]);

                //Ordenación por Burbuja List
                list.Clear();
                list.AddRange(numerosA);
                temporizador.Reset();
                temporizador.Start();
                BurbujaList(list);
                TPlist += temporizador.ElapsedMilliseconds;
                valor += list[1];

                //Ordenación por Burbuja Arreglo estático unidimensional
                Array.Copy(numerosA, 0, numerosB, 0, numerosA.Length);
                temporizador.Reset();
                temporizador.Start();
            }
        }
    }
}

```

```

        BurbujaArreglo(numerosB);
        TParreglo += temporizador.ElapsedMilliseconds;
        valor += numerosB[2];
    }

    double Tarreglo = (double)TParreglo / numPruebas;
    double Tarraylist = (double)TArrayList / numPruebas;
    double Tlist = (double)TList / numPruebas;

    Console.WriteLine("=====");
    Console.WriteLine("\r\nValor de control (contra optimización agresiva): " +
valor.ToString());
    Console.WriteLine("Número de elementos: " + Limite.ToString());
    Console.WriteLine("Burbuja arreglo estático unidimensional, tiempo promedio en milisegundos:
" + Tarreglo.ToString());
    Console.WriteLine("Burbuja ArrayList, tiempo promedio en milisegundos: " +
Tarraylist.ToString());
    Console.WriteLine("Burbuja List, tiempo promedio en milisegundos: " + Tlist.ToString());
}

//Llena el arreglo unidimensional con valores aleatorios
static void LlenaAzar(int[] numerosA, Random azar) {
    for (int cont = 0; cont < numerosA.Length; cont++)
        numerosA[cont] = azar.Next(0, 10000);
}

//Ordenamiento por burbuja usando ArrayList
static void BurbujaArrayList(ArrayList arraylist) {
    int tamano = arraylist.Count;
    object tmp;
    for (int i = 0; i < tamano - 1; i++) {
        for (int j = 0; j < tamano - 1; j++) {
            if (Convert.ToInt32(arraylist[j]) > Convert.ToInt32(arraylist[j + 1])) {
                tmp = arraylist[j];
                arraylist[j] = arraylist[j + 1];
                arraylist[j + 1] = tmp;
            }
        }
    }
}

//Ordenamiento por burbuja usando List
static void BurbujaList(List<int> list) {
    int tamano = list.Count;
    int tmp;
    for (int i = 0; i < tamano - 1; i++) {
        for (int j = 0; j < tamano - 1; j++) {
            if (list[j] > list[j + 1]) {
                tmp = list[j];
                list[j] = list[j + 1];
                list[j + 1] = tmp;
            }
        }
    }
}

//Ordenamiento por burbuja usando arreglo unidimensional estático
static void BurbujaArreglo(int[] arregloestatico) {
    int tamano = arregloestatico.Length;
    int tmp;
    for (int i = 0; i < tamano - 1; i++) {
        for (int j = 0; j < tamano - 1; j++) {
            if (arregloestatico[j] > arregloestatico[j + 1]) {
                tmp = arregloestatico[j];
                arregloestatico[j] = arregloestatico[j + 1];
                arregloestatico[j + 1] = tmp;
            }
        }
    }
}
}

```

Valor de control (contra optimización agresiva): 232
Número de elementos: 7000
Burbuja arreglo estático unidimensional, tiempo promedio en milisegundos: 39,06666666666667
Burbuja ArrayList, tiempo promedio en milisegundos: 366,2
Burbuja List, tiempo promedio en milisegundos: 72,6333333333334
=====

Valor de control (contra optimización agresiva): 216
Número de elementos: 7500
Burbuja arreglo estático unidimensional, tiempo promedio en milisegundos: 44,96666666666667
Burbuja ArrayList, tiempo promedio en milisegundos: 394,6
Burbuja List, tiempo promedio en milisegundos: 82,2
=====

Valor de control (contra optimización agresiva): 167
Número de elementos: 8000
Burbuja arreglo estático unidimensional, tiempo promedio en milisegundos: 53,1
Burbuja ArrayList, tiempo promedio en milisegundos: 455,0666666666666
Burbuja List, tiempo promedio en milisegundos: 96,0333333333333
=====

Ilustración 174: Métrica: Ordenando con tipo int

Lista de objetos

Un frecuente uso de List es para tener un listado de objetos del mismo tipo, no solo tipos de datos nativo. Se requiere entonces dos clases, en una está definido el tipo de objeto a guardar y en la otra se crean, adicionan, actualizan y borran del List

E/021.cs

```
namespace Ejemplo {
    class Ejemplo {
        //Atributos variados
        public int ValorEntero { get; set; }
        public double NumeroReal { get; set; }
        public char Caracter { get; set; }
        public string Cadena { get; set; }

        //Constructor
        public Ejemplo(int ValorEntero, double NumeroReal, char Caracter, string Cadena) {
            this.ValorEntero = ValorEntero;
            this.NumeroReal = NumeroReal;
            this.Caracter = Caracter;
            this.Cadena = Cadena;
        }

        //Imprime los valores
        public void Imprime() {
            Console.WriteLine("\r\nEntero: " + ValorEntero.ToString());
            Console.WriteLine("Real: " + NumeroReal.ToString());
            Console.WriteLine("Caracter: " + Caracter.ToString());
            Console.WriteLine("Cadena: [" + Cadena + "]");
        }
    }

    class Program {
        static void Main() {
            List<Ejemplo> listado = new List<Ejemplo>();

            //Adiciona objetos a la lista
            listado.Add(new Ejemplo(16, 83.29, 'R', "Ruiseñor"));
            listado.Add(new Ejemplo(29, 89.7, 'A', "Águila"));
            listado.Add(new Ejemplo(2, 80.19, 'M', "Manatí"));
            listado.Add(new Ejemplo(95, 7.21, 'P', "Puma"));

            //Llama al método de imprimir del objeto
            for (int cont = 0; cont < listado.Count; cont++) listado[cont].Imprime();

            //Inserta un objeto
            listado.Insert(1, new Ejemplo(88, 3.33, 'Z', "QQQQQQQQ"));

            //Elimina un objeto
            listado.RemoveAt(3);

            //Llama al método de imprimir del objeto
            Console.WriteLine("\r\nDespués de modificar");
            for (int cont = 0; cont < listado.Count; cont++) listado[cont].Imprime();

            Console.WriteLine("\r\nFinal");
        }
    }
}
```

Entero: 16
Real: 83,29
Carácter: R
Cadena: [Ruiſeñor]

Entero: 29
Real: 89,7
Carácter: A
Cadena: [Águila]

Entero: 2
Real: 80,19
Carácter: M
Cadena: [Manatí]

Entero: 95
Real: 7,21
Carácter: P
Cadena: [Puma]

Después de modificar

Entero: 16
Real: 83,29
Carácter: R
Cadena: [Ruiſeñor]

Entero: 88
Real: 3,33
Carácter: Z
Cadena: [QQQQQQQQ]

Entero: 29
Real: 89,7
Carácter: A
Cadena: [Águila]

Ilustración 175: Lista de objetos

Listas en Listas

Un objeto de una lista, a su vez tiene listas. Un ejemplo con series de TV, personajes y actores:

1. La serie tiene un nombre y se puede ver información sobre esta en IMDB.
2. Los personajes son interpretados por actores y actrices.
3. No es nada extraño que los actores participen en diversas series interpretando algún personaje en alguna serie, sólo es ver su ficha en IMDB.

E/022.cs

```
namespace Ejemplo {

    //Datos del actor o actriz
    class ActorActriz {
        public string Nombre { get; set; }
        public string URLIMDB { get; set; }

        //Constructor
        public ActorActriz(string Nombre, string URLIMDB) {
            this.Nombre = Nombre;
            this.URLIMDB = URLIMDB;
        }
    }

    //La parte que simula la capa de persistencia
    class Persistencia {
        List<ActorActriz> Actores;
        List<Serie> Series;

        //Carga datos de prueba
        public Persistencia() {
            Actores = new List<ActorActriz>();
            Series = new List<Serie>();

            //Un listado de actores y actrices
            Actores.Add(new ActorActriz("Ana María Orozco", "https://www.imdb.com/name/nm0650450/"));
            Actores.Add(new ActorActriz("Laura Londoño", "https://www.imdb.com/name/nm2256810/"));
            Actores.Add(new ActorActriz("Carolina Ramírez", "https://www.imdb.com/name/nm1329835/"));
            Actores.Add(new ActorActriz("Catherine Siachoque", "https://www.imdb.com/name/nm0796171/"));
            Actores.Add(new ActorActriz("Carmenza González", "https://www.imdb.com/name/nm1863990/"));
            Actores.Add(new ActorActriz("Andrés Londoño", "https://www.imdb.com/name/nm2150265/"));

            //Un listado de series
            Series.Add(new Serie("Yo soy Betty, la fea", "https://www.imdb.com/title/tt0233127/"));
            Series.Add(new Serie("La reina del flow", "https://www.imdb.com/title/tt8560918/"));
            Series.Add(new Serie("Café con Aroma de Mujer", "https://www.imdb.com/title/tt14471346/"));
            Series.Add(new Serie("Los Briceño", "https://www.imdb.com/title/tt10348478/"));
            Series.Add(new Serie("Distrito Salvaje", "https://www.imdb.com/title/tt8105958/"));
            Series.Add(new Serie("Mil Colmillos", "https://www.imdb.com/title/tt9701670/"));
            Series.Add(new Serie("Perdida", "https://www.imdb.com/title/tt10064124/"));

            //Añado actores y actrices a la tercera serie
            Series[2].Protagonistas.Add(Actores[1]);

            //Observe que un mismo actor o actriz puede estar en dos series distintas
            Series[0].Protagonistas.Add(Actores[0]);
            Series[6].Protagonistas.Add(Actores[0]);
        }

        //Trae datos de la serie
        public string SerieNombre(int pos) { return Series[pos].Nombre; }
        public string SerieIMDB(int pos) { return Series[pos].URLIMDB; }

        //Trae datos del actor
        public string ActorNombre(int pos) { return Actores[pos].Nombre; }
        public string ActorURL(int pos) { return Actores[pos].URLIMDB; }

        //Total de registros
        public int ActorTotal() { return Actores.Count; }
        public int SerieTotal() { return Series.Count; }

        //Adicionar actor
        public void ActorAdiciona(string Nombre, string URL) {
            Actores.Add(new ActorActriz(Nombre, URL));
        }

        //Editar actor
        public void ActorEdita(int codigo, string Nombre, string URL) {
    }
```

```

Actores[codigo].Nombre = Nombre;
Actores[codigo].URLIMDB = URL;
}

//Borrar actor
public void ActorBorra(int codigo) {
    Actores.RemoveAt(codigo);
}

//Retorna una lista de series donde el actor trabaja
public List<string> ActorTrabaja(int codigo) {
    List<string> ListaSeries = new List<string>();
    for (int cont = 0; cont < Series.Count; cont++) {
        for (int num = 0; num < Series[cont].Protagonistas.Count; num++) {
            if (Actores[codigo] == Series[cont].Protagonistas[num])
                ListaSeries.Add(Series[cont].Nombre);
        }
    }
    return ListaSeries;
}

//Adicionar serie
public void SerieAdiciona(string Nombre, string URL) {
    Series.Add(new Serie(Nombre, URL));
}

//Editar serie
public void SerieEdita(int codigo, string Nombre, string URL) {
    Series[codigo].Nombre = Nombre;
    Series[codigo].URLIMDB = URL;
}

//Borrar serie
public void SerieBorra(int codigo) {
    Series.RemoveAt(codigo);
}

//Retornar los actores que trabajan en la serie
public List<string> SerieActores(int codigo) {
    List<ActorActriz> actores = Series[codigo].Protagonistas;
    List<string> Nombres = new List<string>();
    for (int cont = 0; cont < actores.Count; cont++)
        Nombres.Add(actores[cont].Nombre);
    return Nombres;
}

//Añade un actor a una serie
public void SerieAsocia(int serie, int actor) {
    Series[serie].Protagonistas.Add(Actores[actor]);
}

//Quita un actor de una serie
public void SerieDisocia(int serie, int actor) {
    Series[serie].Protagonistas.RemoveAt(actor);
}
}

//Datos de la serie de televisión
class Serie {
    public string Nombre { get; set; }
    public string URLIMDB { get; set; }

    //Listado de actores y actrices que actúan en la serie
    public List<ActorActriz> Protagonistas = new List<ActorActriz>();

    //Constructor
    public Serie(string Nombre, string URLIMDB) {
        this.Nombre = Nombre;
        this.URLIMDB = URLIMDB;
    }
}

//La parte visual del programa
class Visual {
    public Persistencia Datos;

    //Conecta con la capa de persistencia
    public Visual(Persistencia objDatos) {
        Datos = objDatos;
    }
}

```

```

//Menú principal
public void Menu() {
    int opcion;
    do {
        Console.Clear();
        Console.WriteLine("\n===== Software TV Show 1.4 (Enero de 2024) =====");
        Console.WriteLine("1. CRUD de actores y actrices");
        Console.WriteLine("2. CRUD de series");
        Console.WriteLine("3. Salir");
        Console.Write("Opción? ");
        opcion = Convert.ToInt32(Console.ReadLine());
        switch (opcion) {
            case 1: CRUDactores(); break;
            case 2: CRUDseries(); break;
        }
    } while (opcion != 3);
}

//Menú de actores y actrices
public void CRUDactores() {
    int opcion;
    do {
        Console.Clear();
        Console.WriteLine("\n===== Software TV Show. Actores/Actrices =====");
        for (int cont = 0; cont < Datos.ActorTotal(); cont++) {
            Console.Write("[ " + cont.ToString() + " ] ");
            Console.Write(Datos.ActorNombre(cont));
            Console.WriteLine(" URL: " + Datos.ActorURL(cont).ToString());
        }
        Console.WriteLine("\n1. Adicionar");
        Console.WriteLine("2. Editar");
        Console.WriteLine("3. Borrar");
        Console.WriteLine("4. ¿En cuáles series trabaja?");
        Console.WriteLine("5. Volver a menú principal");
        Console.Write("Opción? ");
        opcion = Convert.ToInt32(Console.ReadLine());
        switch (opcion) {
            case 1: ActorAdiciona(); break;
            case 2: ActorEdita(); break;
            case 3: ActorBorra(); break;
            case 4: ActorTrabaja(); break;
        }
    } while (opcion != 5);
}

//Menú de series de TV
public void CRUDseries() {
    int opcion;
    do {
        Console.Clear();
        Console.WriteLine("\n===== Software TV Show. Series =====");
        for (int cont = 0; cont < Datos.SerieTotal(); cont++) {
            Console.Write("[ " + cont.ToString() + " ] ");
            Console.Write(Datos.SerieNombre(cont));
            Console.WriteLine(" URL: " + Datos.SerieIMDB(cont));
        }
        Console.WriteLine("\n1. Adicionar");
        Console.WriteLine("2. Editar");
        Console.WriteLine("3. Borrar");
        Console.WriteLine("4. Detalles de la serie");
        Console.WriteLine("5. Asociar actor/actriz a serie");
        Console.WriteLine("6. Desasociar actor/actriz a serie");
        Console.WriteLine("7. Volver a menú principal");
        Console.Write("Opción? ");
        opcion = Convert.ToInt32(Console.ReadLine());
        switch (opcion) {
            case 1: SerieAdiciona(); break;
            case 2: SerieEdita(); break;
            case 3: SerieBorra(); break;
            case 4: SerieDetalle(); break;
            case 5: SerieAsocia(); break;
            case 6: SerieDesocia(); break;
        }
    } while (opcion != 7);
}

//Pantalla para adicionar actores
public void ActorAdiciona() {
    Console.WriteLine("\tAdicionar actor o actriz al listado");
}

```

```

Console.WriteLine("¿Nombre? ");
string nombre = Console.ReadLine();
Console.WriteLine("¿URL de IMDB? ");
string URL = Console.ReadLine();
Datos.ActorAdiciona(nombre, URL);
Console.WriteLine("\nActor/actriz adicionado. Presione ENTER para continuar");
Console.ReadKey();
}

//Pantalla para editar actores
public void ActorEdita() {
    Console.WriteLine("\tEditar actor o actriz");
    Console.Write("¿Cuál? Escriba el número que está entre [ ]: ");
    int codigo = Convert.ToInt32(Console.ReadLine());
    Console.WriteLine("¿Nombre? ");
    string nombre = Console.ReadLine();
    Console.WriteLine("¿URL de IMDB? ");
    string URL = Console.ReadLine();
    Datos.ActorEdita(codigo, nombre, URL);
    Console.WriteLine("\nActor/actriz editado. Presione ENTER para continuar");
    Console.ReadKey();
}

//Pantalla para borrar actores
public void ActorBorra() {
    Console.WriteLine("\tBorrar actor o actriz");
    Console.Write("¿Cuál? Escriba el número que está entre [ ]: ");
    int codigo = Convert.ToInt32(Console.ReadLine());
    Datos.ActorBorra(codigo);
    Console.WriteLine("\nActor/actriz borrado. Presione ENTER para continuar");
    Console.ReadKey();
}

//Pantalla para mostrar en que series trabaja el actor
public void ActorTrabaja() {
    List<string> ListaSeries;
    Console.WriteLine("\tListar las series donde trabaja el actor/actriz");
    Console.Write("¿Cuál? Escriba el número que está entre [ ]: ");
    int codigo = Convert.ToInt32(Console.ReadLine());
    ListaSeries = Datos.ActorTrabaja(codigo);
    for (int cont = 0; cont < ListaSeries.Count; cont++) Console.WriteLine(ListaSeries[cont]);
    Console.WriteLine("\nPresione ENTER para continuar");
    Console.ReadKey();
}

//Pantalla para adicionar series
public void SerieAdiciona() {
    Console.WriteLine("\tAdicionar serie al listado");
    Console.Write("¿Nombre? ");
    string nombre = Console.ReadLine();
    Console.WriteLine("¿URL en IMDB? ");
    string url = Console.ReadLine();
    Datos.SerieAdiciona(nombre, url);
    Console.WriteLine("\nSerie adicionada. Presione ENTER para continuar");
    Console.ReadKey();
}

//Pantalla para editar series
public void SerieEdita() {
    Console.WriteLine("\tEditar serie");
    Console.Write("¿Cuál? Escriba el número que está entre [ ]: ");
    int codigo = Convert.ToInt32(Console.ReadLine());
    Console.WriteLine("¿Nombre? ");
    string nombre = Console.ReadLine();
    Console.WriteLine("¿URL en IMDB? ");
    string url = Console.ReadLine();
    Datos.SerieEdita(codigo, nombre, url);
    Console.WriteLine("\nSerie editada. Presione ENTER para continuar");
    Console.ReadKey();
}

//Pantalla para borrar series
public void SerieBorra() {
    Console.WriteLine("\tBorrar serie");
    Console.Write("¿Cuál? Escriba el número que está entre [ ]: ");
    int codigo = Convert.ToInt32(Console.ReadLine());
    Datos.SerieBorra(codigo);
    Console.WriteLine("\nSerie borrada. Presione ENTER para continuar");
    Console.ReadKey();
}

```

```

//Pantalla para ver el detalle de la serie
public void SerieDetalle() {
    List<string> ListaActores;

    Console.WriteLine("\t === Detalle de una serie ===");
    Console.Write("¿Cuál? Escriba el número que está entre [ ]: ");
    int codigo = Convert.ToInt32(Console.ReadLine());

    Console.WriteLine("Nombre: " + Datos.SerieNombre(codigo));
    Console.WriteLine("URL: " + Datos.SerieIMDB(codigo));
    Console.WriteLine("Actores");
    ListaActores = Datos.SerieActores(codigo);
    for (int cont = 0; cont < ListaActores.Count; cont++) Console.WriteLine("\t" +
ListaActores[cont]);

    Console.WriteLine("\nPresione ENTER para continuar");
    Console.ReadKey();
}

//Asociar actor o actriz a una serie
public void SerieAsocia() {
    Console.WriteLine("\tAsocia un actor o actriz a una serie");
    Console.Write("¿Cuál serie? Escriba el número que está entre [ ]: ");
    int serie = Convert.ToInt32(Console.ReadLine());
    for (int cont = 0; cont < Datos.ActorTotal(); cont++) {
        Console.Write("[ " + cont.ToString() + " ] ");
        Console.Write(Datos.ActorNombre(cont));
        Console.WriteLine(" URL: " + Datos.ActorURL(cont).ToString());
    }
    Console.Write("¿Cuál actor/actriz? Escriba el número que está entre [ ]: ");
    int actor = Convert.ToInt32(Console.ReadLine());
    Datos.SerieAsocia(serie, actor);
    Console.WriteLine("\nActor/actriz asociado a la serie. Presione ENTER para continuar");
    Console.ReadKey();
}

//Pantalla para disociar actor de alguna serie
public void SerieDisocia() {
    List<string> ListaActores;

    Console.WriteLine("\t === Disociar actor de la serie ===");
    Console.Write("¿Cuál serie? Escriba el número que está entre [ ]: ");
    int serie = Convert.ToInt32(Console.ReadLine());

    ListaActores = Datos.SerieActores(serie);
    for (int cont = 0; cont < ListaActores.Count; cont++)
        Console.WriteLine("[ " + cont.ToString() + " ] " + ListaActores[cont]);

    Console.Write("¿Cuál actor/actriz quiere quitar? Escriba el número que está entre [ ]: ");
    int actor = Convert.ToInt32(Console.ReadLine());

    Datos.SerieDisocia(serie, actor);
    Console.WriteLine("\nActor/actriz retirado de la serie. Presione ENTER para continuar");
    Console.ReadKey();
}

class Program {
    static void Main() {
        //Se debe llamar primero la capa de persistencia (carga datos de ejemplo)
        Persistencia objDatos = new Persistencia();

        //Luego se llama la capa visual
        Visual objVisual = new Visual(objDatos);
        objVisual.Menu();
    }
}
}

```

```
C:\Users\engin\source\repos\ + ▾  
===== Software TV Show 1.4 (Enero de 2024) =====  
1. CRUD de actores y actrices  
2. CRUD de series  
3. Salir  
¿Opción? |
```

Ilustración 176: Uso de listas para simular un sistema de información

```
C:\Users\engin\source\repos\ + ▾  
===== Software TV Show. Actores/Actrices =====  
[0] Ana María Orozco URL: https://www.imdb.com/name/nm0650450/  
[1] Laura Londoño URL: https://www.imdb.com/name/nm2256810/  
[2] Carolina Ramírez URL: https://www.imdb.com/name/nm1329835/  
[3] Catherine Siachoque URL: https://www.imdb.com/name/nm0796171/  
[4] Carmenza González URL: https://www.imdb.com/name/nm1863990/  
[5] Andrés Londoño URL: https://www.imdb.com/name/nm2150265/  
  
1. Adicionar  
2. Editar  
3. Borrar  
4. ¿En cuáles series trabaja?  
5. Volver a menú principal  
¿Opción? |
```

Ilustración 177: Uso de listas para simular un sistema de información

```
1. Adicionar  
2. Editar  
3. Borrar  
4. ¿En cuáles series trabaja?  
5. Volver a menú principal  
¿Opción? 1  
    Adicionar actor o actriz al listado  
¿Nombre? Rafael Alberto Moreno Parra  
¿URL de IMDB? https://www.imdb.com/name/nmrfllbretmrnpr  
  
Actor/actriz adicionado. Presione ENTER para continuar
```

Ilustración 178: Uso de listas para simular un sistema de información

```
===== Software TV Show. Actores/Actrices =====
[0] Ana María Orozco URL: https://www.imdb.com/name/nm0650450/
[1] Laura Londoño URL: https://www.imdb.com/name/nm2256810/
[2] Carolina Ramírez URL: https://www.imdb.com/name/nm1329835/
[3] Catherine Siachoque URL: https://www.imdb.com/name/nm0796171/
[4] Carmenza González URL: https://www.imdb.com/name/nm1863990/
[5] Andrés Londoño URL: https://www.imdb.com/name/nm2150265/
[6] Rafael Alberto Moreno Parra URL: https://www.imdb.com/name/nmrflibrtmrnpr
```

1. Adicionar
2. Editar
3. Borrar
4. ¿En cuáles series trabaja?
5. Volver a menú principal

¿Opción? 2

 Editar actor o actriz

¿Cuál? Escriba el número que está entre []: 6

¿Nombre? Rafael Alberto

¿URL de IMDB? https://www.imdb.com/name/nmrflibrtmrnpr2024

Actor/actriz editado. Presione ENTER para continuar

Ilustración 179: Uso de listas para simular un sistema de información

```
[6] Rafael Alberto URL: https://www.imdb.com/name/nmrflibrtmrnpr2024
```

1. Adicionar
2. Editar
3. Borrar
4. ¿En cuáles series trabaja?
5. Volver a menú principal

¿Opción? 3

 Borrar actor o actriz

¿Cuál? Escriba el número que está entre []: 6

Actor/actriz borrado. Presione ENTER para continuar

Ilustración 180: Uso de listas para simular un sistema de información

```
===== Software TV Show. Actores/Actrices =====
```

```
[0] Ana María Orozco URL: https://www.imdb.com/name/nm0650450/
[1] Laura Londoño URL: https://www.imdb.com/name/nm2256810/
[2] Carolina Ramírez URL: https://www.imdb.com/name/nm1329835/
[3] Catherine Siachoque URL: https://www.imdb.com/name/nm0796171/
[4] Carmenza González URL: https://www.imdb.com/name/nm1863990/
[5] Andrés Londoño URL: https://www.imdb.com/name/nm2150265/
```

1. Adicionar
2. Editar
3. Borrar
4. ¿En cuáles series trabaja?
5. Volver a menú principal

¿Opción? 4

 Listar las series donde trabaja el actor/actriz

¿Cuál? Escriba el número que está entre []: 0

Yo soy Betty, la fea

Perdida

Presione ENTER para continuar

Ilustración 181: Uso de listas para simular un sistema de información

```
===== Software TV Show. Series =====
[0] Yo soy Betty, la fea URL: https://www.imdb.com/title/tt0233127/
[1] La reina del flow URL: https://www.imdb.com/title/tt8560918/
[2] Café con Aroma de Mujer URL: https://www.imdb.com/title/tt14471346/
[3] Los Briceño URL: https://www.imdb.com/title/tt10348478/
[4] Distrito Salvaje URL: https://www.imdb.com/title/tt8105958/
[5] Mil Colmillos URL: https://www.imdb.com/title/tt9701670/
[6] Perdida URL: https://www.imdb.com/title/tt10064124/
```

1. Adicionar
 2. Editar
 3. Borrar
 4. Detalles de la serie
 5. Asociar actor/actriz a serie
 6. Disociar actor/actriz a serie
 7. Volver a menú principal
- ¿Opción?

Ilustración 182: Uso de listas para simular un sistema de información

```
1. Adicionar
2. Editar
3. Borrar
4. Detalles de la serie
5. Asociar actor/actriz a serie
6. Disociar actor/actriz a serie
7. Volver a menú principal
¿Opción? 4
    === Detalle de una serie ===
¿Cuál? Escriba el número que está entre [ ]: 6
Nombre: Perdida
URL: https://www.imdb.com/title/tt10064124/
Actores
    Ana María Orozco

Presione ENTER para continuar
```

Ilustración 183: Uso de listas para simular un sistema de información

```
1. Adicionar
2. Editar
3. Borrar
4. Detalles de la serie
5. Asociar actor/actriz a serie
6. Disociar actor/actriz a serie
7. Volver a menú principal
¿Opción? 5
    Asocia un actor o actriz a una serie
¿Cuál serie? Escriba el número que está entre [ ]: 2
[0] Ana María Orozco URL: https://www.imdb.com/name/nm0650450/
[1] Laura Londoño URL: https://www.imdb.com/name/nm2256810/
[2] Carolina Ramírez URL: https://www.imdb.com/name/nm1329835/
[3] Catherine Siachoque URL: https://www.imdb.com/name/nm0796171/
[4] Carmenza González URL: https://www.imdb.com/name/nm1863990/
[5] Andrés Londoño URL: https://www.imdb.com/name/nm2150265/
¿Cuál actor/actriz? Escriba el número que está entre [ ]: 3

Actor/actriz asociado a la serie. Presione ENTER para continuar
```

Ilustración 184: Uso de listas para simular un sistema de información

Dictionary

Uso de llaves

En una estructura diccionario, hay una llave y un valor (entero, cadena, objeto). Se puede llegar a ese valor usando la llave. Con la instrucción: NombreDiccionario[Llave]. Ejemplo:

E/023.cs

```
namespace Ejemplo {
    class Program {
        static void Main() {
            Random Azar = new Random();

            //Se define un diccionario: llave, cadena
            //En este caso la llave es un número entero
            Dictionary<int, string> Animales = new Dictionary<int, string> {
                {11, "Ballena"}, 
                {12, "Tortuga marina"}, 
                {13, "Tiburón"}, 
                {14, "Estrella de mar"}, 
                {15, "Hipocampo"}, 
                {16, "Serpiente marina"}, 
                {17, "Delfín"}, 
                {18, "Pulpo"}, 
                {19, "Caballito de mar"}, 
                {20, "Coral"}, 
                {21, "Pingüinos"}, 
                {22, "Calamar"}, 
                {23, "Gaviota"}, 
                {24, "Foca"}, 
                {25, "Manatíes"}, 
                {26, "Ballena con barba"}, 
                {27, "Peces Guppy"}, 
                {28, "Orca"}, 
                {29, "Medusas"}, 
                {30, "Mejillones"}, 
                {31, "Caracoles"} 
            };

            for (int cont = 1; cont <= 10; cont++) {
                int Llave = Azar.Next(11, Animales.Count + 11);
                Console.WriteLine("Llave: " + Llave + " cadena: " + Animales[Llave]);
            }
        }
    }
}
```

```
Llave: 17 cadena: Delfín
Llave: 15 cadena: Hipocampo
Llave: 19 cadena: Caballito de mar
Llave: 12 cadena: Tortuga marina
Llave: 16 cadena: Serpiente marina
Llave: 11 cadena: Ballena
Llave: 30 cadena: Mejillones
Llave: 28 cadena: Orca
Llave: 24 cadena: Foca
Llave: 15 cadena: Hipocampo
```

Ilustración 185: Dictionary

Llaves tipo string

También se puede usar una llave de tipo cadena. El diccionario tiene instrucciones de adicionar y borrar.

E/024.cs

```
namespace Ejemplo {
    class Program {
        static void Main() {
            //Se define un diccionario: llave, cadena
            //En este caso la llave es una cadena
            Dictionary<string, string> Extensiones = new Dictionary<string, string> {
                {"exe", "Ejecutable"}, 
                {"com", "Ejecutable DOS"}, 
                {"vb", "Visual Basic .NET"}, 
                {"cs", "C#"}, 
                {"js", "JavaScript"}, 
                {"xlsx", "Excel"}, 
                {"docx", "Word"}, 
                {"html", "HTML 5"}
            };

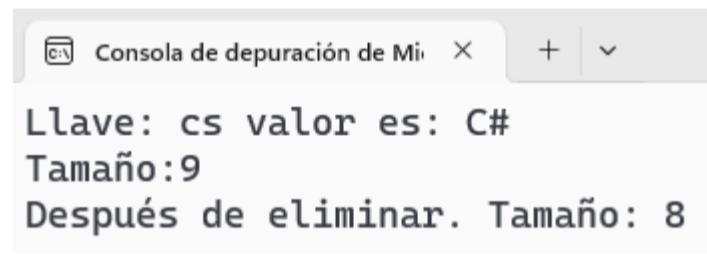
            Extensiones.Add("pptx", "PowerPoint"); //Otra forma de adicionar

            //Trae un elemento dada una llave
            string Llave = "cs";
            Console.WriteLine("Llave: " + Llave + " valor es: " + Extensiones[Llave]);

            //Tamaño del diccionario
            Console.WriteLine("Tamaño:" + Extensiones.Count);

            //Elimina un elemento
            Extensiones.Remove("docx");

            //Tamaño del diccionario
            Console.WriteLine("Después de eliminar. Tamaño: " + Extensiones.Count);
        }
    }
}
```



```
Consola de depuración de Mi... X + ▾
Llave: cs valor es: C#
Tamaño:9
Después de eliminar. Tamaño: 8
```

Ilustración 186: Dictionary

Manejo de objetos en un Dictionary

Un "Dictionary" puede albergar objetos. Además, tiene una serie de métodos (adicionar, consultar, listar llaves, verificar si existe llave) que se ven a continuación:

E/025.cs

```
namespace Ejemplo {
    //Una clase con varios atributos
    class Ejemplo {
        public int Numero { get; set; }
        public double Valor { get; set; }
        public char Caracter { get; set; }
        public string Cadena { get; set; }

        public Ejemplo(int Numero, double Valor, char Caracter, string Cadena) {
            this.Numero = Numero;
            this.Valor = Valor;
            this.Caracter = Caracter;
            this.Cadena = Cadena;
        }
    }

    class Program {
        static void Main() {
            //Se define un diccionario: llave, objeto
            //En este caso la llave es una cadena
            var Objetos = new Dictionary<string, Ejemplo> {
                {"uno", new Ejemplo(1, 0.2, 'r', "Leafar") },
                {"dos", new Ejemplo(8, -7.1, 'a', "Otrebla") },
                {"tres", new Ejemplo(23, -13.6, 'm', "Onerom") },
                {"cuatro", new Ejemplo(49, 16.83, 'p', "Arrap") }
            };

            //Trae los datos del objeto guardado en el diccionario
            string Llave = "tres";
            Console.WriteLine("Llave: " + Llave + " atributo es: " + Objetos[Llave].Cadena);
            Console.WriteLine("Llave: " + Llave + " atributo es: " + Objetos[Llave].Numero);
            Console.WriteLine("Llave: " + Llave + " atributo es: " + Objetos[Llave].Valor);

            //Guarda las llaves en una lista
            Console.WriteLine("\r\nLista de Llaves:");
            var ListaLlaves = new List<string>(Objetos.Keys);
            foreach (string Llaves in ListaLlaves) {
                Console.WriteLine("Llave: " + Llaves);
            }

            //Verifica si existe una llave
            Console.WriteLine("\r\nVerifica si existe una llave:");
            if (Objetos.ContainsKey("cuatro")) {
                Console.WriteLine(Objetos["cuatro"].Cadena);
            }
            else {
                Console.WriteLine("No existe esa llave");
            }
        }
    }
}
```

```
Llave: tres atributo es: Onerom
Llave: tres atributo es: 23
Llave: tres atributo es: -13,6

Lista de Llaves:
Llave: uno
Llave: dos
Llave: tres
Llave: cuatro

Verifica si existe una llave:
Arrap
```

Ilustración 187: Dictionary, manejo de objetos

Una cola se parece a un ArrayList, la diferencia es que NO se puede acceder a los elementos por un índice, se respeta el orden de llegada, primero en entrar es primero en salir.

```

using System.Collections;

namespace Ejemplo {
    class Program {
        static void Main() {
            //Se define una cola: Queue
            Queue Cola = new Queue();

            //Se agregan elementos a la cola
            Cola.Enqueue("aaa");
            Cola.Enqueue("bbb");
            Cola.Enqueue("ccc");
            Cola.Enqueue("ddd");
            Cola.Enqueue("eee");
            Cola.Enqueue("fff");

            //Número de elementos en la cola
            Console.WriteLine("Número de elementos: " + Cola.Count);

            //Imprimir la cola
            Console.WriteLine("\r\nElementos: ");
            foreach(object elemento in Cola) Console.Write(elemento + ", ");

            //Quitar elemento de la cola
            Cola.Dequeue(); //Primero en llegar, primero en salir, luego quitaría a "aaa"
            Console.WriteLine("\r\nAl quitar un elemento de la cola: ");
            foreach (object elemento in Cola) Console.Write(elemento + ", ");

            //Verificar si hay un elemento en la cola
            string Buscar = "ddd";
            if (Cola.Contains(Buscar) == true) {
                Console.WriteLine("\r\nLa cola contiene el elemento: " + Buscar);
            }
            else
                Console.WriteLine("\r\nLa cola NO contiene el elemento: " + Buscar);

            //Obtener el primer elemento de la cola sin borrar ese elemento
            string PrimerElemento = Convert.ToString(Cola.Peek());
            Console.WriteLine("\r\nPrimer elemento de la cola: " + PrimerElemento);

            //Leer y borrar la cola
            Console.WriteLine("\r\nLee y borra la cola: ");
            while (Cola.Count > 0)
                Console.Write(Cola.Dequeue() + "; ");
            Console.WriteLine("\r\nNúmero de elementos: " + Cola.Count);
        }
    }
}

```

The screenshot shows the 'Consola de depuración de Mi...' (Debug Console) window with the following output:

```

Número de elementos: 6

Elementos:
aaa, bbb, ccc, ddd, eee, fff,
Al quitar un elemento de la cola:
bbb, ccc, ddd, eee, fff,

La cola contiene el elemento: ddd

Primer elemento de la cola: bbb

Lee y borra la cola:
bbb; ccc; ddd; eee; fff;
Número de elementos: 0

```

Dato definido en la cola

Los elementos de la cola pueden ser de tipo definido. Se modifica el programa anterior para que trabaje con el tipo string, obteniendo el mismo resultado.

E/027.cs

```
namespace Ejemplo {
    class Program {
        static void Main() {
            //Se define una cola de tipo string: Queue
            Queue<string> Cola = new Queue<string>();

            //Se agregan elementos a la cola
            Cola.Enqueue("aaa");
            Cola.Enqueue("bbb");
            Cola.Enqueue("ccc");
            Cola.Enqueue("ddd");
            Cola.Enqueue("eee");
            Cola.Enqueue("fff");

            //Número de elementos en la cola
            Console.WriteLine("Número de elementos: " + Cola.Count);

            //Imprimir la cola
            Console.WriteLine("\r\nElementos: ");
            foreach(object elemento in Cola) Console.Write(elemento + ", ");

            //Quitar elemento de la cola
            Cola.Dequeue(); //Primero en llegar, primero en salir, luego quitaría a "aaa"
            Console.WriteLine("\r\nAl quitar un elemento de la cola: ");
            foreach (object elemento in Cola) Console.Write(elemento + ", ");

            //Verificar si hay un elemento en la cola
            string Buscar = "ddd";
            if (Cola.Contains(Buscar) == true) {
                Console.WriteLine("\r\nLa cola contiene el elemento: " + Buscar);
            }
            else
                Console.WriteLine("\r\nLa cola NO contiene el elemento: " + Buscar);

            //Obtener el primer elemento de la cola sin borrar ese elemento
            string PrimerElemento = Cola.Peek();
            Console.WriteLine("\r\nPrimer elemento de la cola: " + PrimerElemento);

            //Leer y borrar la cola
            Console.WriteLine("\r\nLee y borra la cola: ");
            while (Cola.Count > 0)
                Console.Write(Cola.Dequeue() + "; ");
            Console.WriteLine("\r\nNúmero de elementos: " + Cola.Count);
        }
    }
}
```

```
Número de elementos: 6

Elementos:
aaa, bbb, ccc, ddd, eee, fff,
Al quitar un elemento de la cola:
bbb, ccc, ddd, eee, fff,

La cola contiene el elemento: ddd

Primer elemento de la cola: bbb

Lee y borra la cola:
bbb; ccc; ddd; eee; fff;
Número de elementos: 0
```

Ilustración 189: Dato definido en la cola

Una cola puede tener objetos personalizados.

```

namespace Ejemplo {
    //Una clase con varios atributos
    class Ejemplo {
        public int Numero { get; set; }
        public double Valor { get; set; }
        public char Caracter { get; set; }
        public string Cadena { get; set; }

        public Ejemplo(int Numero, double Valor, char Caracter, string Cadena) {
            this.Numero = Numero;
            this.Valor = Valor;
            this.Caracter = Caracter;
            this.Cadena = Cadena;
        }
    }

    class Program {
        static void Main() {
            //Se define una cola de tipo objeto personalizado
            Queue<Ejemplo> Cola = new Queue<Ejemplo>();

            //Se agregan elementos a la cola
            Cola.Enqueue(new Ejemplo(1, 0.2, 'r', "Leafar"));
            Cola.Enqueue(new Ejemplo(8, -7.1, 'a', "Otrebla"));
            Cola.Enqueue(new Ejemplo(23, -13.6, 'm', "Onerom"));
            Cola.Enqueue(new Ejemplo(49, 16.83, 'p', "Arrap"));

            //Número de elementos en la cola
            Console.WriteLine("Número de elementos: " + Cola.Count);

            //Imprimir la cola
            Console.WriteLine("\r\nElementos: ");
            foreach (Ejemplo elemento in Cola) Console.Write(elemento.Cadena + ", ");

            //Quitar elemento de la cola
            Cola.Dequeue(); //Primero en llegar, primero en salir, luego quitaría a "aaa"
            Console.WriteLine("\r\nAl quitar un elemento de la cola: ");
            foreach (Ejemplo elemento in Cola) Console.Write(elemento.Cadena + ", ");

            //Obtener el primer elemento de la cola sin borrar ese elemento
            Ejemplo PrimerElemento = Cola.Peek();
            Console.WriteLine("\r\n\r\nPrimer elemento de la cola: " + PrimerElemento.Cadena);

            //Leer y borrar la cola
            Console.WriteLine("\r\nLee y borra la cola: ");
            while (Cola.Count > 0)
                Console.Write(Cola.Dequeue().Cadena + "; ");
            Console.WriteLine("\r\nNúmero de elementos: " + Cola.Count);

            //Agrega elementos a la cola y luego la borra
            Cola.Enqueue(new Ejemplo(7, 6.5, 'z', "qwerty"));
            Cola.Enqueue(new Ejemplo(4, -3.2, 'y', "asdfg"));
            Console.WriteLine("\r\nNúmero de elementos: " + Cola.Count);
            Cola.Clear();
            Console.WriteLine("Después de borrar. Número de elementos: " + Cola.Count);
        }
    }
}

```

```
Consola de depuración de Mi + X
Número de elementos: 4

Elementos:
Leafar, Otrebla, Onerom, Arrap,
Al quitar un elemento de la cola:
Otrebla, Onerom, Arrap,

Primer elemento de la cola: Otrebla

Lee y borra la cola:
Otrebla; Onerom; Arrap;
Número de elementos: 0

Número de elementos: 2
Después de borrar. Número de elementos: 0
```

Ilustración 190: Objetos en la cola

Stack (Pila)

La pila es una estructura que análogo a una pila de platos, cuando se adicionan elementos, estos van quedando encima, por lo que el último en entrar es el primero en salir. Es muy similar a la cola, sólo cambian algunos métodos como el Push (poner) y Pop (retirar).

E/029.cs

```
using System.Collections;

namespace Ejemplo {
    class Program {
        static void Main() {
            //Se define una pila: Queue
            Stack Pila = new Stack();

            //Se agregan elementos a la pila
            Pila.Push("aaa");
            Pila.Push("bbb");
            Pila.Push("ccc");
            Pila.Push("ddd");
            Pila.Push("eee");
            Pila.Push("fff");

            //Número de elementos en la pila
            Console.WriteLine("Número de elementos: " + Pila.Count);

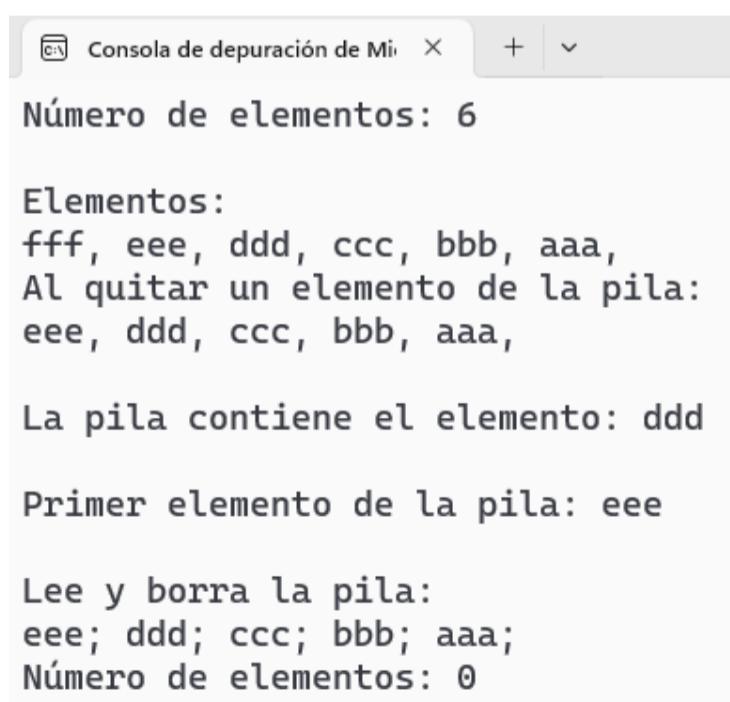
            //Imprimir la pila
            Console.WriteLine("\r\nElementos: ");
            foreach (object elemento in Pila) Console.Write(elemento + ", ");

            //Quitar elemento de la pila
            Pila.Pop(); //Último en llegar, primero en salir, luego quitaría a "fff"
            Console.WriteLine("\r\nAl quitar un elemento de la pila: ");
            foreach (object elemento in Pila) Console.Write(elemento + ", ");

            //Verificar si hay un elemento en la pila
            string Buscar = "ddd";
            if (Pila.Contains(Buscar) == true) {
                Console.WriteLine("\r\n\r\nLa pila contiene el elemento: " + Buscar);
            }
            else
                Console.WriteLine("\r\n\r\nLa pila NO contiene el elemento: " + Buscar);

            //Obtener el primer elemento de la pila sin borrar ese elemento
            string PrimerElemento = Convert.ToString(Pila.Peek());
            Console.WriteLine("\r\nPrimer elemento de la pila: " + PrimerElemento);

            //Leer y borrar la pila
            Console.WriteLine("\r\nLee y borra la pila: ");
            while (Pila.Count > 0)
                Console.Write(Pila.Pop() + "; ");
            Console.WriteLine("\r\nNúmero de elementos: " + Pila.Count);
        }
    }
}
```



```
Consola de depuración de Mi... X + ▾
Número de elementos: 6

Elementos:
fff, eee, ddd, ccc, bbb, aaa,
Al quitar un elemento de la pila:
eee, ddd, ccc, bbb, aaa,

La pila contiene el elemento: ddd

Primer elemento de la pila: eee

Lee y borra la pila:
eee; ddd; ccc; bbb; aaa;
Número de elementos: 0
```

Ilustración 191: Stack

Dato definido en la pila

Los elementos de la pila pueden ser de tipo definido. Se modifica el programa anterior para que trabaje con el tipo string, obteniendo el mismo resultado.

E/030.cs

```
namespace Ejemplo {
    class Program {
        static void Main() {
            //Se define una pila: Queue
            Stack<string> Pila = new Stack<string>();

            //Se agregan elementos a la pila
            Pila.Push("aaa");
            Pila.Push("bbb");
            Pila.Push("ccc");
            Pila.Push("ddd");
            Pila.Push("eee");
            Pila.Push("fff");

            //Número de elementos en la pila
            Console.WriteLine("Número de elementos: " + Pila.Count);

            //Imprimir la pila
            Console.WriteLine("\r\nElementos: ");
            foreach (object elemento in Pila) Console.Write(elemento + ", ");

            //Quitar elemento de la pila
            Pila.Pop(); //Último en llegar, primero en salir, luego quitaría a "fff"
            Console.WriteLine("\r\nAl quitar un elemento de la pila: ");
            foreach (string elemento in Pila) Console.Write(elemento + ", ");

            //Verificar si hay un elemento en la pila
            string Buscar = "ddd";
            if (Pila.Contains(Buscar) == true) {
                Console.WriteLine("\r\nLa pila contiene el elemento: " + Buscar);
            }
            else
                Console.WriteLine("\r\nLa pila NO contiene el elemento: " + Buscar);

            //Obtener el primer elemento de la pila sin borrar ese elemento
            string PrimerElemento = Pila.Peek();
            Console.WriteLine("\r\nPrimer elemento de la pila: " + PrimerElemento);

            //Leer y borrar la pila
            Console.WriteLine("\r\nLee y borra la pila: ");
            while (Pila.Count > 0)
                Console.Write(Pila.Pop() + "; ");
            Console.WriteLine("\r\nNúmero de elementos: " + Pila.Count);
        }
    }
}
```

```
Consola de depuración de Mi... X + ▾
Número de elementos: 6

Elementos:
fff, eee, ddd, ccc, bbb, aaa,
Al quitar un elemento de la pila:
eee, ddd, ccc, bbb, aaa,

La pila contiene el elemento: ddd

Primer elemento de la pila: eee

Lee y borra la pila:
eee; ddd; ccc; bbb; aaa;
Número de elementos: 0
```

Ilustración 192: Dato definido en la pila

```
namespace Ejemplo {
    //Una clase con varios atributos
    class Ejemplo {
        public int Numero { get; set; }
        public double Valor { get; set; }
        public char Caracter { get; set; }
        public string Cadena { get; set; }

        public Ejemplo(int Numero, double Valor, char Caracter, string Cadena) {
            this.Numero = Numero;
            this.Valor = Valor;
            this.Caracter = Caracter;
            this.Cadena = Cadena;
        }
    }

    class Program {
        static void Main() {
            //Se define una pila de tipo objeto personalizado
            Stack<Ejemplo> Pila = new Stack<Ejemplo>();

            //Se agregan elementos a la pila
            Pila.Push(new Ejemplo(1, 0.2, 'r', "Leafar"));
            Pila.Push(new Ejemplo(8, -7.1, 'a', "Otrebla"));
            Pila.Push(new Ejemplo(23, -13.6, 'm', "Onerom"));
            Pila.Push(new Ejemplo(49, 16.83, 'p', "Arrap"));

            //Número de elementos en la pila
            Console.WriteLine("Número de elementos: " + Pila.Count);

            //Imprimir la pila
            Console.WriteLine("\r\nElementos: ");
            foreach (Ejemplo elemento in Pila) Console.Write(elemento.Cadena + ", ");

            //Quitar elemento de la pila
            Pila.Pop(); //Último en llegar, primero en salir
            Console.WriteLine("\r\nAl quitar un elemento de la pila: ");
            foreach (Ejemplo elemento in Pila) Console.Write(elemento.Cadena + ", ");

            //Obtener el primer elemento de la pila sin borrar ese elemento
            Ejemplo PrimerElemento = Pila.Peek();
            Console.WriteLine("\r\n\r\nElemento más arriba de la pila: " + PrimerElemento.Cadena);

            //Leer y borrar la pila
            Console.WriteLine("\r\nLee y borra la pila: ");
            while (Pila.Count > 0)
                Console.Write(Pila.Pop().Cadena + "; ");
            Console.WriteLine("\r\nNúmero de elementos: " + Pila.Count);

            //Agrega elementos a la pila y luego la borra
            Pila.Push(new Ejemplo(7, 6.5, 'z', "qwerty"));
            Pila.Push(new Ejemplo(4, -3.2, 'y', "asdfg"));
            Console.WriteLine("\r\nNúmero de elementos: " + Pila.Count);
            Pila.Clear();
            Console.WriteLine("Después de borrar. Número de elementos: " + Pila.Count);
        }
    }
}
```

```
Consola de depuración de Mi + ▾
Número de elementos: 4

Elementos:
Arrap, Onerom, Otrebla, Leafar,
Al quitar un elemento de la pila:
Onerom, Otrebla, Leafar,

Elemento más arriba de la pila: Onerom

Lee y borra la pila:
Onerom; Otrebla; Leafar;
Número de elementos: 0

Número de elementos: 2
Después de borrar. Número de elementos: 0
```

Ilustración 193: Objetos en la pila

Hashtable

Hashtable funciona similar a Dictionary. Estas son sus diferencias:

Hashtable	Dictionary
Es seguro ser accedido por múltiples hilos ("thread safe").	Sólo miembros públicos estáticos son seguros para ser accedidos por hilos.
Retorna "null" si se intenta acceder a un dato por una llave inexistente.	Genera un error si intenta acceder por una llave inexistente. Requiere usar try catch.
La recuperación de datos es más lenta.	La recuperación de datos es más rápida.
No requiere definir el tipo de dato de la llave y el valor.	Requiere definir el tipo de datos de la llave y el valor.

E/032.cs

```
using System.Collections;

namespace Ejemplo {
    class Program {
        static void Main() {
            Random Azar = new Random();

            //Se define un Hashtable
            //En este caso la llave es un número entero
            Hashtable Animales = new Hashtable();
            Animales.Add(11, "Ballena");
            Animales.Add(12, "Tortuga marina");
            Animales.Add(13, "Tiburón");
            Animales.Add(14, "Estrella de mar");
            Animales.Add(15, "Hipocampo");
            Animales.Add(16, "Serpiente marina");
            Animales.Add(17, "Delfín");
            Animales.Add(18, "Pulpo");
            Animales.Add(19, "Caballito de mar");
            Animales.Add(20, "Coral");
            Animales.Add(21, "Pingüinos");
            Animales.Add(22, "Calamar");
            Animales.Add(23, "Gaviota");
            Animales.Add(24, "Foca");
            Animales.Add(25, "Manatíes");
            Animales.Add(26, "Ballena con barba");
            Animales.Add(27, "Peces Guppy");
            Animales.Add(28, "Orca");
            Animales.Add(29, "Medusas");
            Animales.Add(30, "Mejillones");
            Animales.Add(31, "Caracoles");

            for (int cont = 1; cont <= 10; cont++) {
                //Busque al azar un número entre mínimo y máximo valor de llave
                //Hay que sumarle +1 al máximo valor de llave para que quede dentro del rango de los
                //números aleatorios
                int Llave = Azar.Next(11, 31+1);

                //Muestre el registro según la llave
                Console.WriteLine("Llave: " + Llave + " cadena: " + Animales[Llave]);
            }
        }
    }
}
```

```
Llave: 18 cadena: Pulpo
Llave: 28 cadena: Orca
Llave: 30 cadena: Mejillones
Llave: 18 cadena: Pulpo
Llave: 29 cadena: Medusas
Llave: 21 cadena: Pingüinos
Llave: 26 cadena: Ballena con barba
Llave: 24 cadena: Foca
Llave: 18 cadena: Pulpo
Llave: 31 cadena: Caracoles
```

Ilustración 194: Hashtable

Manejo de objetos en un Hashtable

Cabe recordar que hay que hacer la conversión para acceder a los atributos del objeto almacenado así:

(objeto as clase).atributo

E/033.cs

```
using System.Collections;

namespace Ejemplo {
    class MiClase {
        public int Numero { get; set; }
        public double Valor { get; set; }
        public char Caracter { get; set; }
        public string Cadena { get; set; }

        public MiClase(int Numero, double Valor, char Caracter, string Cadena) {
            this.Numero = Numero;
            this.Valor = Valor;
            this.Caracter = Caracter;
            this.Cadena = Cadena;
        }
    }

    class Program {
        static void Main() {
            //Se define un Hashtable
            Hashtable Objetos = new Hashtable();

            Objetos.Add("uno", new MiClase(1, 0.2, 'r', "Leafar"));
            Objetos.Add("dos", new MiClase(8, -7.1, 'a', "Otrebla"));
            Objetos.Add("tres", new MiClase(23, -13.6, 'm', "Onerom"));
            Objetos.Add("cuatro", new MiClase(49, 16.83, 'p', "Arrap"));

            //Traq los datos del objeto guardado en el diccionario
            string Llave = "tres";
            Console.WriteLine("Llave: " + Llave + " atributo es: " + (Objetos[Llave] as MiClase).Cadena);
            Console.WriteLine("Llave: " + Llave + " atributo es: " + (Objetos[Llave] as MiClase).Numero);
            Console.WriteLine("Llave: " + Llave + " atributo es: " + (Objetos[Llave] as MiClase).Valor);

            //Guarda las llaves en una variable de colección
            Console.WriteLine("\r\nLista de Llaves:");
            var ListaLlaves = Objetos.Keys;
            foreach (string Llaves in ListaLlaves) {
                Console.WriteLine("Llave: " + Llaves);
            }

            //Verifica si existe una llave
            Console.WriteLine("\r\nVerifica si existe una llave:");
            if (Objetos.ContainsKey("cuatro")) {
                Console.WriteLine((Objetos["cuatro"] as MiClase).Cadena);
            }
            else {
                Console.WriteLine("No existe esa llave");
            }
        }
    }
}
```

```
Consola de depuración de Mi... X + ▾  
Llave: tres atributo es: Onerom  
Llave: tres atributo es: 23  
Llave: tres atributo es: -13,6  
  
Lista de Llaves:  
Llave: cuatro  
Llave: uno  
Llave: dos  
Llave: tres  
  
Verifica si existe una llave:  
Arrap
```

Ilustración 195: Manejo de objetos en un Hashtable

SortedList

SortedList es muy similar a Dictionary, en este caso la lista es ordenada automáticamente por las llaves. Eso es visible al imprimirla.

E/034.cs

```
namespace Ejemplo {
    class Program {
        static void Main() {
            //Se define una lista ordenada: llave, cadena
            //En este caso la llave es una cadena
            SortedList<string, string> Extensiones = new SortedList<string, string> {
                {"exe", "Ejecutable"},
                {"com", "Ejecutable DOS"},
                {"vb", "Visual Basic .NET"},
                {"cs", "C#"},
                {"js", "JavaScript"},
                {"xlsx", "Excel"},
                {"docx", "Word"},
                {"html", "HTML 5"}
            };

            Extensiones.Add("pptx", "PowerPoint"); //Otra forma de adicionar

            //Imprime la lista ordenada
            foreach (object elemento in Extensiones) Console.WriteLine(elemento);

            //Imprime llave y valor
            var ListaLlaves = Extensiones.Keys;
            Console.WriteLine("\r\nImprime llave y valor en separado");
            foreach (string Llave in ListaLlaves) {
                Console.WriteLine("Llave: " + Llave + " Valor: " + Extensiones[Llave]);
            }
        }
    }
}
```

The screenshot shows the 'Consola de depuración de Mi...' (Debug Console) window. It displays two sections of output. The first section lists the key-value pairs from the SortedList, each enclosed in square brackets. The second section shows the output of the 'foreach' loop that prints each key followed by its corresponding value separated by a colon.

```
[com, Ejecutable DOS]
[cs, C#]
[docx, Word]
[exe, Ejecutable]
[html, HTML 5]
[js, JavaScript]
[pptx, PowerPoint]
[vb, Visual Basic .NET]
[xlsx, Excel]

Imprime llave y valor en separado
Llave: com Valor: Ejecutable DOS
Llave: cs Valor: C#
Llave: docx Valor: Word
Llave: exe Valor: Ejecutable
Llave: html Valor: HTML 5
Llave: js Valor: JavaScript
Llave: pptx Valor: PowerPoint
Llave: vb Valor: Visual Basic .NET
Llave: xlsx Valor: Excel
```

Ilustración 196: SortedList

Lista enlazada, no se accede directamente por un índice.

```

namespace Ejemplo {
    class Program {
        static void Main() {
            //Se define una lista enlazada
            LinkedList<string> Lenguajes = new LinkedList<string>();

            //Agrega al final
            Lenguajes.AddLast("Visual Basic .NET");
            Lenguajes.AddLast("F#");
            Lenguajes.AddLast("C#");
            Lenguajes.AddLast("TypeScript");

            //Imprime esa lista
            Console.WriteLine("Agregando con AddLast");
            foreach (string elemento in Lenguajes) Console.Write(elemento + "; ");

            //Agrega al inicio
            Lenguajes.AddFirst("C++");
            Lenguajes.AddFirst("C");

            //Imprime esa lista
            Console.WriteLine("\r\n\r\nAgregando con AddFirst");
            foreach (string elemento in Lenguajes) Console.Write(elemento + "; ");

            //Agrega al final
            Lenguajes.AddLast("Python");
            Console.WriteLine("\r\n\r\nAgregando con AddLast");
            foreach (string elemento in Lenguajes) Console.Write(elemento + "; ");

            //Cantidad
            Console.WriteLine("\r\n\r\nCantidad es: " + Lenguajes.Count);

            //Elimina primer elemento
            Lenguajes.RemoveFirst();
            Console.WriteLine("\r\nEliminado el primer elemento");
            foreach (string elemento in Lenguajes) Console.Write(elemento + "; ");

            //Elimina último elemento
            Lenguajes.RemoveLast();
            Console.WriteLine("\r\n\r\nEliminado el último elemento");
            foreach (string elemento in Lenguajes) Console.Write(elemento + "; ");

            //Elimina determinado elemento
            Lenguajes.Remove("F#");
            Console.WriteLine("\r\n\r\nEliminado F#");
            foreach (string elemento in Lenguajes) Console.Write(elemento + "; ");

            //Adiciona antes de C#
            LinkedListNode<string> nodoPosiciona = Lenguajes.Find("C#"); //Busca el nodo que tiene C#
            Lenguajes.AddBefore(nodoPosiciona, "Assembler");
            Console.WriteLine("\r\n\r\nAdiciona antes de C#");
            foreach (string elemento in Lenguajes) Console.Write(elemento + "; ");

            //Adiciona después de C#
            Lenguajes.AddAfter(nodoPosiciona, "Ada");
            Console.WriteLine("\r\n\r\nAdiciona después de C#");
            foreach (string elemento in Lenguajes) Console.Write(elemento + "; ");
        }
    }
}

```

Consola de depuración de Mi
Agregando con AddLast
Visual Basic .NET; F#; C#; TypeScript;

Agregando con AddFirst
C; C++; Visual Basic .NET; F#; C#; TypeScript;

Agregando con AddLast
C; C++; Visual Basic .NET; F#; C#; TypeScript; Python;

Cantidad es: 7

Eliminado el primer elemento
C++; Visual Basic .NET; F#; C#; TypeScript; Python;

Eliminado el último elemento
C++; Visual Basic .NET; F#; C#; TypeScript;

Eliminado F#
C++; Visual Basic .NET; C#; TypeScript;

Adiciona antes de C#
C++; Visual Basic .NET; Assembler; C#; TypeScript;

Adiciona después de C#
C++; Visual Basic .NET; Assembler; C#; Ada; TypeScript;

Ilustración 197: *LinkedList*

```

namespace Ejemplo {

    //Una clase con varios atributos
    class MiClase {
        public int Numero { get; set; }
        public double Valor { get; set; }
        public char Caracter { get; set; }
        public string Cadena { get; set; }

        public MiClase(int Numero, double Valor, char Caracter, string Cadena) {
            this.Numero = Numero;
            this.Valor = Valor;
            this.Caracter = Caracter;
            this.Cadena = Cadena;
        }
    }

    class Program {
        static void Main() {
            //Se define una lista enlazada
            LinkedList<MiClase> Lenguajes = new LinkedList<MiClase>();

            //Agrega al final
            Lenguajes.AddLast(new MiClase(16, 83.29, 'R', "Lenguaje R"));
            Lenguajes.AddLast(new MiClase(29, 89.7, 'A', "ADA"));
            Lenguajes.AddLast(new MiClase(2, 80.19, 'M', "Máquina"));
            Lenguajes.AddLast(new MiClase(95, 7.21, 'P', "PHP"));

            //Imprime esa lista
            Console.WriteLine("Agregando con AddLast");
            foreach (MiClase elemento in Lenguajes) Console.Write(elemento.Cadena + "; ");

            //Agrega al inicio
            Lenguajes.AddFirst(new MiClase(78, 12.32, 'C', "C#"));
            Lenguajes.AddFirst(new MiClase(55, -3.21, 'V', "Visual Basic .NET"));

            //Imprime esa lista
            Console.WriteLine("\r\n\r\nAgregando con AddFirst");
            foreach (MiClase elemento in Lenguajes) Console.Write(elemento.Cadena + "; ");

            //Agrega al final
            Lenguajes.AddLast(new MiClase(16, 83.29, 'T', "TypeScript"));
            Console.WriteLine("\r\n\r\nAgregando con AddLast");
            foreach (MiClase elemento in Lenguajes) Console.Write(elemento.Cadena + "; ");

            //Cantidad
            Console.WriteLine("\r\n\r\nCantidad es: " + Lenguajes.Count);

            //Elimina primer elemento
            Lenguajes.RemoveFirst();
            Console.WriteLine("\r\nEliminado el primer elemento");
            foreach (MiClase elemento in Lenguajes) Console.Write(elemento.Cadena + "; ");

            //Elimina último elemento
            Lenguajes.RemoveLast();
            Console.WriteLine("\r\n\r\nEliminado el último elemento");
            foreach (MiClase elemento in Lenguajes) Console.Write(elemento.Cadena + "; ");
        }
    }
}

```

```
Consola de depuración de Mi... X + ▾
Agregando con AddLast
Lenguaje R; ADA; Máquina; PHP;

Agregando con AddFirst
Visual Basic .NET; C#; Lenguaje R; ADA; Máquina; PHP;

Agregando con AddLast
Visual Basic .NET; C#; Lenguaje R; ADA; Máquina; PHP; TypeScript;

Cantidad es: 7

Eliminado el primer elemento
C#; Lenguaje R; ADA; Máquina; PHP; TypeScript;

Eliminado el último elemento
C#; Lenguaje R; ADA; Máquina; PHP;
```

Ilustración 198: Objetos en *LinkedList*

Parte 6: LINQ

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Fuente de datos, un arreglo unidimensional
            int[] Lista = new int[] { 1, 9, 7, 2, 0, 6, 2, 6, 1, 6, 8, 3, 2, 9, 2, 9 };

            //Consulta: Extrayendo solo los números impares
            //¡OJO! Sólo crea la instrucción de consulta pero todavía no la hace
            IEnumerable<int> Consulta =
                from numero in Lista where (numero % 2) == 1 select numero;

            //Ejecuta la consulta y la imprime
            foreach (int Valor in Consulta)
                Console.WriteLine(Valor.ToString() + ", ");
        }
    }
}
```

```
1, 9, 7, 1, 3, 9, 9,
```

Ilustración 199: LINQ: Filtrar de un arreglo

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Fuente de datos, un arreglo unidimensional
            int[] Lista = new int[] { 1, 9, 7, 2, 0, 6, 2, 6, 1, 6, 8, 3, 2, 9, 2, 9 };

            //Consulta: Extrayendo solo los números pares
            //¡OJO! Aquí si se ejecuta la consulta de una vez
            List<int> Resultados = (from numero in Lista where (numero % 2) == 0 select
numero).ToList();

            //Ejecuta la consulta y la imprime
            for (int contador = 0; contador < Resultados.Count; contador++) {
                Console.Write(Resultados[contador].ToString() + ", ");
            }
        }
    }
}
```

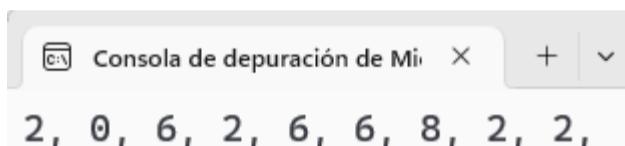
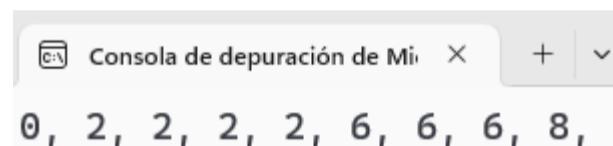


Ilustración 200: LINQ: Filtrar y poner en un List

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Fuente de datos, un arreglo unidimensional
            int[] Lista = new int[] { 1, 9, 7, 2, 0, 6, 2, 6, 1, 6, 8, 3, 2, 9, 2, 9 };

            //Consulta: Extrayendo solo los números pares en orden ascendente
            //¡OJO! Aquí si se ejecuta la consulta de una vez
            List<int> Resultados = (from numero in Lista where (numero % 2) == 0 orderby numero select numero).ToList();

            //Ejecuta la consulta y la imprime
            for (int contador = 0; contador < Resultados.Count; contador++) {
                Console.Write(Resultados[contador].ToString() + ", ");
            }
        }
    }
}
```



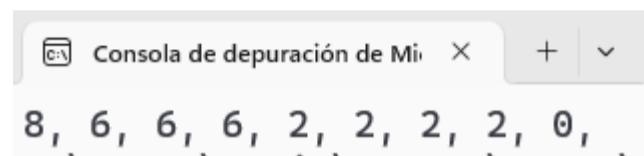
```
0, 2, 2, 2, 2, 6, 6, 6, 8,
```

Ilustración 201: LINQ: Ordenación

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Fuente de datos, un arreglo unidimensional
            int[] Lista = new int[] { 1, 9, 7, 2, 0, 6, 2, 6, 1, 6, 8, 3, 2, 9, 2, 9 };

            //Consulta: Extrayendo solo los números pares en orden descendente
            //¡OJO! Aquí si se ejecuta la consulta de una vez
            List<int> Resultados = (from numero in Lista where (numero % 2) == 0 orderby numero
descending select numero).ToList();

            //Ejecuta la consulta y la imprime
            for (int contador = 0; contador < Resultados.Count; contador++) {
                Console.WriteLine(Resultados[contador].ToString() + ", ");
            }
        }
    }
}
```



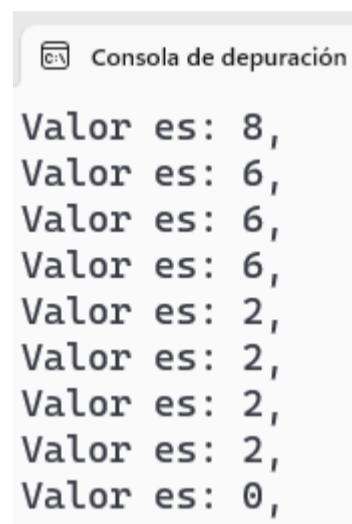
```
8, 6, 6, 6, 2, 2, 2, 2, 0,
```

Ilustración 202: LINQ: Ordenación descendente

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Fuente de datos, un arreglo unidimensional
            int[] Lista = new int[] { 1, 9, 7, 2, 0, 6, 2, 6, 1, 6, 8, 3, 2, 9, 2, 9 };

            //Consulta: Extrayendo solo los números pares en orden descendente
            //¡OJO! Aquí si se ejecuta la consulta de una vez
            List<string> Resultados =
                (from numero in Lista where (numero % 2) == 0
                 orderby numero descending select $"Valor es: {numero}, ").ToList();

            //Ejecuta la consulta y la imprime
            for (int contador = 0; contador < Resultados.Count; contador++) {
                Console.WriteLine(Resultados[contador]);
            }
        }
    }
}
```



The screenshot shows the 'Consola de depuración' (Debug Console) window with the following output:

```
Valor es: 8,
Valor es: 6,
Valor es: 6,
Valor es: 6,
Valor es: 2,
Valor es: 2,
Valor es: 2,
Valor es: 2,
Valor es: 0,
```

Ilustración 203: LINQ: Consulta con salida a texto personalizado

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Fuente de datos, un arreglo unidimensional
            int[] Lista = new int[] { 9, 2, 9, 2, 3, 8, 6, 1 };

            //Consulta: Contando los números pares
            //¡OJO! Aquí si se ejecuta la consulta de una vez
            int TotalRegistros =
                (from numero in Lista
                 where (numero % 2) == 0
                 select numero).Count();

            //Ejecuta la consulta y la imprime
            Console.WriteLine(TotalRegistros);
        }
    }
}
```

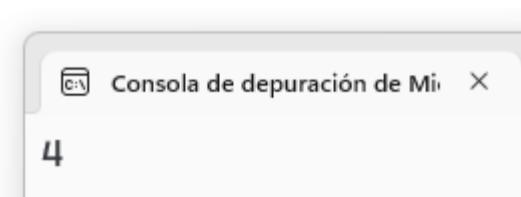


Ilustración 204: LINQ: Contar los registros

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Fuente de datos, un arreglo unidimensional
            int[] Lista = new int[] { 8, -3, 2, 10, -7, 3, 0, 4, 13, -17, 9 };

            //Consulta: Máximo, mínimo y suma
            //¡OJO! Aquí si se ejecuta la consulta de una vez
            int Maximo = (from numero in Lista select numero).Max();
            int Minimo = (from numero in Lista select numero).Min();
            int Suma = (from numero in Lista select numero).Sum();

            //Ejecuta la consulta y la imprime
            Console.WriteLine("Máximo es: " + Maximo.ToString());
            Console.WriteLine("Mínimo es: " + Minimo.ToString());
            Console.WriteLine("Suma es: " + Suma.ToString());

            //Otra forma de hacerlo
            int maximo = Lista.Max();
            int minimo = Lista.Min();
            int suma = Lista.Sum();

            //Imprime
            Console.WriteLine("Máximo: " + maximo.ToString());
            Console.WriteLine("Mínimo: " + minimo.ToString());
            Console.WriteLine("Suma: " + suma.ToString());
        }
    }
}
```

The screenshot shows a command prompt window titled 'Consola de depuración de Mi'. The output displays the results of the LINQ query:

```
Máximo es: 13
Mínimo es: -17
Suma es: 22
Máximo: 13
Mínimo: -17
Suma: 22
```

Ilustración 205: LINQ: Máximo, mínimo y suma

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Fuente de datos, un arreglo unidimensional
            int[] Lista = new int[] { 8, -3, 2, 10, -7, 3, 0, 4, 13, -17, 9 };

            //Consulta: Máximo y mínimo
            //¡OJO! Aquí si se ejecuta la consulta de una vez
            int Maximo = (from numero in Lista where numero % 2 == 0 select numero).Max();
            int Minimo = (from numero in Lista where numero % 2 == 0 select numero).Min();
            int Suma = (from numero in Lista where numero % 2 == 0 select numero).Sum();

            //Ejecuta la consulta y la imprime
            Console.WriteLine("Máximo es: " + Maximo.ToString());
            Console.WriteLine("Mínimo es: " + Minimo.ToString());
            Console.WriteLine("Suma es: " + Suma.ToString());
        }
    }
}
```



```
Máximo es: 10
Mínimo es: 0
Suma es: 24
```

Ilustración 206: LINQ: Máximo, mínimo y suma con condiciones

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Fuente de datos, un arreglo unidimensional
            string[] Lista = new string[] { "búho", "loro", "gaviota", "azulejo", "bichofue", "canario" };
            //Consulta, especies de aves que tengan la letra 'a'
            //¡OJO! Aquí si se ejecuta la consulta de una vez
            int Cuenta = (from aves in Lista where aves.Contains("a")) select aves).Count();
            //Ejecuta la consulta y la imprime
            Console.WriteLine("Aves con la letra a: " + Cuenta.ToString());
        }
    }
}
```



Ilustración 207: LINQ: Consulta con elementos tipo string

```

namespace Ejemplo {
    internal class Mascota {
        public int Codigo;
        public string Nombre;
        public int FechaNace; //Formato: aaaammdd

        public Mascota(int Codigo, string Nombre, int FechaNace) {
            this.Codigo = Codigo;
            this.Nombre = Nombre;
            this.FechaNace = FechaNace;
        }

        public void Imprime() {
            Console.WriteLine("Código: " + Codigo + " Nombre: " + Nombre + " Fecha Nacimiento
(aaaammdd): " + FechaNace);
        }
    }

    internal class Program {
        static void Main() {
            //Fuente de datos, una lista
            List<Mascota> listaMascotas = new List<Mascota>();
            listaMascotas.Add(new Mascota(1, "Suini", 20121012));
            listaMascotas.Add(new Mascota(2, "Sally", 20100701));
            listaMascotas.Add(new Mascota(3, "Capuchina", 20161210));
            listaMascotas.Add(new Mascota(4, "Grisú", 20161120));
            listaMascotas.Add(new Mascota(5, "Arian", 20200102));
            listaMascotas.Add(new Mascota(6, "Milú", 20160706));

            //Extraiga los registros donde el nombre tenga la letra 'a'
            List<Mascota> Resultados = (from animal in listaMascotas where animal.Nombre.Contains("a")
select animal).ToList();

            //Ejecuta la consulta y la imprime
            for (int contador = 0; contador < Resultados.Count; contador++) {
                Resultados[contador].Imprime();
            }
        }
    }
}

```

```

Código: 2 Nombre: Sally Fecha Nacimiento (aaaammdd): 20100701
Código: 3 Nombre: Capuchina Fecha Nacimiento (aaaammdd): 20161210
Código: 5 Nombre: Arian Fecha Nacimiento (aaaammdd): 20200102

```

Ilustración 208: LINQ: Consulta con objetos

```

namespace Ejemplo {
    internal class Mascota {
        public int Codigo { get; set; }
        public string Nombre { get; set; }
        public int FechaNace { get; set; } //Formato: aaaammdd

        public Mascota(int Codigo, string Nombre, int FechaNace) {
            this.Codigo = Codigo;
            this.Nombre = Nombre;
            this.FechaNace = FechaNace;
        }

        public void Imprime() {
            Console.WriteLine("Código: " + Codigo + " Nombre: " + Nombre + " Fecha Nacimiento
(dddmmmmaaaa): " + FechaNace);
        }
    }

    internal class Program {
        static void Main() {
            //Fuente de datos, una lista
            List<Mascota> listaMascotas = new List<Mascota>();
            listaMascotas.Add(new Mascota(1, "Suini", 20121012));
            listaMascotas.Add(new Mascota(2, "Sally", 20100701));
            listaMascotas.Add(new Mascota(3, "Capuchina", 20161210));
            listaMascotas.Add(new Mascota(4, "Grisú", 20161120));
            listaMascotas.Add(new Mascota(5, "Arian", 20200102));
            listaMascotas.Add(new Mascota(6, "Milú", 20160706));

            //Extraiga los registros donde la fecha de nacimiento esté en un rango
            List<Mascota> Resultados = (from animal in listaMascotas where animal.FechaNace > 20150101
&& animal.FechaNace <= 20161231 select animal).ToList();

            //Ejecuta la consulta y la imprime
            for (int contador = 0; contador < Resultados.Count; contador++) {
                Resultados[contador].Imprime();
            }
        }
    }
}

```

Código: 3 Nombre: Capuchina Fecha Nacimiento (aaaammdd): 20161210
Código: 4 Nombre: Grisú Fecha Nacimiento (aaaammdd): 20161120
Código: 6 Nombre: Milú Fecha Nacimiento (aaaammdd): 20160706

Ilustración 209: LINQ: Consulta con objetos y resultado en un List

```

using System.Collections;

namespace Ejemplo {
    internal class Program {
        static void Main() {
            ArrayList ListadoVarios = new ArrayList();

            ListadoVarios.Add(1822);
            ListadoVarios.Add('M');
            ListadoVarios.Add(true);
            ListadoVarios.Add(639.9);
            ListadoVarios.Add("Rafael");

            ListadoVarios.Add(6094);
            ListadoVarios.Add('J');
            ListadoVarios.Add(false);
            ListadoVarios.Add(55.5);
            ListadoVarios.Add("José");

            //Muestra los ítems que son strings
            Console.WriteLine("Strings en el listado:");
            List<string> Cadenas = (from nombre in ListadoVarios.OfType<string>() select
nombre).ToList();
            for (int Contador = 0; Contador < Cadenas.Count; Contador++) {
                Console.WriteLine(Cadenas[Contador]);
            }

            //Muestra los ítems que son booleanos
            Console.WriteLine("\r\nBooleanos en el listado:");
            List<bool> Booleanos = (from valorbool in ListadoVarios.OfType<bool>() select
valorbool).ToList();
            for (int Contador = 0; Contador < Booleanos.Count; Contador++) {
                Console.WriteLine(Booleanos[Contador]);
            }

            //Muestra los ítems que son enteros
            Console.WriteLine("\r\nEnteros en el listado:");
            List<int> Enteros = (from valorentero in ListadoVarios.OfType<int>() select
valorentero).ToList();
            for (int Contador = 0; Contador < Enteros.Count; Contador++) {
                Console.WriteLine(Enteros[Contador]);
            }
        }
    }
}

```

```

Consola de depuración de Mi... + ▾

Strings en el listado:
Rafael
José

Booleanos en el listado:
True
False

Enteros en el listado:
1822
6094

```

Ilustración 210: LINQ: Determinación de tipo de dato

```

namespace Ejemplo {
    internal class Mascota {
        public string Especie { get; set; }
        public string Nombre { get; set; }
        public Mascota(string Especie, string Nombre) {
            this.Especie = Especie;
            this.Nombre = Nombre;
        }

        public void Imprime() {
            Console.WriteLine("Especie: " + Especie + " Nombre: " + Nombre);
        }
    }

    internal class Program {
        static void Main() {
            //Fuente de datos, una lista
            List<Mascota> listaMascotas = new List<Mascota>();
            listaMascotas.Add(new Mascota("gato", "Suini"));
            listaMascotas.Add(new Mascota("gato", "Gris"));
            listaMascotas.Add(new Mascota("gato", "Sally"));
            listaMascotas.Add(new Mascota("gato", "Tinita"));
            listaMascotas.Add(new Mascota("conejo", "Krousky"));
            listaMascotas.Add(new Mascota("gato", "Capuchina"));
            listaMascotas.Add(new Mascota("gato", "Tammy"));
            listaMascotas.Add(new Mascota("gato", "Grisú"));
            listaMascotas.Add(new Mascota("ave", "Lua"));
            listaMascotas.Add(new Mascota("conejo", "Copo"));
            listaMascotas.Add(new Mascota("gato", "Vikingo"));
            listaMascotas.Add(new Mascota("gato", "Arian"));
            listaMascotas.Add(new Mascota("gato", "Milú"));
            listaMascotas.Add(new Mascota("ave", "Azulin"));
            listaMascotas.Add(new Mascota("gato", "Frac"));
            listaMascotas.Add(new Mascota("ave", "Negro"));
            listaMascotas.Add(new Mascota("conejo", "Clopa"));

            //Ordene primero por especie y luego por nombre
            List<Mascota> Resultados = (from animal in listaMascotas orderby animal.Especie,
animal.Nombre select animal).ToList();

            //Ejecuta la consulta y la imprime
            for (int contador = 0; contador < Resultados.Count; contador++) {
                Resultados[contador].Imprime();
            }
        }
    }
}

```

```

Especie: ave Nombre: Azulin
Especie: ave Nombre: Lua
Especie: ave Nombre: Negro
Especie: conejo Nombre: Clopa
Especie: conejo Nombre: Copo
Especie: conejo Nombre: Krousky
Especie: gato Nombre: Arian
Especie: gato Nombre: Capuchina
Especie: gato Nombre: Frac
Especie: gato Nombre: Gris
Especie: gato Nombre: Grisú
Especie: gato Nombre: Milú
Especie: gato Nombre: Sally
Especie: gato Nombre: Suini
Especie: gato Nombre: Tammy
Especie: gato Nombre: Tinita
Especie: gato Nombre: Vikingo

```

Ilustración 211: LINQ: Ordenación por un campo y luego por otro

```
namespace Ejemplo {
    internal class Mascota {
        public string Especie { get; set; }
        public string Nombre { get; set; }
        public Mascota(string Especie, string Nombre) {
            this.Especie = Especie;
            this.Nombre = Nombre;
        }
    }

    internal class Program {
        static void Main() {
            //Fuente de datos, una lista
            List<Mascota> listaMascotas = new List<Mascota>();
            listaMascotas.Add(new Mascota("gato", "Suini"));
            listaMascotas.Add(new Mascota("gato", "Gris"));
            listaMascotas.Add(new Mascota("gato", "Sally"));
            listaMascotas.Add(new Mascota("gato", "Tinita"));
            listaMascotas.Add(new Mascota("conejo", "Krousky"));
            listaMascotas.Add(new Mascota("gato", "Capuchina"));
            listaMascotas.Add(new Mascota("gato", "Tammy"));
            listaMascotas.Add(new Mascota("gato", "Grisú"));
            listaMascotas.Add(new Mascota("ave", "Lua"));
            listaMascotas.Add(new Mascota("conejo", "Copo"));
            listaMascotas.Add(new Mascota("gato", "Vikingo"));
            listaMascotas.Add(new Mascota("gato", "Arian"));
            listaMascotas.Add(new Mascota("gato", "Milú"));
            listaMascotas.Add(new Mascota("ave", "Azulin"));
            listaMascotas.Add(new Mascota("gato", "Frac"));
            listaMascotas.Add(new Mascota("ave", "Negro"));
            listaMascotas.Add(new Mascota("conejo", "Clopa"));

            var ConjuntoGrupos = from animal in listaMascotas group animal by animal.Especie;

            //Itera por grupo
            foreach (var grupo in ConjuntoGrupos) {
                Console.WriteLine("\r\nGrupo: {0}", grupo.Key); //Cada grupo tiene una llave

                foreach (Mascota individuo in grupo) // Cada grupo tiene una colección interna
                    Console.WriteLine("Nombre: {0}", individuo.Nombre);
            }
        }
    }
}
```

The screenshot shows a Windows application window titled 'Consola de depuración de Mi...' (Debug Console of My...). The console displays a list of animal names grouped by category:

- Grupo: gato**
 - Nombre: Suini
 - Nombre: Gris
 - Nombre: Sally
 - Nombre: Tinita
 - Nombre: Capuchina
 - Nombre: Tammy
 - Nombre: Grisú
 - Nombre: Vikingo
 - Nombre: Arian
 - Nombre: Milú
 - Nombre: Frac
- Grupo: conejo**
 - Nombre: Krousky
 - Nombre: Copo
 - Nombre: Clopa
- Grupo: ave**
 - Nombre: Lua
 - Nombre: Azulin
 - Nombre: Negro

Ilustración 212: LINQ: Agrupación por un campo

```
namespace Ejemplo {
    internal class Especie {
        public int Codigo { get; set; }
        public string Nombre { get; set; }

        public Especie(int Codigo, string Nombre) {
            this.Codigo = Codigo;
            this.Nombre = Nombre;
        }
    }

    internal class Mascota {
        public int Especie { get; set; }
        public string Nombre { get; set; }
        public Mascota(int Especie, string Nombre) {
            this.Especie = Especie;
            this.Nombre = Nombre;
        }
    }

    internal class Program {
        static void Main() {
            List<Especie> listaEspecies = new List<Especie>();
            listaEspecies.Add(new Especie(1, "Gato"));
            listaEspecies.Add(new Especie(2, "Conejo"));
            listaEspecies.Add(new Especie(3, "Ave"));

            List<Mascota> listaMascotas = new List<Mascota>();
            listaMascotas.Add(new Mascota(1, "Suini"));
            listaMascotas.Add(new Mascota(1, "Gris"));
            listaMascotas.Add(new Mascota(1, "Sally"));
            listaMascotas.Add(new Mascota(1, "Tinita"));
            listaMascotas.Add(new Mascota(2, "Krousky"));
            listaMascotas.Add(new Mascota(1, "Capuchina"));
            listaMascotas.Add(new Mascota(1, "Tammy"));
            listaMascotas.Add(new Mascota(1, "Grisú"));
            listaMascotas.Add(new Mascota(3, "Lua"));
            listaMascotas.Add(new Mascota(2, "Copo"));
            listaMascotas.Add(new Mascota(1, "Vikingo"));
            listaMascotas.Add(new Mascota(1, "Arian"));
            listaMascotas.Add(new Mascota(1, "Milú"));
            listaMascotas.Add(new Mascota(3, "Azulin"));
            listaMascotas.Add(new Mascota(1, "Frac"));
            listaMascotas.Add(new Mascota(3, "Negro"));
            listaMascotas.Add(new Mascota(2, "Clopa"));

            var Consulta = from mascota in listaMascotas
                           join especie in listaEspecies
                           on mascota.Especie equals especie.Codigo
                           select new {
                               Especie = especie.Nombre,
                               Mascota = mascota.Nombre
                           };

            foreach (var item in Consulta) {
                Console.WriteLine($"La mascota {item.Mascota} es de la especie {item.Especie}");
            }
        }
    }
}
```



```
La mascota Suini es de la especie Gato
La mascota Gris es de la especie Gato
La mascota Sally es de la especie Gato
La mascota Tinita es de la especie Gato
La mascota Krousky es de la especie Conejo
La mascota Capuchina es de la especie Gato
La mascota Tammy es de la especie Gato
La mascota Grisú es de la especie Gato
La mascota Lua es de la especie Ave
La mascota Copo es de la especie Conejo
La mascota Vikingo es de la especie Gato
La mascota Arian es de la especie Gato
La mascota Milú es de la especie Gato
La mascota Azulin es de la especie Ave
La mascota Frac es de la especie Gato
La mascota Negro es de la especie Ave
La mascota Clopa es de la especie Conejo
```

Ilustración 213: LINQ: Hacer un "join" entre listas

```

namespace Ejemplo {
    internal class Especie {
        public int Codigo { get; set; }
        public string Nombre { get; set; }

        public Especie(int Codigo, string Nombre) {
            this.Codigo = Codigo;
            this.Nombre = Nombre;
        }
    }

    internal class Mascota {
        public int Especie { get; set; }
        public string Nombre { get; set; }
        public Mascota(int Especie, string Nombre) {
            this.Especie = Especie;
            this.Nombre = Nombre;
        }
    }

    internal class Program {
        static void Main() {
            List<Especie> listaEspecies = new List<Especie>();
            listaEspecies.Add(new Especie(1, "Gato"));
            listaEspecies.Add(new Especie(2, "Conejo"));
            listaEspecies.Add(new Especie(3, "Ave"));

            List<Mascota> listaMascotas = new List<Mascota>();
            listaMascotas.Add(new Mascota(1, "Suini"));
            listaMascotas.Add(new Mascota(1, "Gris"));
            listaMascotas.Add(new Mascota(1, "Sally"));
            listaMascotas.Add(new Mascota(1, "Tinita"));
            listaMascotas.Add(new Mascota(2, "Krousky"));
            listaMascotas.Add(new Mascota(1, "Capuchina"));
            listaMascotas.Add(new Mascota(1, "Tammy"));
            listaMascotas.Add(new Mascota(1, "Grisú"));
            listaMascotas.Add(new Mascota(3, "Lua"));
            listaMascotas.Add(new Mascota(2, "Copo"));
            listaMascotas.Add(new Mascota(1, "Vikingo"));
            listaMascotas.Add(new Mascota(1, "Arian"));
            listaMascotas.Add(new Mascota(1, "Milú"));
            listaMascotas.Add(new Mascota(3, "Azulin"));
            listaMascotas.Add(new Mascota(1, "Frac"));
            listaMascotas.Add(new Mascota(3, "Negro"));
            listaMascotas.Add(new Mascota(2, "Clopa"));

            var Consulta = from mascota in listaMascotas
                           join especie in listaEspecies
                           on mascota.Especie equals especie.Codigo
                           select new {
                               Mascota = "Nombre Mascota: " + mascota.Nombre,
                               Especie = "Especie: " + especie.Nombre
                           };

            foreach (var item in Consulta) {
                Console.WriteLine($"{item.Mascota} {item.Especie}");
            }
        }
    }
}

```

```
Nombre Mascota: Suini Especie: Gato
Nombre Mascota: Gris Especie: Gato
Nombre Mascota: Sally Especie: Gato
Nombre Mascota: Tinita Especie: Gato
Nombre Mascota: Krousky Especie: Conejo
Nombre Mascota: Capuchina Especie: Gato
Nombre Mascota: Tammy Especie: Gato
Nombre Mascota: Grisú Especie: Gato
Nombre Mascota: Lua Especie: Ave
Nombre Mascota: Copo Especie: Conejo
Nombre Mascota: Vikingo Especie: Gato
Nombre Mascota: Arian Especie: Gato
Nombre Mascota: Milú Especie: Gato
Nombre Mascota: Azulin Especie: Ave
Nombre Mascota: Frac Especie: Gato
Nombre Mascota: Negro Especie: Ave
Nombre Mascota: Clopa Especie: Conejo
```

Ilustración 214: LINQ: Un "join" con resultado personalizado

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            List<string> Animales = new List<string>() { "Gato", "Condor", "Perro", "Conejo", "Guacamaya" };
            List<string> Mamiferos = new List<string>() { "Perro", "Conejo", "Gato" };

            /* Extrae los animales que no están en mamíferos */
            var Resultado = Animales.Except(Mamiferos);

            foreach(string Cadena in Resultado)
                Console.WriteLine(Cadena);
        }
    }
}
```

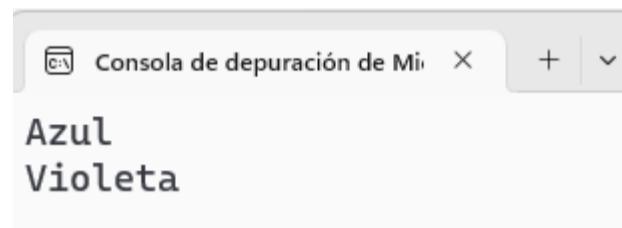
The screenshot shows a Windows Command Prompt window. The title bar says 'Consola de depuración de Mi...'. The window contains the following text:
Condor
Guacamaya

Ilustración 215: LINQ: Extraer los datos de una lista que no están en otra

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            List<string> ColoresA = new List<string>() { "Azul", "Rojo", "Verde", "Violeta" };
            List<string> ColoresB = new List<string>() { "Violeta", "Azul", "Marrón" };

            /* Muestra los colores comunes a ambas listas */
            var Resultado = ColoresA.Intersect(ColoresB);

            foreach(string Cadena in Resultado)
                Console.WriteLine(Cadena);
        }
    }
}
```



```
Azul
Violeta
```

Ilustración 216: LINQ: Intersección de dos listas

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            List<string> ColoresA = new List<string>() { "Azul", "Rojo", "Verde", "Marrón", "Violeta" };
            List<string> ColoresB = new List<string>() { "Violeta", "Azul", "Marrón", "Naranja" };

            /* Une los valores de ambas listas evitando repetir */
            var Resultado = ColoresA.Union(ColoresB);

            foreach(string Cadena in Resultado)
                Console.WriteLine(Cadena);
        }
    }
}
```

```
Azul
Rojo
Verde
Marrón
Violeta
Naranja
```

Ilustración 217: LINQ: Unir dos listas sin repetir elementos

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            List<string> Textos = new List<string>() { "abc", "Opq", "Afv", "Tkl", "qaz", "Akh", "osd",
"uyt", "oxv" };

            //Extrae las palabras que empiezan con "a"
            var PalabraMinusculaA = from palabra in Textos
                where palabra.ToLower().StartsWith("a")
                select palabra.ToLower();

            foreach(string Cadena in PalabraMinusculaA) Console.WriteLine(Cadena);

            //Extrae las palabras que empiezan con "o" usando la instrucción Let
            var PalabraMinusculaB = from palabra in Textos
                let minuscula = palabra.ToLower()
                where minuscula.StartsWith("o")
                select minuscula;

            foreach(string Cadena in PalabraMinusculaB) Console.WriteLine(Cadena);
        }
    }
}
```



```
abc
afv
akh
opq
osd
oxv
```

Ilustración 218: LINQ: Consulta de texto por algún patrón

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Una cadena
            string Cadena = "esta-es-una-prueba-de-ordenamiento-del-interior-de-una-cadena";
            Console.WriteLine("Cadena: " + Cadena);

            /* Se ordena usando LINQ
             * Si desea ordenar los elementos dentro de una secuencia, deberá pasar
             * un método keySelector de identidad que indique que cada elemento de
             * la secuencia es, en sí mismo, una clave. */
            IEnumerable<char> resultado = Cadena.OrderBy(str => str);
            Console.WriteLine("Arreglo con las letras ordenadas");
            foreach (int valor in resultado) {
                Console.Write("[" + (char)valor + "] ");
            }

            // Y convierte ese arreglo en cadena
            string Ordenado = String.Concat(resultado);

            //Imprime
            Console.WriteLine("\r\nOrdenado por letra: " + Ordenado);
        }
    }
}
```

```
Cadena: esta-es-una-prueba-de-ordenamiento-del-interior-de-una-cadena
Arreglo con las letras ordenadas
[-] [-] [-] [-] [-] [-] [-] [a] [a] [a] [a] [a] [a] [a] [b] [c] [d] [d] [d] [d] [d] [e] [e] [e] [e] [e]
[e] [e] [e] [e] [i] [i] [i] [l] [m] [n] [n] [n] [n] [o] [o] [o] [p] [r] [r] [r] [r] [s] [s] [t] [t] [t] [u] [u]
[u]
Ordenado por letra: -----aaaaaaaaabccccdddeeeeeeeeiiilmnnnnnnnooprrrrsssttuuu
```

Ilustración 219: LINQ: Ordenar internamente una cadena

```

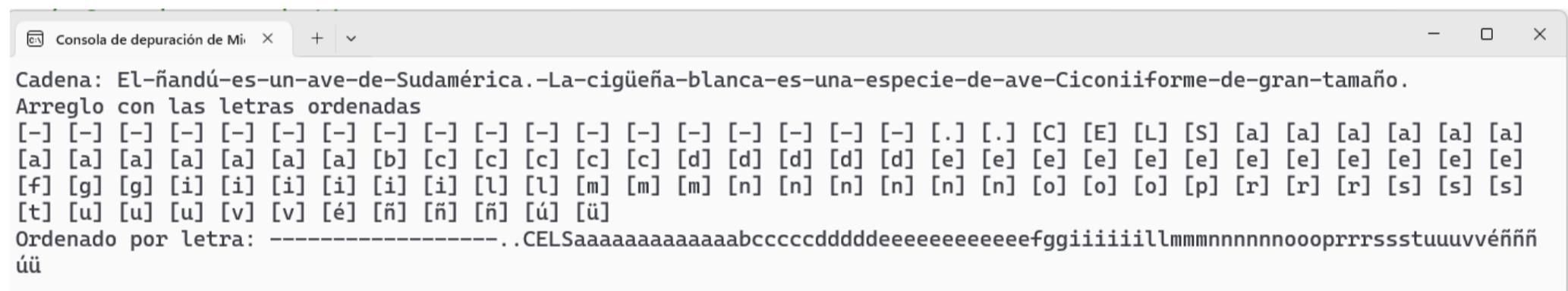
namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Una cadena
            string Cadena = "El-ñandú-es-un-ave-de-Sudamérica.-La-cigüeña-blanca-es-una-especie-de-ave-
Ciconiiforme-de-gran-tamaño.";
            Console.WriteLine("Cadena: " + Cadena);

            /* Se ordena usando Linq
             * Si desea ordenar los elementos dentro de una secuencia, deberá pasar
             * un método keySelector de identidad que indique que cada elemento de
             * la secuencia es, en sí mismo, una clave.*/
            IEnumerable<char> resultado = Cadena.OrderBy(str => str);
            Console.WriteLine("Arreglo con las letras ordenadas");
            foreach (int valor in resultado) {
                Console.Write("[" + (char)valor + "] ");
            }

            // Y convierte ese arreglo en cadena
            string Ordenado = String.Concat(resultado);

            //Imprime
            Console.WriteLine("\r\nOrdenado por letra: " + Ordenado);
        }
    }
}

```



The screenshot shows the 'Consola de depuración' (Debug Console) window in Visual Studio. It displays the output of the program's execution. The first two lines are the original string and the header for the sorted array. The third line shows the individual characters of the sorted array on separate lines, grouped by their ASCII value. The fourth line is the final sorted string. The fifth line is a blank line.

```

Consola de depuración de Mi... X + ▾ - □ ×
Cadena: El-ñandú-es-un-ave-de-Sudamérica.-La-cigüeña-blanca-es-una-especie-de-ave-Ciconiiforme-de-gran-tamaño.
Arreglo con las letras ordenadas
[-] [-] [-] [-] [-] [-] [-] [-] [-] [-] [-] [-] [-] [-] [-] [.] [.] [C] [E] [L] [S] [a] [a] [a] [a] [a]
[a] [a] [a] [a] [a] [a] [b] [c] [c] [c] [c] [d] [d] [d] [d] [d] [d] [e] [e] [e] [e] [e] [e] [e] [e] [e] [e]
[f] [g] [g] [i] [i] [i] [i] [i] [l] [l] [m] [m] [m] [n] [n] [n] [n] [n] [n] [o] [o] [o] [p] [r] [r] [r] [s] [s]
[t] [u] [u] [u] [v] [é] [ñ] [ñ] [ñ] [ú] [ü]
Ordenado por letra: -----
..CELSaaaaaaaaaaaabccccccdddddeeeeeeeefggiiiiillmmmmnnnnnooprrrsssstuuuvvénññ
úü

```

Ilustración 220: LINQ: Ordenar internamente una cadena con diversos caracteres alfanuméricos

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Una cadena
            string[] Cadenas = { "gato", "perro", "avestruz", "toro", "ballena", "kakapo" };

            /* Se ordena usando Linq
             * Ordena por tamaño de las cadenas. */
            IEnumerable<string> resultado = Cadenas.OrderBy(str => str.Length);
            Console.WriteLine("Ordenado por tamaño");
            foreach (string valor in resultado) {
                Console.WriteLine("[" + valor + "]");
            }
        }
    }
}
```

```
Ordenado por tamaño
[gato]
[toro]
[perro]
[kakapo]
[ballena]
[avestruz]
```

Ilustración 221: LINQ: Ordenamiento según tamaño de la palabra

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Una cadena
            string[] Cadenas = { "gato", "perro", "avestruz", "toro", "ballena", "kakapo" };

            /* Se ordena usando Linq
             * Ordena por la segunda letra. */
            IEnumerable<string> resultado = Cadenas.OrderBy(str => str[1]);
            Console.WriteLine("Ordenado por la segunda letra");
            foreach (string valor in resultado) {
                Console.WriteLine("[" + valor + "]");
            }
        }
    }
}
```

The screenshot shows a command prompt window titled 'Consola de depuración de Mi' (Debug Console of My). The output is as follows:

```
Ordenado por la segunda letra
[gato]
[ballena]
[kakapo]
[perro]
[toro]
[avestruz]
```

Ilustración 222: LINQ: Ordenamiento por la segunda letra de cada palabra

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Una cadena
            string[] Cadenas = { "gato", "perro", "avestruz", "toro", "ballena", "kakapo" };

            /* Se ordena usando Linq
             * Invierte el orden */
            IEnumerable<string> resultado = Cadenas.Reverse();
            Console.WriteLine("Invierte el orden");
            foreach (string valor in resultado) {
                Console.WriteLine("[" + valor + "]");
            }
        }
    }
}
```

```
Consola de depuración de Mi... + ▾
Invierte el orden
[kakapo]
[ballena]
[toro]
[avestruz]
[perro]
[gato]
```

Ilustración 223: LINQ: Invertir el ordenamiento

Parte 7. Un evaluador de expresiones algebraicas

El problema

Dada una expresión algebraica almacenada en una cadena "string", esta debe ser interpretada para devolver un valor real. Ejemplo:

Cadena: "3*4+1" → Resultado: 13

Hay que considerar que las expresiones algebraicas pueden tener:

1. Números reales
2. Variables (de la a .. z)
3. Operadores (suma, resta, multiplicación, división, potencia)
4. Uso de paréntesis
5. Uso de funciones (seno, coseno, tangente, valor absoluto, arcoseno, arcocoseno, arcotangente, logaritmo natural, valor techo, exponencial, raíz cuadrada).

El algoritmo

Esta es la versión 4.0 del evaluador de expresiones.

La expresión algebraica está almacenada en una variable de tipo cadena (string). Por ejemplo:

String Cadena = "0.004 - (1.78 / 3.45 + h) * sen(k ^ x)"

La expresión algebraica puede tener números reales, operadores, paréntesis, variables y funciones. Luego hay que interpretarla y evaluarla siguiendo las estrictas reglas matemáticas.

Paso 1: Convertirla a minúsculas y retirar cualquier carácter que no sea de una expresión algebraica

Como la expresión algebraica ha sido digitada por un usuario final entonces se quita cualquier carácter(char) que no sea parte de una expresión algebraica. Los únicos caracteres permitidos son: "abcdefghijklmnoprstuvwxyz0123456789.+*/^()"

Paso 2: Verificando la sintaxis de la expresión algebraica

Luego se verifica si la expresión cumple con las estrictas reglas sintácticas del álgebra. Se hacen 26 validaciones que son:

1. Un número seguido de una letra. Ejemplo: 2q-(*)3
2. Un número seguido de un paréntesis que abre. Ejemplo: 7-2(5-6)
3. Doble punto seguido. Ejemplo: 3..1
4. Punto seguido de operador. Ejemplo: 3.*1
5. Un punto y sigue una letra. Ejemplo: 3+5.w-8
6. Punto seguido de paréntesis que abre. Ejemplo: 2-5.(4+1)*3
7. Punto seguido de paréntesis que cierra. Ejemplo: 2-(5.)*3
8. Un operador seguido de un punto. Ejemplo: 2-(4+.1)-7
9. Dos operadores estén seguidos. Ejemplo: 2++4, 5-*3
10. Un operador seguido de un paréntesis que cierra. Ejemplo: 2-(4+)-7
11. Una letra seguida de número. Ejemplo: 7-2a-6
12. Una letra seguida de punto. Ejemplo: 7-a.-6
13. Un paréntesis que abre seguido de punto. Ejemplo: 7-(.4-6)
14. Un paréntesis que abre seguido de un operador. Ejemplo: 2-(*)3
15. Un paréntesis que abre seguido de un paréntesis que cierra. Ejemplo: 7-()-6
16. Un paréntesis que cierra y sigue un número. Ejemplo: (3-5)7
17. Un paréntesis que cierra y sigue un punto. Ejemplo: (3-5).
18. Un paréntesis que cierra y sigue una letra. Ejemplo: (3-5)t
19. Un paréntesis que cierra y sigue un paréntesis que abre. Ejemplo: (3-5)(4*5)
20. Hay dos o más letras seguidas (obviando las funciones)
21. Los paréntesis están desbalanceados. Ejemplo: 3-(2*4))
22. Doble punto en un número de tipo real. Ejemplo: 7-6.46.1+2
23. Paréntesis que abre no corresponde con el que cierra. Ejemplo: 2+3)-2*(4 ,
24. Inicia con operador. Ejemplo: +3*5
25. Finaliza con operador. Ejemplo: 3*5*
26. Letra seguida de paréntesis que abre (obviando las funciones). Ejemplo: 4*a(6-2)

Paso 3: Transformando la expresión

Se agrega un paréntesis que abre al inicio y luego un paréntesis al final.

De ser: "0.004-(1.78/3.45+h)*sen(k^x)"

Pasa a: "(0.004-(1.78/3.45+h)*sen(k^x))"

Es necesario ese paso porque el evaluador busca los paréntesis para extraer la expresión interna.

Luego convierte las funciones (seno, coseno, tangente) detectadas en alguna letra mayúscula predeterminada.

De ser: "(0.004-(1.78/3.45+h)*sen(k^x))"

Pasa a: "(0.004-(1.78/3.45+h)*A(k^x))"

Esta es la tabla de conversión:

Función	Descripción	Letra con que se reemplaza
Sen	Seno	A
Cos	Coseno	B
Tan	Tangente	C
Abs	Valor absoluto	D
Asn	Arcoseno	E
Acs	Arcocoseno	F
Atn	Arcotangente	G
Log	Logaritmo Natural	H
Cei	Valor techo	I
Exp	Exponencial	J
Sqr	Raíz cuadrada	K

Paso 4: Dividiendo la cadena en partes

Se toma la cadena y se divide en partes diferenciadas: número real, operador, variable, paréntesis que abre, paréntesis que cierra y función. Por ejemplo:

"(0.004-(1.78/3.45+h)*A(k^x))"

Queda en:

(0.004 - (1.78 / 3.45 + h) * A (k ^ x))

Las variables y números constantes quedan en una lista dinámica llamada Valores de tipo double (las variables guardan sus respectivos valores allí). De esa forma cuando el evaluador necesite un valor para consultar o modificarlo, lo busca directamente en esa lista. Este cambio algorítmico da como resultado, que esta versión del evaluador (la 4.0), sea más rápida que la versión anterior del evaluador de expresiones.

Lista Valores

Posición 0 a 25	26	27	28
Variables de la expresión	0.004	1.78	3.45

Por lo que la lista de partes queda ahora así:

([26] - ([27] / [28] + [7]) * A ([10] ^ [23]))

Lo que está entre [] es la posición del valor en la lista de Valores.

En C# esta sería la lista de Valores

```
/* Donde guarda los valores de variables, constantes y piezas */
private List<double> Valores = new List<double>();
```

Las partes se guardan aquí:

```
/* Listado de partes en que se divide la expresión
   Toma una expresión, por ejemplo: 1.68 + sen( 3 / x ) * ( 2.9 ^ 2 - 9 ) y la divide en partes
   así:
   [1.68] [+] [sen()] [3] [/] [x] []] [*] [() [2.9] [^] [2] [-] [9] []]
   Cada parte puede tener un número, un operador, una función, un paréntesis que abre o un
   paréntesis que cierra.
   En esta versión 4.0, las constantes, piezas y variables guardan sus valores en la lista Valores.
   En
   partes, se almacena la posición en Valores */
private List<ParteEvl4> Partes = new List<ParteEvl4>();
```

Y la clase de las partes es así:

```
internal class ParteEvl4 {
    public int Tipo; /* Acumulador, función, paréntesis que abre, paréntesis que cierra, operador,
                      número, variable */
    public int Funcion; /* Código de la función 0:seno, 1:coseno, 2:tangente, 3: valor absoluto, 4:
                        arcoseno, 5: arcocoseno, 6: arcotangente, 7: logaritmo natural, 8: valor tope, 9: exponencial, 10: raíz
                        cuadrada */
    public int Operador; /* + suma - resta * multiplicación / división ^ potencia */
    public int posNumero; /* Posición en lista de valores del número literal, por ejemplo: 3.141592 */
    public int posVariable; /* Posición en lista de valores del valor de la variable algebraica */
    public int posAcumula; /* Posición en lista de valores del valor de la pieza. Por ejemplo:
                           3 + 2 / 5 se convierte así:
                           |3| |+| |2| |/| |5|
                           |3| |+| |A| A es un identificador de acumulador */
}
```

Paso 5: Generando las Piezas desde las Partes

Las Piezas tienen esta estructura: [Pieza] Función Parte1 Operador Parte2

Esta sería la clase que representa las piezas:

```
internal class PiezaEvl4 {
    public int PosResultado; /* Posición donde se almacena el valor que genera la pieza al evaluarse */
    public int Funcion; /* Código de la función 0:seno, 1:coseno, 2:tangente, 3: valor absoluto, 4:
                        arcoseno, 5: arcocoseno, 6: arcotangente, 7: logaritmo natural, 8: valor tope, 9: exponencial, 10: raíz
                        cuadrada, 11: raíz cúbica */
    public int PosValorA; /* Posición donde se almacena la primera parte de la pieza */
    public int Operador; /* 0 suma 1 resta 2 multiplicación 3 división 4 potencia */
    public int PosValorB; /* Posición donde se almacena la segunda parte de la pieza */
}
```

Esas Piezas se construyen desde las Partes. Se sigue el orden de evaluación de los paréntesis y operadores. De:

([26] - ([27] / [28] + [7]) * A ([10] ^ [23]))

Queda así:

Pieza	Función	Parte1	Operador	Parte2
[29]	A	[10]	^	[23]
[30]		[27]	/	[28]
[31]		[30]	+	[7]
[32]		[31]	*	[29]
[33]		[26]	-	[32]

Paso 6: Evaluando a partir de las Piezas

Yendo de pieza en pieza se evalúa toda la expresión.

```

/* Evaluador de expresiones versión 4 (enero de 2024)
 * Autor: Rafael Alberto Moreno Parra
 * Correo: ramsoftware@gmail.com ; enginelife@hotmail.com
 * URL: http://darwin.50webs.com
 * GitHub: https://github.com/ramsoftware
 */

using System.Globalization;
using System.Text;

namespace Ejemplo {

    internal class ParteEvl4 {
        /* Constantes de los diferentes tipos de datos que tendrán las piezas */
        private const int ESFUNCION = 1;
        private const int ESPARABRE = 2;
        private const int ESOPERADOR = 4;
        private const int ESNUMERO = 5;
        private const int ESVARIABLE = 6;

        public int Tipo; /* Acumulador, función, paréntesis que abre, paréntesis que cierra, operador, número, variable */
        public int Funcion; /* Código de la función 0:seno, 1:coseno, 2:tangente, 3: valor absoluto, 4: arcoseno, 5: arcocoseno, 6: arcotangente, 7: logaritmo natural, 8: valor tope, 9: exponencial, 10: raíz cuadrada */
        public int Operador; /* + suma - resta * multiplicación / división ^ potencia */
        public int posNumero; /* Posición en lista de valores del número literal, por ejemplo: 3.141592 */
        public int posVariable; /* Posición en lista de valores del valor de la variable algebraica */
        public int posAcumula; /* Posición en lista de valores del valor de la pieza. Por ejemplo:
            3 + 2 / 5 se convierte así:
            |3| |+| |2| |/| |5|
            |3| |+| |A| A es un identificador de acumulador */

        public ParteEvl4(int TipoParte, int Valor) {
            this.Tipo = TipoParte;
            switch (TipoParte) {
                case ESFUNCION: this.Funcion = Valor; break;
                case ESNUMERO: this.posNumero = Valor; break;
                case ESVARIABLE: this.posVariable = Valor; break;
                case ESPARABRE: this.Funcion = -1; break;
            }
        }

        public ParteEvl4(char Operador) {
            this.Tipo = ESOPERADOR;
            switch (Operador) {
                case '+': this.Operador = 0; break;
                case '-': this.Operador = 1; break;
                case '*': this.Operador = 2; break;
                case '/': this.Operador = 3; break;
                case '^': this.Operador = 4; break;
            }
        }
    }

    internal class PiezaEvl4 {
        public int PosResultado; /* Posición donde se almacena el valor que genera la pieza al evaluarse */
        public int Funcion; /* Código de la función 0:seno, 1:coseno, 2:tangente, 3: valor absoluto, 4: arcoseno, 5: arcocoseno, 6: arcotangente, 7: logaritmo natural, 8: valor tope, 9: exponencial, 10: raíz cuadrada, 11: raíz cúbica */
        public int PosValorA; /* Posición donde se almacena la primera parte de la pieza */
        public int Operador; /* 0 suma 1 resta 2 multiplicación 3 división 4 potencia */
        public int PosValorB; /* Posición donde se almacena la segunda parte de la pieza */
    }
}

internal class Evaluador4 {
    /* Constantes de los diferentes tipos de datos que tendrán las piezas */
    private const int ESFUNCION = 1;
    private const int ESPARABRE = 2;
    private const int ESPARCIERRA = 3;
    private const int ESOPERADOR = 4;
    private const int ESNUMERO = 5;
    private const int ESVARIABLE = 6;
    private const int ESACUMULA = 7;
}

```

```

/* Expresión algebraica convertida de la original dada por el usuario */
private StringBuilder ParaSerAnalizada = new StringBuilder();

/* Donde guarda los valores de variables, constantes y piezas */
private List<double> Valores = new List<double>();

/* Listado de partes en que se divide la expresión
Toma una expresión, por ejemplo: 1.68 + sen( 3 / x ) * ( 2.9 ^ 2 - 9 ) y la divide en partes
así:
[1.68] [+] [sen() [3] [/] [x] []] [*] [() [2.9] [^] [2] [-] [9] []]
Cada parte puede tener un número, un operador, una función, un paréntesis que abre o un
paréntesis que cierra.
En esta versión 4.0, las constantes, piezas y variables guardan sus valores en la lista
Valores. En
    partes, se almacena la posición en Valores */
private List<ParteEvl4> Partes = new List<ParteEvl4>();

/* Listado de piezas que ejecutan
Toma las partes y las divide en piezas con la siguiente estructura:
acumula = función numero/variable/acumula operador numero/variable/acumula
Siguiendo el ejemplo anterior sería:
A = 2.9 ^ 2
B = A - 9
C = seno ( 3 / x )
D = C * B
E = 1.68 + D

Esas piezas se evalúan de arriba a abajo y así se interpreta la ecuación */
private List<PiezaEvl4> Piezas = new List<PiezaEvl4>();

/* Analiza la expresión */
public int Analizar(string ExpresionOriginal) {
    Partes.Clear();
    Piezas.Clear();
    Valores.Clear();

    /* Hace espacio para las 26 variables que puede manejar el evaluador */
    for (int Variables = 1; Variables <= 26; Variables++) Valores.Add(0);

    int Sintaxis = ChequeaSintaxis(ExpresionOriginal);
    if (Sintaxis == 0) {
        CrearPartes();
        CrearPiezas();
    }
    return Sintaxis;
}

/* Retorna mensaje de error sintáctico */
public string MensajeError(int CodigoError) {
    string Mensaje = "";
    switch (CodigoError) {
        case 1: Mensaje = "1. Un número seguido de una letra. Ejemplo: 2q-(*3)"; break;
        case 2: Mensaje = "2. Un número seguido de un paréntesis que abre. Ejemplo: 7-2(5-6)";
break;
        case 3: Mensaje = "3. Doble punto seguido. Ejemplo: 3..1"; break;
        case 4: Mensaje = "4. Punto seguido de operador. Ejemplo: 3.*1"; break;
        case 5: Mensaje = "5. Un punto y sigue una letra. Ejemplo: 3+5.w-8"; break;
        case 6: Mensaje = "6. Punto seguido de paréntesis que abre. Ejemplo: 2-5.(4+1)*3";
break;
        case 7: Mensaje = "7. Punto seguido de paréntesis que cierra. Ejemplo: 2-(5.)*3";
break;
        case 8: Mensaje = "8. Un operador seguido de un punto. Ejemplo: 2-(4+.1)-7"; break;
        case 9: Mensaje = "9. Dos operadores estén seguidos. Ejemplo: 2++4, 5-*3"; break;
        case 10: Mensaje = "10. Un operador seguido de un paréntesis que cierra. Ejemplo: 2-
(4+)-7"; break;
        case 11: Mensaje = "11. Una letra seguida de número. Ejemplo: 7-2a-6"; break;
        case 12: Mensaje = "12. Una letra seguida de punto. Ejemplo: 7-a.-6"; break;
        case 13: Mensaje = "13. Una letra seguida de otra letra. Ejemplo: 4-xy+3"; break;
        case 14: Mensaje = "14. Una letra seguida de un paréntesis que abre. Ejemplo: 2-
a(8*3)"; break;
        case 15: Mensaje = "15. Un paréntesis que abre seguido de un punto. Ejemplo: 7-(.8+4)-
6"; break;
        case 16: Mensaje = "16. Un paréntesis que abre y sigue un operador. Ejemplo: (+3-5)*7";
break;
        case 17: Mensaje = "17. Un paréntesis que abre y sigue un paréntesis que cierra.
Ejemplo: 4+()*2"; break;
        case 18: Mensaje = "18. Un paréntesis que cierra y sigue un número. Ejemplo: (3-5)8";
break;
    }
}

```

```

        case 19: Mensaje = "19. Un paréntesis que cierra y sigue un punto. Ejemplo: (3-5).+2";
break;
        case 20: Mensaje = "20. Un paréntesis que cierra y sigue una letra. Ejemplo: 2-
(7*3) k+7"; break;
        case 21: Mensaje = "21. Un paréntesis que cierra y sigue un paréntesis que abre.
Ejemplo: (4-3)(2+1)"; break;
        case 22: Mensaje = "22. Inicia con un operador. Ejemplo: *3+5"; break;
        case 23: Mensaje = "23. Finaliza con un operador. Ejemplo: 7+9*"; break;
        case 24: Mensaje = "24. No hay correspondencia entre paréntesis que cierran y abren";
break;
        case 25: Mensaje = "25. El número de paréntesis que cierran no es igual al número de
paréntesis que abren"; break;
        case 26: Mensaje = "26. Dos o más puntos en número real"; break;
    }
    return Mensaje;
}

/* Da valor a las variables que tendrá la expresión algebraica */
public void DarValorVariable(char varAlgebra, double Valor) {
    Valores[varAlgebra - 'a'] = Valor;
}

/* Evalúa la expresión convertida en piezas */
public double Evaluar() {
    double Resultado = 0;
    PiezaEvl4 tmpPieza;

    /* Va de pieza en pieza */
    for (int Posicion = 0; Posicion < Piezas.Count; Posicion++) {
        tmpPieza = Piezas[Posicion];

        switch (tmpPieza.Operador) {
            case 0: Resultado = Valores[tmpPieza.PosValorA] + Valores[tmpPieza.PosValorB];
break;
            case 1: Resultado = Valores[tmpPieza.PosValorA] - Valores[tmpPieza.PosValorB];
break;
            case 2: Resultado = Valores[tmpPieza.PosValorA] * Valores[tmpPieza.PosValorB];
break;
            case 3: Resultado = Valores[tmpPieza.PosValorA] / Valores[tmpPieza.PosValorB];
break;
            default: Resultado = Math.Pow(Valores[tmpPieza.PosValorA],
Valores[tmpPieza.PosValorB]); break;
        }

        switch (tmpPieza.Funcion) {
            case 0: Resultado = Math.Sin(Resultado); break;
            case 1: Resultado = Math.Cos(Resultado); break;
            case 2: Resultado = Math.Tan(Resultado); break;
            case 3: Resultado = Math.Abs(Resultado); break;
            case 4: Resultado = Math.Asin(Resultado); break;
            case 5: Resultado = Math.Acos(Resultado); break;
            case 6: Resultado = Math.Atan(Resultado); break;
            case 7: Resultado = Math.Log(Resultado); break;
            case 8: Resultado = Math.Ceiling(Resultado); break;
            case 9: Resultado = Math.Exp(Resultado); break;
            case 10: Resultado = Math.Sqrt(Resultado); break;
        }

        Valores[tmpPieza.PosResultado] = Resultado;
    }
    return Resultado;
}

/* Divide la expresión en partes, yendo de carácter en carácter */
private void CrearPartes() {
    StringBuilder Numero = new StringBuilder();
    for (int Posicion = 0; Posicion < this.ParaSerAnalizada.Length; Posicion++) {
        char Letra = this.ParaSerAnalizada[Posicion];
        switch (Letra) {
            case '.':
            case '0':
            case '1':
            case '2':
            case '3':
            case '4':
            case '5':
            case '6':
            case '7':
            case '8':
            case '9': /* Si es un dígito o un punto va acumulando el número */
        }
    }
}

```

```

        Numero.Append(Letra); break;
    case '+':
    case '-':
    case '*':
    case '/':
    case '^': /* Si es un operador matemático entonces verifica si hay un número que
se ha acumulado */
        if (Numero.Length > 0) {
            Valores.Add(double.Parse(Numero.ToString(), CultureInfo.InvariantCulture));
            Partes.Add(new ParteEvl4(ESNUMERO, Valores.Count - 1));
            Numero.Clear();
        }
        /* Agregar el operador matemático */
        Partes.Add(new ParteEvl4(Letra));
        break;

    case '(': /* Es paréntesis que abre */
        Partes.Add(new ParteEvl4(ESPARABRE, 0));
        break;

    case ')': /* Si es un paréntesis que cierra entonces verifica si hay un número
que se ha acumulado */
        if (Numero.Length > 0) {
            Valores.Add(double.Parse(Numero.ToString(), CultureInfo.InvariantCulture));
            Partes.Add(new ParteEvl4(ESNUMERO, Valores.Count - 1));
            Numero.Clear();
        }
        /* Si sólo había un número o variable dentro del paréntesis le agrega + 0
(por ejemplo: sen(x) o 3*(2) ) */
        if (Partes[Partes.Count - 2].Tipo == ESPARABRE || Partes[Partes.Count - 2].Tipo == ESFUNCION) {
            Partes.Add(new ParteEvl4(ESOPERADOR, 0));
            Valores.Add(0);
            Partes.Add(new ParteEvl4(ESNUMERO, Valores.Count - 1));
        }

        /* Adiciona el paréntesis que cierra */
        Partes.Add(new ParteEvl4(ESPARCIERRA, 0));
        break;
    case 'A': /* Seno */
    case 'B': /* Coseno */
    case 'C': /* Tangente */
    case 'D': /* Valor absoluto */
    case 'E': /* Arcoseno */
    case 'F': /* Arcocoseno */
    case 'G': /* Arcotangente */
    case 'H': /* Logaritmo natural */
    case 'I': /* Exponencial */
    case 'J': /* Raíz cuadrada */
        Partes.Add(new ParteEvl4(ESFUNCION, Letra - 'A'));
        Posicion++;
        break;
    default:
        Partes.Add(new ParteEvl4(ESVARIABLE, Letra - 'a'));
        break;
    }
}

/* Convierte las partes en las piezas finales de ejecución */
private void CrearPiezas() {
    int Contador = Partes.Count - 1;
    do {
        ParteEvl4 tmpParte = Partes[Contador];
        if (tmpParte.Tipo == ESPARABRE || tmpParte.Tipo == ESFUNCION) {
            GenerarPiezasOperador(4, 4, Contador); /* Evalúa las potencias */
            GenerarPiezasOperador(2, 3, Contador); /* Luego evalúa multiplicar y dividir */
            GenerarPiezasOperador(0, 1, Contador); /* Finalmente evalúa sumar y restar */

            if (tmpParte.Tipo == ESFUNCION) { /* Agrega la función a la última pieza */
                Piezas[Piezas.Count - 1].Funcion = tmpParte.Funcion;
            }
        }

        /* Quita el paréntesis/función que abre y el que cierra, dejando el centro */
        Partes.RemoveAt(Contador);
        Partes.RemoveAt(Contador + 1);
    }
}

```

```

        Contador--;
    } while (Contador >= 0);
}

/* Genera las piezas buscando determinado operador */
private void GenerarPiezasOperador(int OperA, int OperB, int Inicia) {
    int Contador = Inicia + 1;
    do {
        ParteEvl4 tmpParte = Partes[Contador];
        if (tmpParte.Tipo == ESOPERADOR && (tmpParte.Operador == OperA || tmpParte.Operador == OperB)) {
            ParteEvl4 tmpParteIzq = Partes[Contador - 1];
            ParteEvl4 tmpParteDer = Partes[Contador + 1];

            /* Crea Pieza */
            PiezaEvl4 NuevaPieza = new PiezaEvl4();
            NuevaPieza.Funcion = -1;

            switch (tmpParteIzq.Tipo) {
                case ESNUMERO: NuevaPieza.PosValorA = tmpParteIzq.posNumero; break;
                case ESVARIABLE: NuevaPieza.PosValorA = tmpParteIzq.posVariable; break;
                default: NuevaPieza.PosValorA = tmpParteIzq.posAcumula; break;
            }

            NuevaPieza.Operador = tmpParte.Operador;

            switch (tmpParteDer.Tipo) {
                case ESNUMERO: NuevaPieza.PosValorB = tmpParteDer.posNumero; break;
                case ESVARIABLE: NuevaPieza.PosValorB = tmpParteDer.posVariable; break;
                default: NuevaPieza.PosValorB = tmpParteDer.posAcumula; break;
            }

            /* Añade a lista de piezas y crea una nueva posición en Valores */
            Valores.Add(0);
            NuevaPieza.PosResultado = Valores.Count - 1;
            Piezas.Add(NuevaPieza);

            /* Elimina la parte del operador y la siguiente */
            Partes.RemoveAt(Contador);
            Partes.RemoveAt(Contador);

            /* Retorna el contador en uno para tomar la siguiente operación */
            Contador--;
        }
        /* Cambia la parte anterior por parte que acumula */
        tmpParteIzq.Tipo = ESACUMULA;
        tmpParteIzq.posAcumula = NuevaPieza.PosResultado;
    }
    Contador++;
} while (Partes[Contador].Tipo != ESPARCIERRA);
}

private int ChequeaSintaxis(string ExpresionOriginal) {
    /* Primero a minúsculas */
    StringBuilder Minusculas = new StringBuilder(ExpresionOriginal.ToLower());

    /* Sólo los caracteres válidos */
    HashSet<char> Permitidos = new HashSet<char>("abcdefghijklmnopqrstuvwxyz0123456789.+-*/^()");

    StringBuilder ConLetrasValidas = new StringBuilder("(");
    for (int Contador = 0; Contador < Minusculas.Length; Contador++)
        if (Permitidos.Contains(Minusculas[Contador]))
            ConLetrasValidas.Append(Minusculas[Contador]);
    ConLetrasValidas.Append(')');

    /* Agrega +0) donde exista )) porque es necesario para crear las piezas */
    string nuevo = ConLetrasValidas.ToString();
    while (nuevo.IndexOf(")")) != -1) nuevo = nuevo.Replace(") ", ")+0)");
    ConLetrasValidas = new StringBuilder(nuevo);

    /* Validación de sintaxis, se genera una copia y allí se reemplaza las funciones por a+( */
    StringBuilder ParaChequearSintaxis = new StringBuilder();
    ParaChequearSintaxis.Append(ConLetrasValidas);
    ParaChequearSintaxis.Replace("sen(", "a+(").Replace("cos(", "a+(").Replace("tan(",
    "a+(").Replace("abs(", "a+(").Replace("asn(", "a+(").Replace("acs(", "a+(").Replace("atn(",
    "a+(").Replace("log(", "a+(").Replace("cei(", "a+(").Replace("exp(", "a+(").Replace("sqr(", "a+(");

    for (int Contador = 1; Contador < ParaChequearSintaxis.Length - 2; Contador++) {
        char cA = ParaChequearSintaxis[Contador];
        char cB = ParaChequearSintaxis[Contador + 1];
    }
}

```

```

        if (Char.IsDigit(cA) && Char.IsLower(cB)) return 1;
        if (Char.IsDigit(cA) && cB == '(') return 2;
        if (cA == '.' && cB == '.') return 3;
        if (cA == '.' && EsUnOperador(cB)) return 4;
        if (cA == '.' && Char.IsLower(cB)) return 5;
        if (cA == '.' && cB == ')') return 6;
        if (cA == '.' && cB == ')') return 7;
        if (EsUnOperador(cA) && cB == '.') return 8;
        if (EsUnOperador(cA) && EsUnOperador(cB)) return 9;
        if (EsUnOperador(cA) && cB == ')') return 10;
        if (Char.IsLower(cA) && Char.IsDigit(cB)) return 11;
        if (Char.IsLower(cA) && cB == '.') return 12;
        if (Char.IsLower(cA) && Char.IsLower(cB)) return 13;
        if (Char.IsLower(cA) && cB == '(') return 14;
        if (cA == '(' && cB == '.') return 15;
        if (cA == '(' && EsUnOperador(cB)) return 16;
        if (cA == '(' && cB == ')') return 17;
        if (cA == ')') && Char.IsDigit(cB)) return 18;
        if (cA == ')') && cB == '.') return 19;
        if (cA == ')') && Char.IsLower(cB)) return 20;
        if (cA == ')') && cB == '(') return 21;
    }

    /* Valida el inicio y fin de la expresión */
    if (EsUnOperador(ParaChequearSintaxis[1])) return 22;
    if (EsUnOperador(ParaChequearSintaxis[ParaChequearSintaxis.Length - 2])) return 23;

    /* Valida balance de paréntesis */
    int ParentesisAbre = 0; /* Contador de paréntesis que abre */
    int ParentesisCierra = 0; /* Contador de paréntesis que cierra */
    for (int Contador = 1; Contador < ParaChequearSintaxis.Length-1; Contador++) {
        switch (ParaChequearSintaxis[Contador]) {
            case '(': ParentesisAbre++; break;
            case ')': ParentesisCierra++; break;
        }
        if (ParentesisCierra > ParentesisAbre) return 24;
    }
    if (ParentesisAbre != ParentesisCierra) return 25;

    /* Validar los puntos decimales de un número real */
    int TotalPuntos = 0;
    for (int Contador = 0; Contador < ParaChequearSintaxis.Length; Contador++) {
        if (EsUnOperador(ParaChequearSintaxis[Contador])) TotalPuntos = 0;
        if (ParaChequearSintaxis[Contador] == '.') TotalPuntos++;
        if (TotalPuntos > 1) return 26;
    }

    /* Deja la expresión para ser analizada. Reemplaza las funciones de tres letras por una
     letra mayúscula. Cambia los )) por )+0) porque es requerido al crear las piezas */
    this.ParaSerAnalizada.Length = 0;
    this.ParaSerAnalizada.Append(ConLetrasValidas);
    this.ParaSerAnalizada.Replace("sen", "A").Replace("cos", "B").Replace("tan",
"C").Replace("abs", "D").Replace("asn", "E").Replace("acs", "F").Replace("atn", "G").Replace("log",
"H").Replace("exp", "I").Replace("sqr", "J");

    /* Sintaxis correcta */
    return 0;
}

/* Retorna si el Caracter es un operador matemático */
private static bool EsUnOperador(char Caracter) {
    switch (Caracter) {
        case '+':
        case '-':
        case '*':
        case '/':
        case '^':
            return true;
        default:
            return false;
    }
}
}
}

```

Como usar el evaluador de expresiones

En Visual Studio 2022, se añade la clase al proyecto:

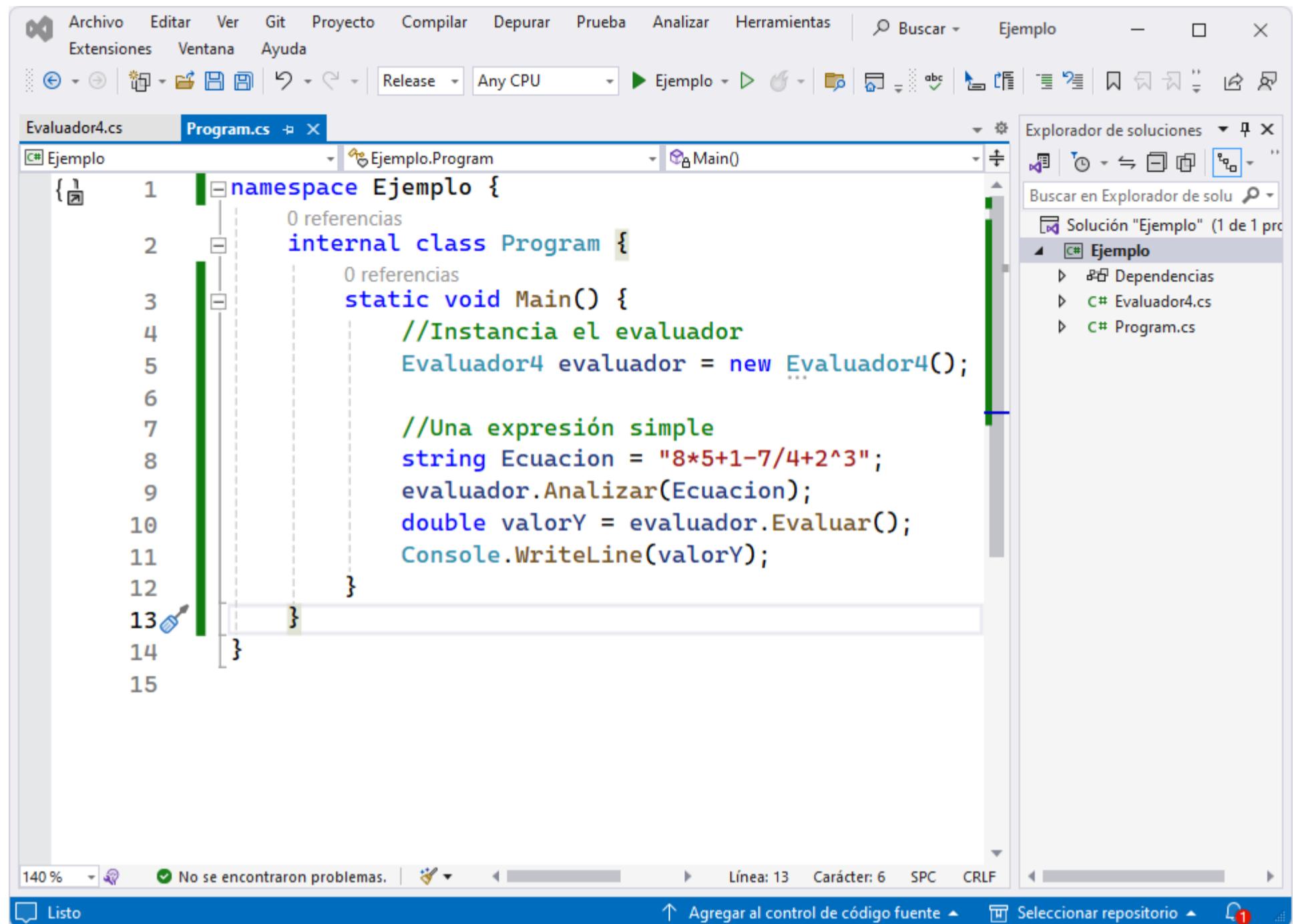


Ilustración 224: Uso del evaluador de expresiones

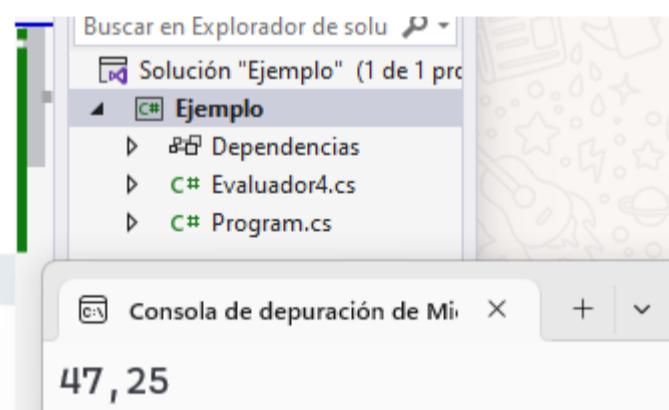


Ilustración 225: Resultado obtenido con el evaluador

G/001.cs

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Instancia el evaluador
            Evaluador4 evaluador = new Evaluador4();

            //Una expresión simple
            string Ecuacion = "8*5+1-7/4+2^3";
            evaluador.Analizar(Ecuacion);
            double valorY = evaluador.Evaluar();
            Console.WriteLine(valorY);
        }
    }
}
```

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Instancia el evaluador
            Evaluador4 evaluador = new Evaluador4();

            //Una expresión simple con números reales
            string Ecuacion = "7.318+5.0045-9.071^2*8.04961";
            evaluador.Analizar(Ecuacion);
            double valorY = evaluador.Evaluar();
            Console.WriteLine(valorY);
        }
    }
}
```



The screenshot shows a Windows-style application window titled 'Consola de depuración de Mi...' (Breakpoint Console of My...). The window contains a single line of text: '-650,02388966401'. There are standard window controls like a close button (X) and a minimize/maximize button.

Ilustración 226: Evaluador 4 operando números reales

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Instancia el evaluador
            Evaluador4 evaluador = new Evaluador4();

            //Una expresión con paréntesis
            string Ecuacion = "5*(3+2.5)-7/(8.03*2-5^3)+4.1*(3-(5.8*2.3))";
            evaluador.Analizar(Ecuacion);
            double valorY = evaluador.Evaluar();
            Console.WriteLine(valorY);
        }
    }
}
```

The screenshot shows a Windows-style application window titled 'Consola de depuración de Mi'. Inside the window, the expression '5*(3+2.5)-7/(8.03*2-5^3)+4.1*(3-(5.8*2.3))' is displayed, followed by the result '-14,829744446484295'.

Ilustración 227: Evaluador 4 operando con paréntesis

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Instancia el evaluador
            Evaluador4 evaluador = new Evaluador4();

            //Una expresión con funciones
            string Ecuacion = "sen(4.90+2.34)-cos(1.89)*tan(3)/abs(4-12)+asn(0.12)-acs(0-
0.4)+atn(0.03)*log(1.3)+cei(3.4)+exp(2.8)-sqr(9)";
            evaluador.Analizar(Ecuacion);
            double valorY = evaluador.Evaluar();
            Console.WriteLine(valorY);
        }
    }
}
```

The screenshot shows a Windows-style application window titled 'Consola de depuración de Mi'. Inside the window, the value '-8096,683927575385' is displayed in a monospaced font, indicating the result of the mathematical expression evaluated by the program.

Ilustración 228: Evaluador 4 operando con funciones

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Instancia el evaluador
            Evaluador4 evaluador = new Evaluador4();

            //Una expresión con uso de variables (deben estar en minúsculas)
            string Ecuacion = "3*x+2*y";
            evaluador.Analizar(Ecuacion);

            //Le da valor a las variables (deben estar en minúsculas)
            evaluador.DarValorVariable('x', 1.8);
            evaluador.DarValorVariable('y', 3.5);
            double valorY = evaluador.Evaluar();
            Console.WriteLine(valorY);
        }
    }
}
```

```
static void Main() {
    //Instancia el evaluador
    Evaluador4 evaluador = new Evaluador4();

    //Una expresión con uso de variables (deben estar en minúsculas)
    string Ecuacion = "3*x+2*y";
    evaluador.Analizar(Ecuacion);

    //Le da valor a las variables (deben estar en minúsculas)
    evaluador.DarValorVariable('x', 1.8);
    evaluador.DarValorVariable('y', 3.5);
    double valorY = evaluador.Evaluar();
    Console.WriteLine(valorY);
}
```

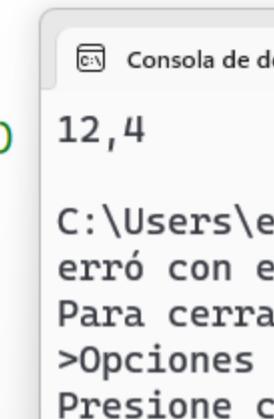


Ilustración 229: Evaluador 4 operando con uso de variables

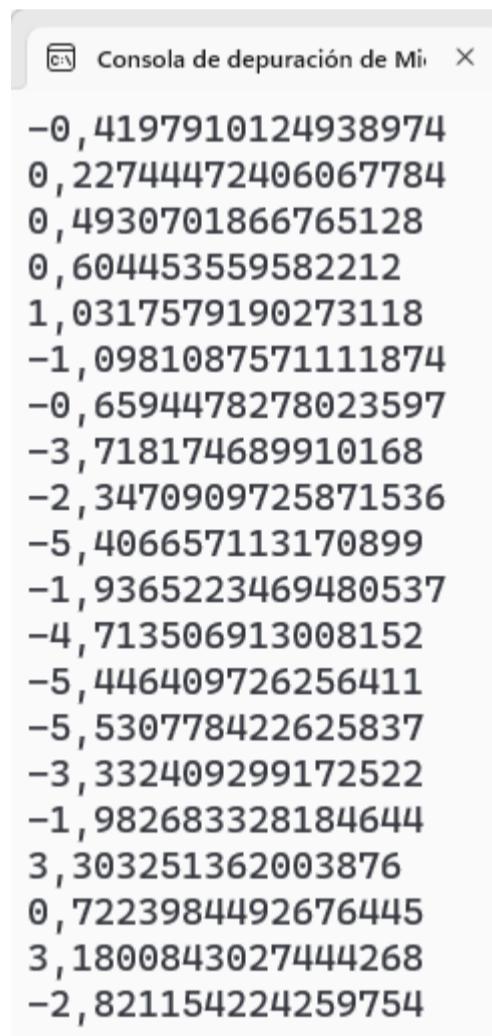
```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            Random Azar = new Random();

            //Instancia el evaluador
            Evaluador4 evaluador = new Evaluador4();

            //Una expresión con uso de variables (deben estar en minúsculas)
            string Ecuacion = "3*cos(2*x+4)-5*sen(4*y-7)";

            //Se analiza primero la ecuación
            evaluador.Analizar(Ecuacion);

            //Después de ser analizada, se le dan los valores a las variables
            //esto hace que el evaluador sea muy rápido
            for (int cont = 1; cont <= 20; cont++) {
                evaluador.DarValorVariable('x', Azar.NextDouble());
                evaluador.DarValorVariable('y', Azar.NextDouble());
                double valorY = evaluador.Evaluar();
                Console.WriteLine(valorY);
            }
        }
    }
}
```



```
-0,4197910124938974
0,22744472406067784
0,4930701866765128
0,604453559582212
1,0317579190273118
-1,0981087571111874
-0,6594478278023597
-3,718174689910168
-2,3470909725871536
-5,406657113170899
-1,9365223469480537
-4,713506913008152
-5,446409726256411
-5,530778422625837
-3,332409299172522
-1,982683328184644
3,303251362003876
0,7223984492676445
3,1800843027444268
-2,821154224259754
```

Ilustración 230: Evaluador 4 usado para generar múltiples valores de una ecuación al cambiar los valores.

Validando la sintaxis

El evaluador 4.0 verifica si la expresión algebraica es sintácticamente correcta al llamar al método Analizar(), si este método retorna el valor cero, significa que la expresión es correcta o no tiene errores de sintaxis, caso contrario, retorna un código de error.

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            string[] exprAlgebraica = new string[] {
                "2q-(*3)", //0
                "7-2(5-6)", //1
                "3..1", //2
                "3.*1", //3
                "3+5.w-8", //4
                "2-5.(4+1)*3", //5
                "2-(5.)*3", //6
                "2-(4+.1)-7", //7
                "5-*3", //8
                "2-(4+)-7", //9
                "7-a2-6", //10
                "7-a.4*3", //11
                "7-qw*9", //12
                "2-u(7-3)", //13
                "7-(.8+4)-6", //14
                "(+3-5)*7", //15
                "4+()*2", //16
                "(3-5)8", //17
                "(3-5).+2", //18
                "2-(7*3)k+7", //19
                "(4-3)(2+1)", //20
                "*3+5", //21
                "3*5*", //22
                "9*4)+(2-6", //23
                "((2+4)", //24
                "2.71*3.56.01" }; //25

            Evaluador4 evaluador = new Evaluador4();

            for (int num = 0; num < exprAlgebraica.Length; num++) {
                Console.WriteLine("\r\nExpresión " + num + ": " + exprAlgebraica[num]);
                int ErrorSintaxis = evaluador.Analizar(exprAlgebraica[num]);
                if (ErrorSintaxis >= 0) {
                    Console.WriteLine(evaluador.MensajeError(ErrorSintaxis));
                }
            }
        }
    }
}
```

Expresión 0: 2q-(*)3

1. Un número seguido de una letra. Ejemplo: 2q-(*)3

Expresión 1: 7-2(5-6)

2. Un número seguido de un paréntesis que abre. Ejemplo: 7-2(5-6)

Expresión 2: 3..1

3. Doble punto seguido. Ejemplo: 3..1

Expresión 3: 3.*1

4. Punto seguido de operador. Ejemplo: 3.*1

Expresión 4: 3+5.w-8

5. Un punto y sigue una letra. Ejemplo: 3+5.w-8

Expresión 5: 2-5.(4+1)*3

6. Punto seguido de paréntesis que abre. Ejemplo: 2-5.(4+1)*3

Expresión 6: 2-(5.)*3

7. Punto seguido de paréntesis que cierra. Ejemplo: 2-(5.)*3

Expresión 7: 2-(4+.1)-7

8. Un operador seguido de un punto. Ejemplo: 2-(4+.1)-7

Expresión 8: 5-*3

9. Dos operadores estén seguidos. Ejemplo: 2++4, 5-*3

Expresión 9: 2-(4+)-7

10. Un operador seguido de un paréntesis que cierra. Ejemplo: 2-(4+)-7

Ilustración 231: Evaluando la sintaxis

Expresión 10: 7-a2-6

11. Una letra seguida de número. Ejemplo: 7-2a-6

Expresión 11: 7-a.4*3

12. Una letra seguida de punto. Ejemplo: 7-a.-6

Expresión 12: 7-qw*9

13. Una letra seguida de otra letra. Ejemplo: 4-xy+3

Expresión 13: 2-u(7-3)

14. Una letra seguida de un paréntesis que abre. Ejemplo: 2-a(8*3)

Expresión 14: 7-(.8+4)-6

15. Un paréntesis que abre seguido de un punto. Ejemplo: 7-(.8+4)-6

Expresión 15: (+3-5)*7

16. Un paréntesis que abre y sigue un operador. Ejemplo: (+3-5)*7

Expresión 16: 4+()*2

17. Un paréntesis que abre y sigue un paréntesis que cierra. Ejemplo: 4+()*2

Expresión 17: (3-5)8

18. Un paréntesis que cierra y sigue un número. Ejemplo: (3-5)8

Expresión 18: (3-5).+2

19. Un paréntesis que cierra y sigue un punto. Ejemplo: (3-5).+2

Expresión 19: 2-(7*3)k+7

20. Un paréntesis que cierra y sigue una letra. Ejemplo: 2-(7*3)k+7

Ilustración 232: Evaluando la sintaxis

Expresión 20: (4-3)(2+1)

21. Un paréntesis que cierra y sigue un paréntesis que abre. Ejemplo: (4-3)(2+1)

Expresión 21: *3+5

22. Inicia con un operador. Ejemplo: *3+5

Expresión 22: 3*5*

23. Finaliza con un operador. Ejemplo: 7+9*

Expresión 23: 9*4)+(2-6

24. No hay correspondencia entre paréntesis que cierran y abren

Expresión 24: ((2+4)

25. El número de paréntesis que cierran no es igual al número de paréntesis que abren

Expresión 25: 2.71*3.56.01

26. Dos o más puntos en número real

Ilustración 233: Evaluando la sintaxis

Métricas: Comparativa de desempeño con el evaluador interno de C#

.NET 8 tiene un evaluador de expresiones propio, así que una forma de probar que el evaluador está funcionando bien es generando ecuaciones al azar y comparar los resultados entre el evaluador propio de C# y el evaluador 4.0, si son iguales, significa que todo marcha bien. A continuación, el código que genera ecuaciones al azar y usa ambos evaluadores, compara los resultados y además el tiempo que tarda uno y otro.

```
using System.Data;
using System.Diagnostics;

namespace Ejemplo {
    internal class Program {
        static void Main() {
            Random Azar = new Random();

            //Versión 2024
            Evaluador4 evaluador2024 = new Evaluador4();

            //Arreglos que guardan valores de X, Y, Z
            double[] arregloX = new double[200];
            double[] arregloY = new double[200];
            double[] arregloZ = new double[200];
            for (int cont = 0; cont < arregloX.Length; cont++) {
                arregloX[cont] = Azar.NextDouble();
                arregloY[cont] = Azar.NextDouble();
                arregloZ[cont] = Azar.NextDouble();
            }

            //Prueba evaluador
            long TotalTiempo2024Evalua = 0, TotalTiempo2024Analiza = 0;
            double valor2024, AcumValor2024 = 0;

            //Evaluador interno
            long TotalTiempoInterno = 0;
            double valorInterno, AcumInterno = 0;

            //Toma el tiempo
            Stopwatch temporizador = new Stopwatch();

            for (int num = 1; num <= 1000; num++) {
                string ecuacion = EcuacionAzar(350, Azar);
                //Console.WriteLine(ecuacion);

                //Versión 2024. Análisis.
                temporizador.Reset();
                temporizador.Start();
                evaluador2024.Analizar(ecuacion);
                temporizador.Stop();
                TotalTiempo2024Analiza += temporizador.ElapsedTicks;

                //Versión 2024. Evaluación
                temporizador.Reset();
                temporizador.Start();
                valor2024 = 0;
                for (int cont = 0; cont < arregloX.Length; cont++) {
                    evaluador2024.DarValorVariable('x', arregloX[cont]);
                    evaluador2024.DarValorVariable('y', arregloY[cont]);
                    evaluador2024.DarValorVariable('z', arregloZ[cont]);
                    valor2024 += Math.Abs(evaluador2024.Evaluar());
                }
                temporizador.Stop();
                TotalTiempo2024Evalua += temporizador.ElapsedTicks;

                //Compara contra el evaluador de expresiones propio que tiene C#
                var EvaluadorInterno = new DataTable();
                temporizador.Reset();
                temporizador.Start();
                valorInterno = 0;
                for (int cont = 0; cont < arregloX.Length; cont++) {
                    string ecuacion2 = ecuacion.Replace("x", arregloX[cont].ToString()).Replace("y",
arregloY[cont].ToString()).Replace("z", arregloZ[cont].ToString().Replace(", ", "."));
                    valorInterno += Math.Abs(Convert.ToDouble(EvaluadorInterno.Compute(ecuacion2,
"")));
                }
                temporizador.Stop();
                TotalTiempoInterno += temporizador.ElapsedTicks;
            }
        }
    }
}
```

```

        if (Math.Abs(valor2024) > 10000000) continue;
        if (double.IsNaN(valor2024) || double.IsInfinity(valor2024)) continue;

        AcumValor2024 += valor2024;
        AcumInterno += valorInterno;
    }
    Console.WriteLine("Evaluador 2024 acumula: " + AcumValor2024);
    Console.WriteLine("Interno C# acumula: " + AcumInterno);

    Console.WriteLine("\r\nEvaluador 2024 tiempo para evaluar: " + TotalTiempo2024Evalua);
    Console.WriteLine("Evaluador 2024 tiempo para analizar: " + TotalTiempo2024Analiza);
    long TotalTiempo = TotalTiempo2024Evalua + TotalTiempo2024Analiza;

    Console.WriteLine("\r\nEvaluador 2024 tiempo para analizar y evaluar: " + TotalTiempo);
    Console.WriteLine("Interno tiempo para analizar y evaluar: " + TotalTiempoInterno);

}

public static string EcuacionAzar(int Longitud, Random Azar) {
    int cont = 0;
    int numParentesisAbre = 0;

    string Ecuacion = "";
    while (cont < Longitud) {

        //Función o paréntesis o nada
        if (Azar.NextDouble() < 0.5) {
            Ecuacion += "(";
            numParentesisAbre++;
            cont++;
        }

        //Variable o número
        cont++;
        switch (Azar.Next(4)) {
            case 0: Ecuacion += NumeroAzar(Azar); break;
            case 1: Ecuacion += "x"; break;
            case 2: Ecuacion += "y"; break;
            case 3: Ecuacion += "z"; break;
        }

        //Paréntesis que cierra
        int numParentesisCierra = Azar.Next(numParentesisAbre + 1);
        for (int num = 1; num <= numParentesisCierra; num++) {
            Ecuacion += ")";
            numParentesisAbre--;
            cont++;
        }

        //Operador
        cont++;
        Ecuacion += OperadorAzar(Azar);
    }

    //Variable o número
    switch (Azar.Next(4)) {
        case 0: Ecuacion += NumeroAzar(Azar); break;
        case 1: Ecuacion += "x"; break;
        case 2: Ecuacion += "y"; break;
        case 3: Ecuacion += "z"; break;
    }

    for (int num = 0; num < numParentesisAbre; num++) Ecuacion += ")";
}

return Ecuacion;
}

private static string OperadorAzar(Random azar) {
    string[] operadores = { "+", "-", "*" };
    return operadores[azar.Next(operadores.Length)];
}

private static string NumeroAzar(Random azar) {
    return "0." + Convert.ToString(azar.Next(1000000) + 1);
}
}
}

```

```
Consola de depuración de Mi X + ▾  
Evaluador 2024 acumula: 617943,9211954527  
Interno C#     acumula: 617943,9211954527  
  
Evaluador 2024 tiempo para evaluar: 1973204  
Evaluador 2024 tiempo para analizar: 1498736  
  
Evaluador 2024 tiempo para analizar y evaluar: 3471940  
Interno      tiempo para analizar y evaluar: 103962015
```

Ilustración 234: Métrica compara ambos evaluadores

Como se puede observar, ambos evaluadores obtienen los mismos resultados evaluando 1000 ecuaciones, cada una del tamaño de 350 caracteres y 300 valores distintos de variables por ecuación. Luego está correcto.

En el lado de desempeño, el Evaluador 4.0 es más rápido que el evaluador interno. La explicación a esta diferencia tan alta (a favor del Evaluador 4) es que el evaluador está diseñado, escrito y optimizado sólo para ecuaciones algebraicas, mientras el evaluador interno de C# es más genérico para diversos tipos de expresiones. La especialización vence.

Otras pruebas:

```
Consola de depuración de Mi X + ▾  
Evaluador 2024 acumula: 671720,3600586591  
Interno C#     acumula: 671720,360058659  
  
Evaluador 2024 tiempo para evaluar: 2023728  
Evaluador 2024 tiempo para analizar: 1434936  
  
Evaluador 2024 tiempo para analizar y evaluar: 3458664  
Interno      tiempo para analizar y evaluar: 104284300
```

Ilustración 235: Métrica compara ambos evaluadores

```
Consola de depuración de Mi X + ▾  
Evaluador 2024 acumula: 648049,8571649974  
Interno C#     acumula: 648049,8571649973  
  
Evaluador 2024 tiempo para evaluar: 1841477  
Evaluador 2024 tiempo para analizar: 1437827  
  
Evaluador 2024 tiempo para analizar y evaluar: 3279304  
Interno      tiempo para analizar y evaluar: 105466114
```

Ilustración 236: Métrica compara ambos evaluadores

Parte 8. Estructuras de datos de bajo nivel

Lista simplemente enlazada

Se representa de esta forma:

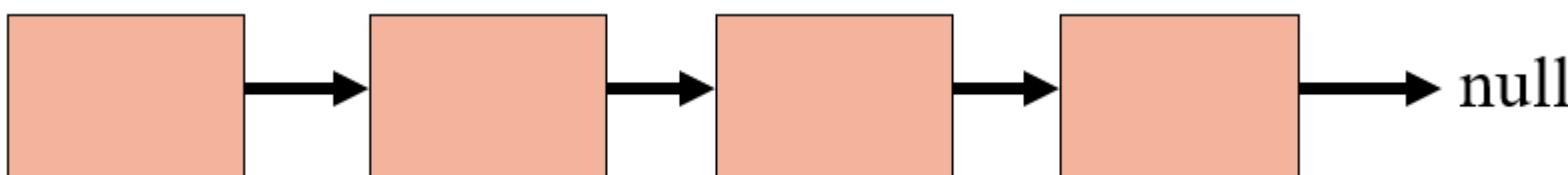


Ilustración 224: Representación gráfica de una lista simplemente enlazada

El rectángulo es el objeto con sus datos, métodos y un apuntador, se le conoce como Nodo

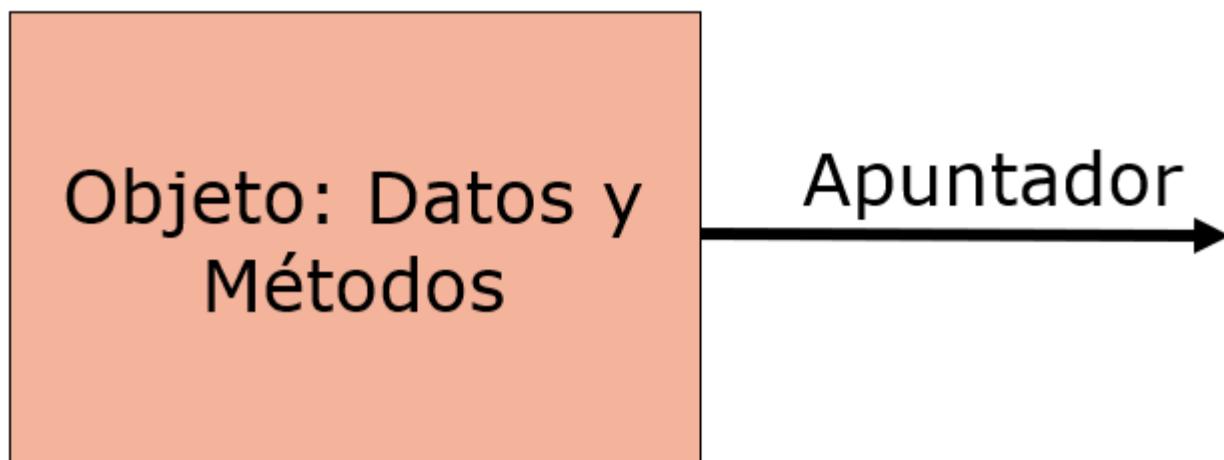


Ilustración 225: El nodo es un objeto con sus atributos, métodos y un apuntador

Esta sería un ejemplo de implementación del Nodo en C#:

H/001.cs

```
namespace Ejemplo {
    class Nodo {
        //Atributos propios
        public string Cadena { get; set; }
        public char Caracter { get; set; }
        public int Entero { get; set; }
        public double NumReal { get; set; }

        //Apuntador para lista simplemente enlazada
        public Nodo Apuntador;

        //Constructor
        public Nodo(string Cadena, char Caracter, int Entero, double NumReal) {
            this.Cadena = Cadena;
            this.Caracter = Caracter;
            this.Entero = Entero;
            this.NumReal = NumReal;
        }

        //Imprime Contenido
        public void Imprime() {
            Console.Write("Cadena: " + Cadena + " Caracter: " + Caracter.ToString());
            Console.WriteLine(" Entero: " + Entero.ToString() + " Real: " + NumReal.ToString());
        }
    }

    class Program {
        static void Main() {
            //Crea dos nodos separados
            Nodo primero = new Nodo("Rafael", 'A', 16, 8.32);
            Nodo segundo = new Nodo("Moreno", 'P', 9, 2.9);
            Nodo tercero = new Nodo("Sally", 'C', 2010, 7.18);

            //Une el primer nodo con el segundo, creando una simple lista
            primero.Apuntador = segundo;

            //Une el segundo nodo con el tercero, aumentando la lista
            segundo.Apuntador = tercero;

            //Imprime la lista
            primero.Imprime();
            primero.Apuntador.Imprime();
            primero.Apuntador.Apuntador.Imprime();
        }
    }
}
```

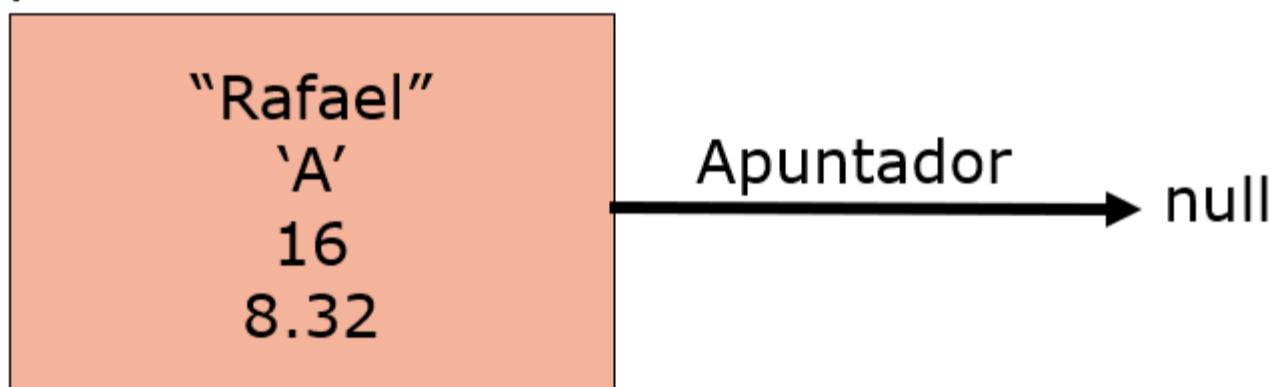
}

```
Consola de depuración de Mi... + ▾
Cadena: Rafael Caracter: A Entero: 16 Real: 8,32
Cadena: Moreno Caracter: P Entero: 9 Real: 2,9
Cadena: Sally Caracter: C Entero: 2010 Real: 7,18
```

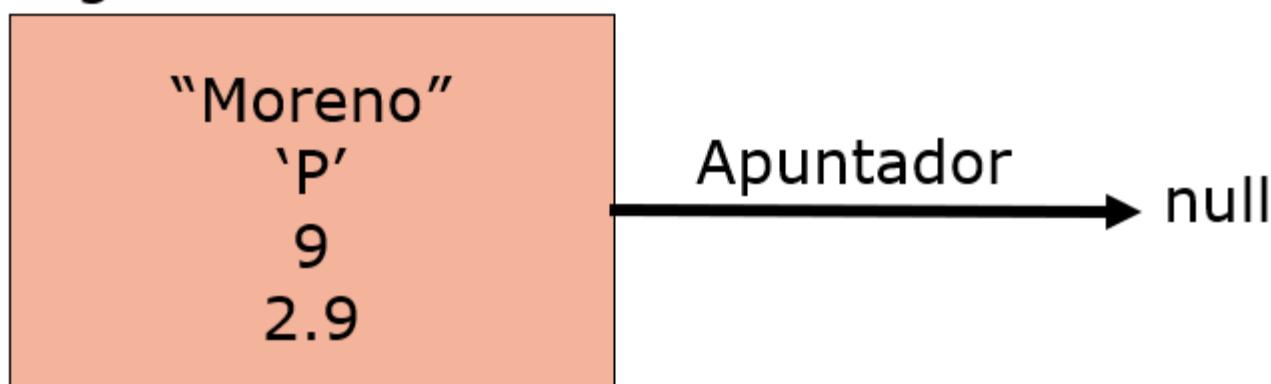
Ilustración 239: Lista simplemente enlazada

Se crean tres nodos:

primero



segundo



tercero

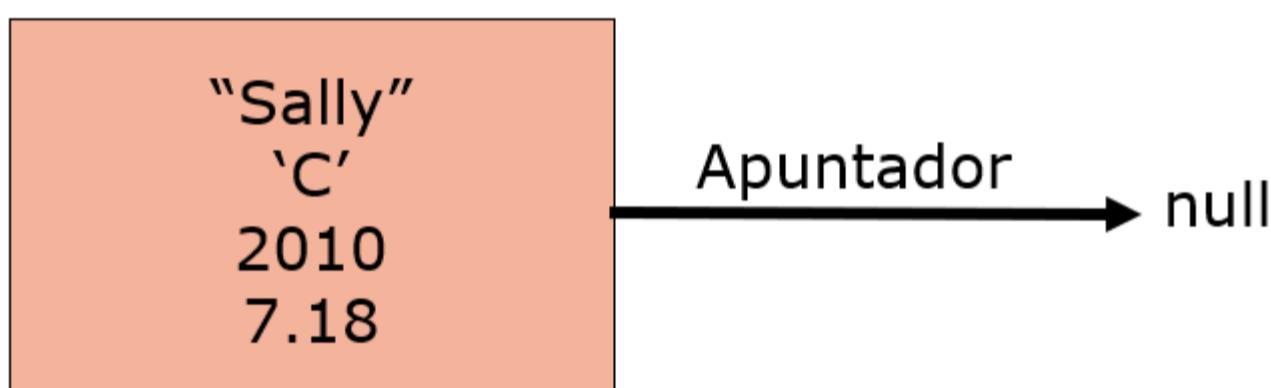


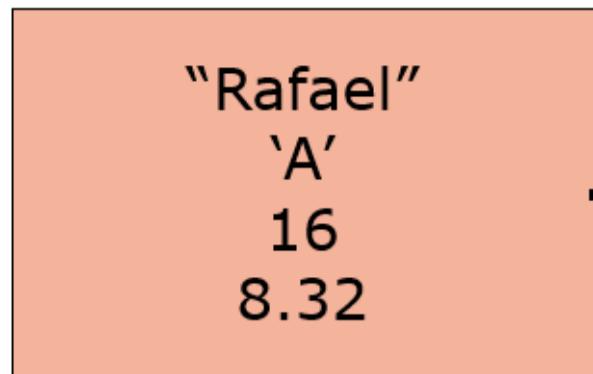
Ilustración 226: Se crean tres nodos

Este sería el código en C#:

```
//Crea tres nodos separados
Nodo primero = new Nodo("Rafael", 'A', 16, 8.32);
Nodo segundo = new Nodo("Moreno", 'P', 9, 2.9);
Nodo tercero = new Nodo("Sally", 'C', 2010, 7.18);
```

Luego debe conectarse el primer nodo con el segundo nodo:

primero



segundo

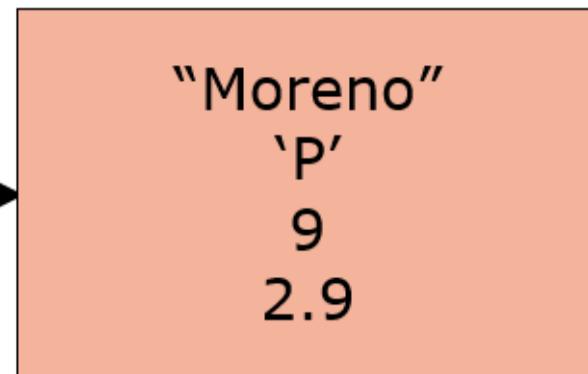


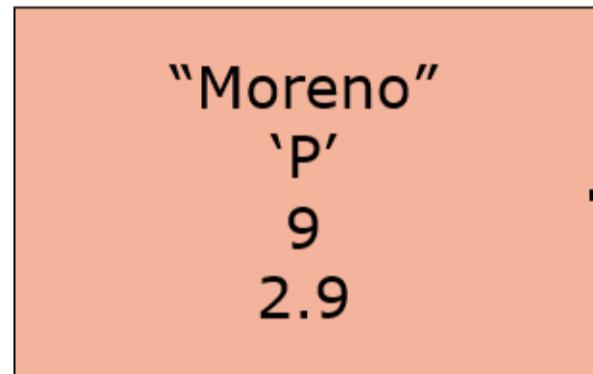
Ilustración 227: Conectar el nodo primero con el nodo segundo

Este sería el código en C#:

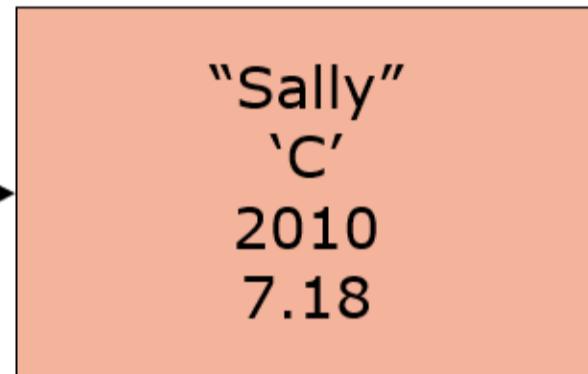
```
//Une el primer nodo con el segundo, creando una simple lista  
primero.Apuntador = segundo;
```

Luego debe conectarse el segundo nodo con el tercer nodo:

segundo



tercero



Este sería el código en C#:

```
//Une el segundo nodo con el tercero, aumentando la lista  
segundo.Apuntador = tercero;
```

Enlazamiento continuo

Con una sola variable declarada, se va armando la lista. En primer lugar, se hacen cambios a la clase Nodo, para que en el constructor se pueda enviar el apuntador.

H/002.cs

```
namespace Ejemplo {
    class Nodo {
        //Atributos propios
        public string Cadena { get; set; }
        public char Caracter { get; set; }
        public int Entero { get; set; }
        public double NumReal { get; set; }

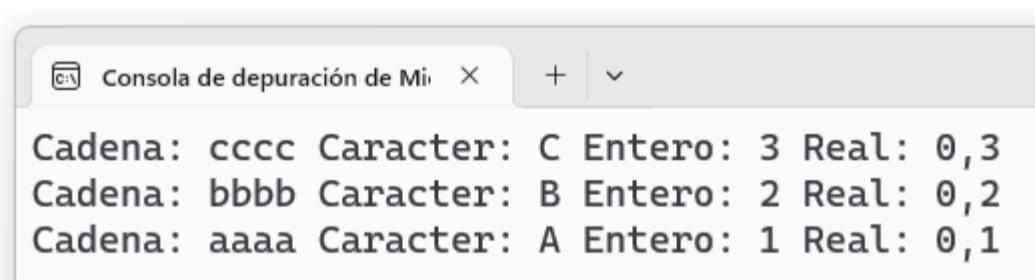
        //Apuntador para lista simplemente enlazada
        public Nodo Apuntador;

        //Constructor
        public Nodo(string Cadena, char Caracter, int Entero, double NumReal, Nodo Apuntador) {
            this.Cadena = Cadena;
            this.Caracter = Caracter;
            this.Entero = Entero;
            this.NumReal = NumReal;
            this.Apuntador = Apuntador;
        }

        //Imprime Contenido
        public void Imprime() {
            Console.WriteLine("Cadena: " + Cadena + " Caracter: " + Caracter.ToString());
            Console.WriteLine("Entero: " + Entero.ToString() + " Real: " + NumReal.ToString());
        }
    }

    class Program {
        static void Main() {
            //Crea la lista
            Nodo lista = new Nodo("aaaa", 'A', 1, 0.1, null);
            lista = new Nodo("bbbb", 'B', 2, 0.2, lista);
            lista = new Nodo("cccc", 'C', 3, 0.3, lista);

            //Imprime la lista
            lista.Imprime(); //Primer nodo
            lista.Apuntador.Imprime(); //Segundo nodo
            lista.Apuntador.Apuntador.Imprime(); //Tercer nodo
        }
    }
}
```



```
Cadena: cccc Caracter: C Entero: 3 Real: 0,3
Cadena: bbbb Caracter: B Entero: 2 Real: 0,2
Cadena: aaaa Caracter: A Entero: 1 Real: 0,1
```

Ilustración 242: Enlazamiento continuo

Recorriendo lista simplemente enlazada

Para recorrer una lista simplemente enlazada, se debe dejar una variable que sostenga la lista y una segunda es la que la recorre. Es importante eso porque sin la variable que sostiene toda la lista, a medida que va recorriendo nodo a nodo, ese estará borrando!

lista

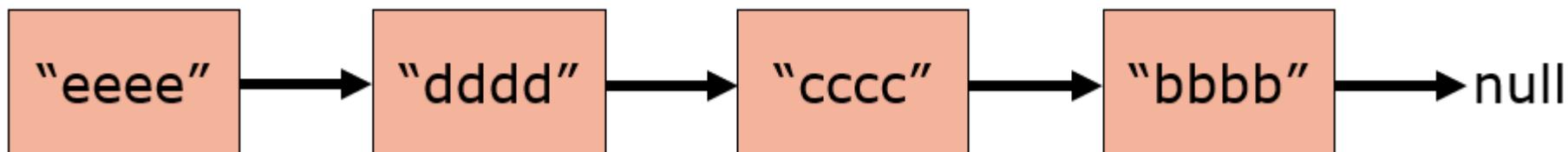


Ilustración 228: Variable "lista" sostiene la lista simplemente enlazada

H/003.cs

```
namespace Ejemplo {
    class Nodo {
        //Atributos propios
        public string Cadena { get; set; }
        public char Caracter { get; set; }
        public int Entero { get; set; }
        public double NumReal { get; set; }

        //Apuntador para lista simplemente enlazada
        public Nodo Apuntador;

        //Constructor
        public Nodo(string Cadena, char Caracter, int Entero, double NumReal, Nodo Apuntador) {
            this.Cadena = Cadena;
            this.Caracter = Caracter;
            this.Entero = Entero;
            this.NumReal = NumReal;
            this.Apuntador = Apuntador;
        }

        //Imprime Contenido
        public void Imprime() {
            Console.WriteLine("Cadena: " + Cadena + " Caracter: " + Caracter.ToString());
            Console.WriteLine("Entero: " + Entero.ToString() + " Real: " + NumReal.ToString());
        }
    }

    class Program {
        static void Main() {
            //Crea la lista
            Nodo lista = new Nodo("aaaa", 'A', 1, 0.1, null);
            lista = new Nodo("bbbb", 'B', 2, 0.2, lista);
            lista = new Nodo("cccc", 'C', 3, 0.3, lista);
            lista = new Nodo("dddd", 'D', 4, 0.4, lista);
            lista = new Nodo("eeee", 'E', 5, 0.5, lista);
            lista = new Nodo("ffff", 'F', 6, 0.6, lista);
            lista = new Nodo("gggg", 'G', 7, 0.7, lista);
            lista = new Nodo("hhhh", 'H', 8, 0.8, lista);
            lista = new Nodo("iiii", 'I', 9, 0.9, lista);

            //Pasea la lista, imprimiéndola
            Nodo pasea = lista;
            while(pasea != null) {
                pasea.Imprime();
                pasea = pasea.Apuntador;
            }
        }
    }
}
```

```
Cadena: iiii Caracter: I Entero: 9 Real: 0,9
Cadena: hhhh Caracter: H Entero: 8 Real: 0,8
Cadena: gggg Caracter: G Entero: 7 Real: 0,7
Cadena: ffff Caracter: F Entero: 6 Real: 0,6
Cadena: eeee Caracter: E Entero: 5 Real: 0,5
Cadena: dddd Caracter: D Entero: 4 Real: 0,4
Cadena: cccc Caracter: C Entero: 3 Real: 0,3
Cadena: bbbb Caracter: B Entero: 2 Real: 0,2
Cadena: aaaa Caracter: A Entero: 1 Real: 0,1
```

Ilustración 244: Recorriendo lista simplemente enlazada

`pasea = lista`

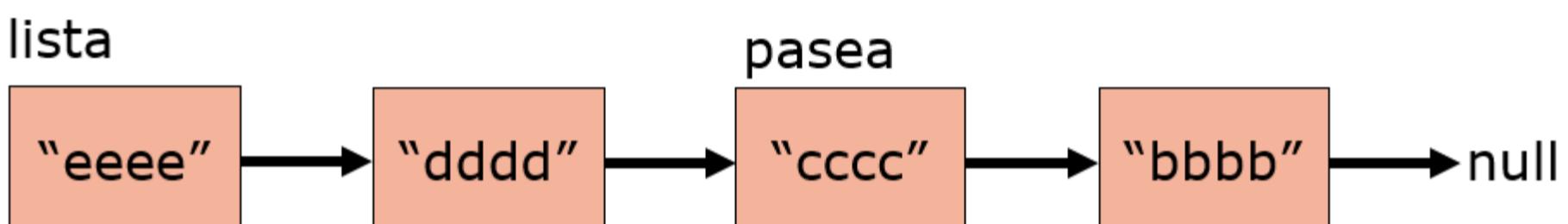
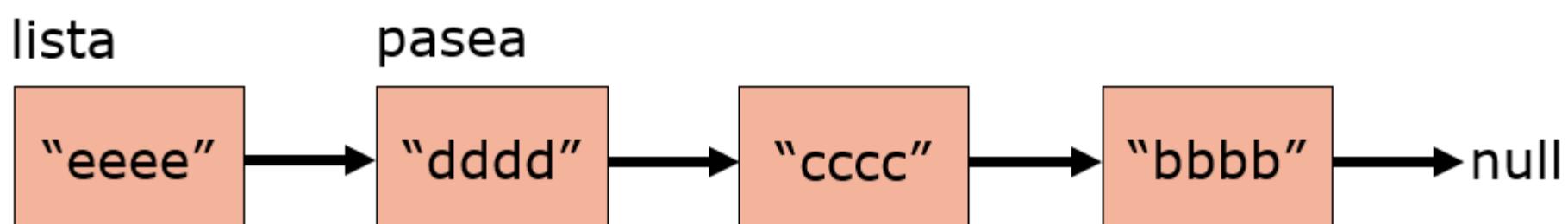
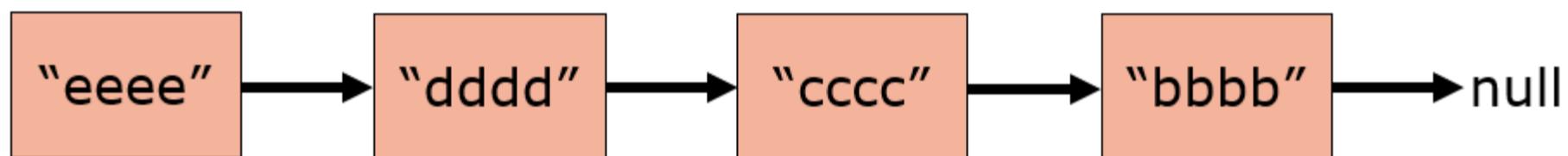


Ilustración 229: La variable "pasea" va de nodo en nodo. Con esta variable se muestra el contenido de cada nodo.

Recorriendo la lista usando un método

Se puede crear un método que recorra la lista enviándole por parámetro el apuntador de la lista. En la función se genera una copia de ese apuntador, así que puede recorrerla dentro de la función sin la preocupación de estar destruyendo la lista.

H/004.cs

```
namespace Ejemplo {
    class Nodo {
        //Atributos propios
        public string Cadena { get; set; }
        public char Caracter { get; set; }
        public int Entero { get; set; }
        public double NumReal { get; set; }

        //Apuntador para lista simplemente enlazada
        public Nodo Apuntador;

        //Constructor
        public Nodo(string Cadena, char Caracter, int Entero, double NumReal, Nodo Apuntador) {
            this.Cadena = Cadena;
            this.Caracter = Caracter;
            this.Entero = Entero;
            this.NumReal = NumReal;
            this.Apuntador = Apuntador;
        }

        //Imprime Contenido
        public void Imprime() {
            Console.WriteLine("Cadena: " + Cadena + " Caracter: " + Caracter.ToString());
            Console.WriteLine(" Entero: " + Entero.ToString() + " Real: " + NumReal.ToString());
        }
    }

    class Program {
        static void Main() {
            //Crea la lista
            Nodo lista = new Nodo("aaaa", 'A', 1, 0.1, null);
            lista = new Nodo("bbbb", 'B', 2, 0.2, lista);
            lista = new Nodo("cccc", 'C', 3, 0.3, lista);
            lista = new Nodo("dddd", 'D', 4, 0.4, lista);
            lista = new Nodo("eeee", 'E', 5, 0.5, lista);
            lista = new Nodo("ffff", 'F', 6, 0.6, lista);
            lista = new Nodo("gggg", 'G', 7, 0.7, lista);
            lista = new Nodo("hhhh", 'H', 8, 0.8, lista);
            lista = new Nodo("iiii", 'I', 9, 0.9, lista);

            //Pasea la lista, imprimiéndola
            Console.WriteLine("RECORRE PRIMERA VEZ");
            ImprimeLista(lista);

            Console.WriteLine("\r\nRECORRE SEGUNDA VEZ");
            ImprimeLista(lista);
        }

        static public void ImprimeLista(Nodo pasear) {
            while (pasear != null) {
                pasear.Imprime();
                pasear = pasear.Apuntador;
            }
        }
    }
}
```

```
Consola de depuración de Mi + v  
RECORRE PRIMERA VEZ  
Cadena: iiii Caracter: I Entero: 9 Real: 0,9  
Cadena: hhhh Caracter: H Entero: 8 Real: 0,8  
Cadena: gggg Caracter: G Entero: 7 Real: 0,7  
Cadena: ffff Caracter: F Entero: 6 Real: 0,6  
Cadena: eeee Caracter: E Entero: 5 Real: 0,5  
Cadena: dddd Caracter: D Entero: 4 Real: 0,4  
Cadena: cccc Caracter: C Entero: 3 Real: 0,3  
Cadena: bbbb Caracter: B Entero: 2 Real: 0,2  
Cadena: aaaa Caracter: A Entero: 1 Real: 0,1  
  
RECORRE SEGUNDA VEZ  
Cadena: iiii Caracter: I Entero: 9 Real: 0,9  
Cadena: hhhh Caracter: H Entero: 8 Real: 0,8  
Cadena: gggg Caracter: G Entero: 7 Real: 0,7  
Cadena: ffff Caracter: F Entero: 6 Real: 0,6  
Cadena: eeee Caracter: E Entero: 5 Real: 0,5  
Cadena: dddd Caracter: D Entero: 4 Real: 0,4  
Cadena: cccc Caracter: C Entero: 3 Real: 0,3  
Cadena: bbbb Caracter: B Entero: 2 Real: 0,2  
Cadena: aaaa Caracter: A Entero: 1 Real: 0,1
```

Ilustración 246: Recorriendo la lista usando un método

Tamaño de la lista

Una función puede retornar el tamaño de la lista recorriendo nodo a nodo.

H/005.cs

```
namespace Ejemplo {
    class Nodo {
        //Atributos propios
        public string Cadena { get; set; }
        public char Caracter { get; set; }
        public int Entero { get; set; }
        public double NumReal { get; set; }

        //Apuntador para lista simplemente enlazada
        public Nodo Apuntador;

        //Constructor
        public Nodo(string Cadena, char Caracter, int Entero, double NumReal, Nodo Apuntador) {
            this.Cadena = Cadena;
            this.Caracter = Caracter;
            this.Entero = Entero;
            this.NumReal = NumReal;
            this.Apuntador = Apuntador;
        }

        //Imprime Contenido
        public void Imprime() {
            Console.WriteLine("Cadena: " + Cadena + " Caracter: " + Caracter.ToString());
            Console.WriteLine(" Entero: " + Entero.ToString() + " Real: " + NumReal.ToString());
        }
    }

    class Program {
        static void Main() {
            //Crea la lista
            Nodo lista = new Nodo("aaaa", 'A', 1, 0.1, null);
            lista = new Nodo("bbbb", 'B', 2, 0.2, lista);
            lista = new Nodo("cccc", 'C', 3, 0.3, lista);
            lista = new Nodo("dddd", 'D', 4, 0.4, lista);
            lista = new Nodo("eeee", 'E', 5, 0.5, lista);
            lista = new Nodo("ffff", 'F', 6, 0.6, lista);
            lista = new Nodo("gggg", 'G', 7, 0.7, lista);
            lista = new Nodo("hhhh", 'H', 8, 0.8, lista);
            lista = new Nodo("iiii", 'I', 9, 0.9, lista);

            //Imprime el tamaño de la lista
            Console.WriteLine("Tamaño de la lista es: " + TamanoLista(lista));
        }

        //Imprime la lista
        static public void ImprimeLista(Nodo pasear) {
            while (pasear != null) {
                pasear.Imprime();
                pasear = pasear.Apuntador;
            }
        }

        //Retorna el tamaño de la lista
        static public int TamanoLista(Nodo pasear) {
            int tamano = 0;
            while (pasear != null) {
                tamano++;
                pasear = pasear.Apuntador;
            }
            return tamano;
        }
    }
}
```

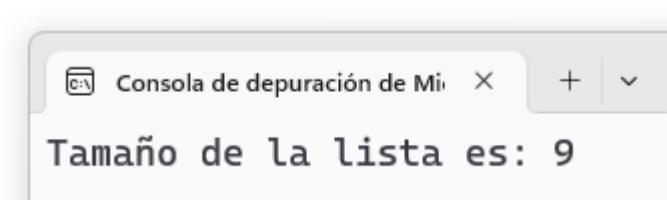


Ilustración 247: Tamaño de la lista

```

namespace Ejemplo {
    class Nodo {
        //Atributos propios
        public string Cadena { get; set; }
        public char Caracter { get; set; }
        public int Entero { get; set; }
        public double NumReal { get; set; }

        //Apuntador para lista simplemente enlazada
        public Nodo Apuntador;

        //Constructor
        public Nodo(string Cadena, char Caracter, int Entero, double NumReal, Nodo Apuntador) {
            this.Cadena = Cadena;
            this.Caracter = Caracter;
            this.Entero = Entero;
            this.NumReal = NumReal;
            this.Apuntador = Apuntador;
        }

        //Imprime Contenido
        public void Imprime() {
            Console.WriteLine("Cadena: " + Cadena + " Caracter: " + Caracter.ToString());
            Console.WriteLine(" Entero: " + Entero.ToString() + " Real: " + NumReal.ToString());
        }
    }

    class Program {
        static void Main() {
            //Crea la lista
            Nodo lista = new Nodo("aaaa", 'A', 1, 0.1, null);
            lista = new Nodo("bbbb", 'B', 2, 0.2, lista);
            lista = new Nodo("cccc", 'C', 3, 0.3, lista);
            lista = new Nodo("dddd", 'D', 4, 0.4, lista);
            lista = new Nodo("eeee", 'E', 5, 0.5, lista);
            lista = new Nodo("ffff", 'F', 6, 0.6, lista);
            lista = new Nodo("gggg", 'G', 7, 0.7, lista);
            lista = new Nodo("hhhh", 'H', 8, 0.8, lista);
            lista = new Nodo("iiii", 'I', 9, 0.9, lista);

            //Trae un determinado nodo
            Nodo particular = TraeNodo(lista, 2);
            particular.Imprime();
        }

        //Retornar nodo de determinada posición
        static public Nodo TraeNodo(Nodo pasear, int posicion) {
            int ubicacion = 0;
            while (pasear != null) {
                if (ubicacion == posicion) return pasear;
                pasear = pasear.Apuntador;
                ubicacion++;
            }
            return null;
        }
    }
}

```

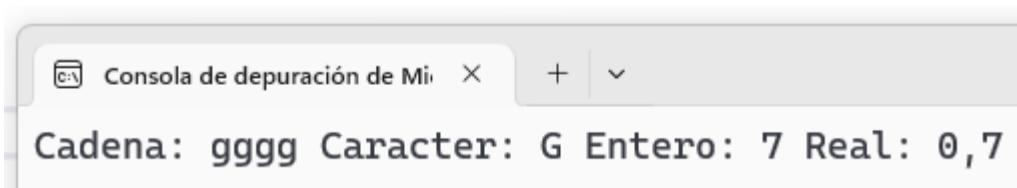
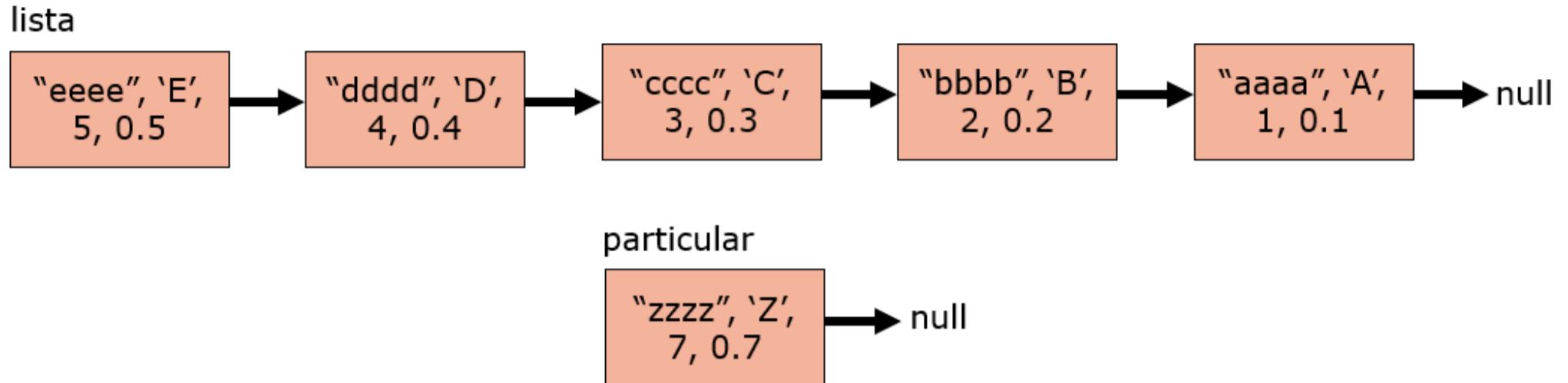


Ilustración 248: Traer un determinado nodo

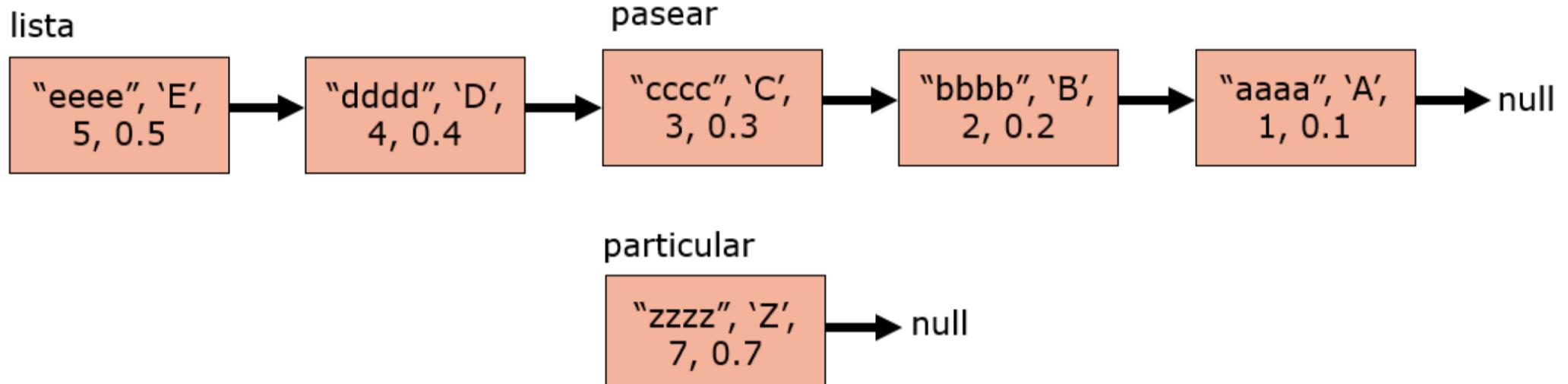
Adicionar un nodo en determinada posición

Para adicionar un nodo, se deben hacer varias operaciones:

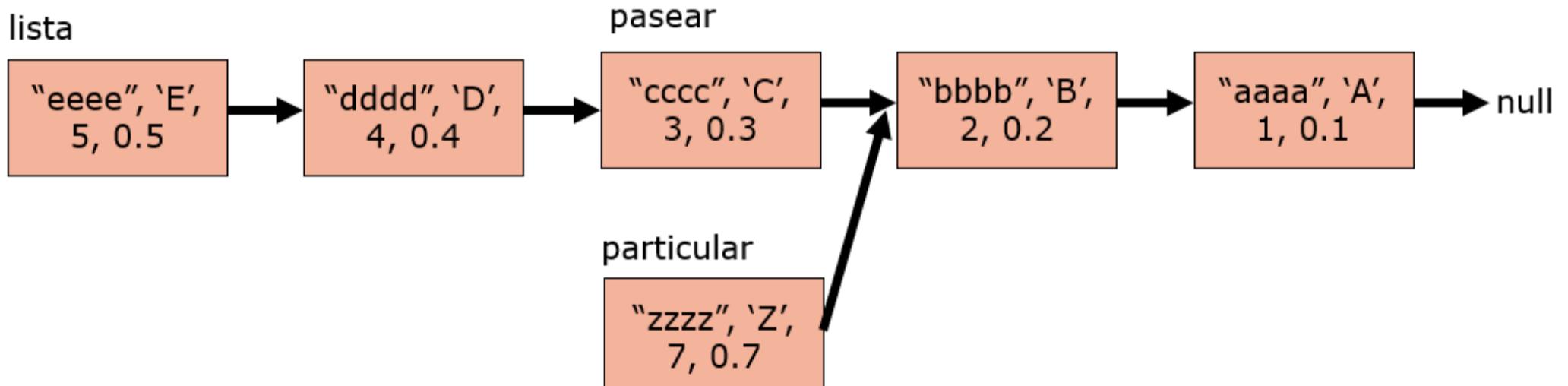
Paso 1: Lista existente y nodo nuevo a insertar



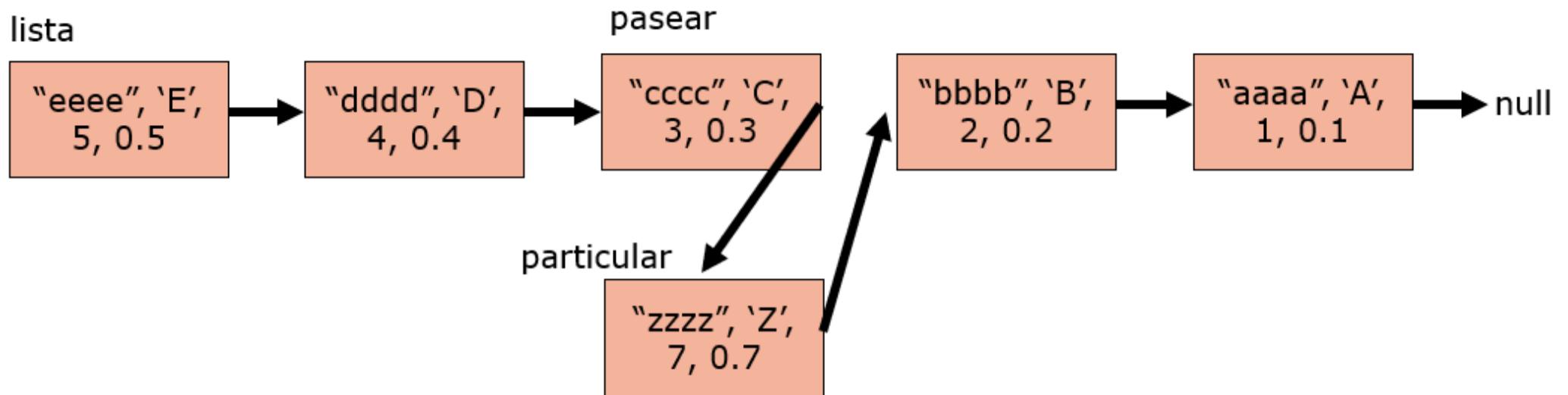
Paso 2: Ir hasta el punto de inserción



Paso 3: El nuevo nodo apunta a la posición particular de la lista



Paso 4: El nodo de la lista apunta al nuevo nodo



H/007.cs

```

namespace Ejemplo {
    class Nodo {
        //Atributos propios
        public string Cadena { get; set; }
        public char Caracter { get; set; }
        public int Entero { get; set; }
        public double NumReal { get; set; }

        //Apuntador para lista simplemente enlazada
        public Nodo Apuntador;

        //Constructor
        public Nodo(string Cadena, char Caracter, int Entero, double NumReal, Nodo Apuntador) {
            this.Cadena = Cadena;
            this.Caracter = Caracter;
            this.Entero = Entero;
            this.NumReal = NumReal;
            this.Apuntador = Apuntador;
        }

        //Imprime Contenido
        public void Imprime() {
            Console.WriteLine("Cadena: " + Cadena + " Caracter: " + Caracter.ToString());
            Console.WriteLine("Entero: " + Entero.ToString() + " Real: " + NumReal.ToString());
        }
    }

    class Program {
        static void Main() {
            //Crea la lista
            Nodo lista = new Nodo("aaaa", 'A', 1, 0.1, null);
            lista = new Nodo("bbbb", 'B', 2, 0.2, lista);
            lista = new Nodo("cccc", 'C', 3, 0.3, lista);
            lista = new Nodo("dddd", 'D', 4, 0.4, lista);
            lista = new Nodo("eeee", 'E', 5, 0.5, lista);

            //Añade un nodo en una determinada posición
            Nodo particular = new Nodo("zzzz", 'Z', 7, 0.7, null);
            lista = AdicionaNodo(particular, lista, 3);
            ImprimeLista(lista);
        }

        //Adiciona un nodo en determinada posición
        static public Nodo AdicionaNodo(Nodo nodo, Nodo lista, int posicion) {
            //Si es al inicio de la lista
            if (posicion == 0) {
                nodo.Apuntador = lista;
                return nodo;
            }

            //Si es en una ubicación intermedia
            int ubicacion = 0;
            Nodo pasear = lista;
            while (pasear != null) {
                if (ubicacion + 1 == posicion) {
                    nodo.Apuntador = pasear.Apuntador;
                    pasear.Apuntador = nodo;
                    return lista;
                }
                ubicacion++;
                pasear = pasear.Apuntador;
            }
        }
    }
}
  
```

```

        }
        pasear = pasear.Apuntador;
        ubicacion++;
    }

    //Si es al final de la lista
    pasear = lista;
    while (pasear.Apuntador != null) pasear = pasear.Apuntador;
    pasear.Apuntador = nodo;
    return lista;
}

//Imprime la lista
static public void ImprimeLista(Nodo pasear) {
    while (pasear != null) {
        pasear.Imprime();
        pasear = pasear.Apuntador;
    }
}
}

```

```

Cadena: eeee Caracter: E Entero: 5 Real: 0,5
Cadena: dddd Caracter: D Entero: 4 Real: 0,4
Cadena: cccc Caracter: C Entero: 3 Real: 0,3
Cadena: zzzz Caracter: Z Entero: 7 Real: 0,7
Cadena: bbbb Caracter: B Entero: 2 Real: 0,2
Cadena: aaaa Caracter: A Entero: 1 Real: 0,1

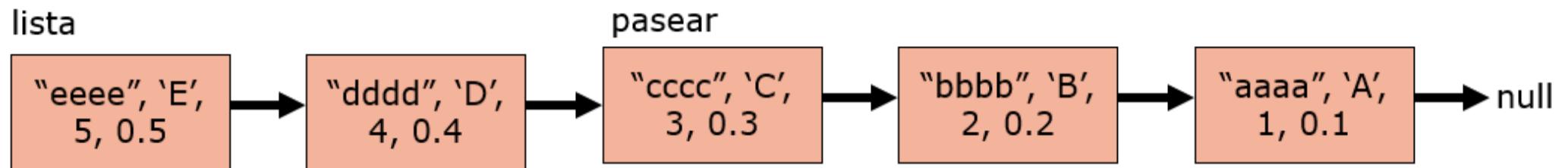
```

Ilustración 249: Adicionar un nodo en determinada posición

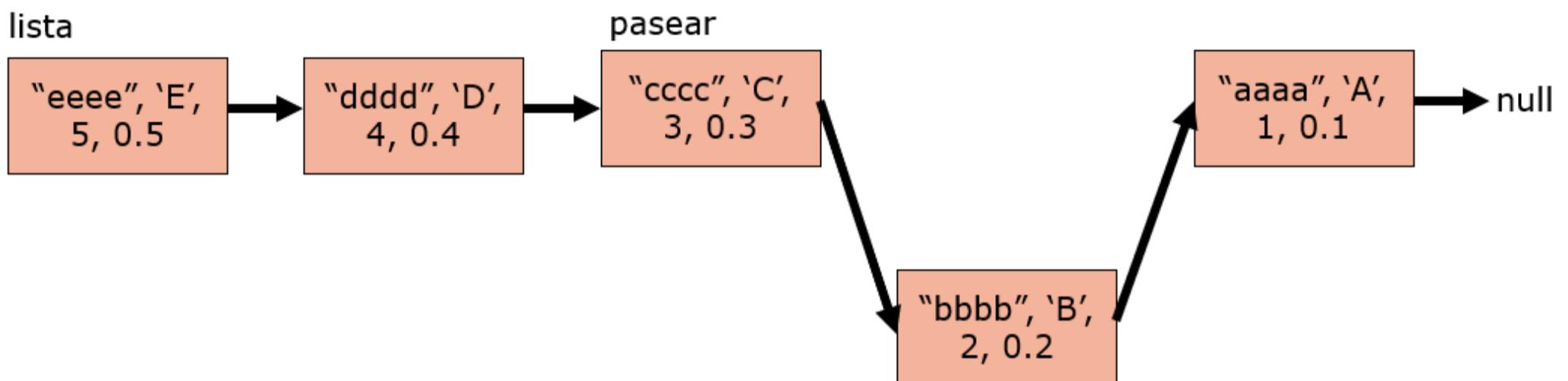
Borrar un nodo de una determinada posición

Para eliminar un nodo, se deben hacer varias operaciones:

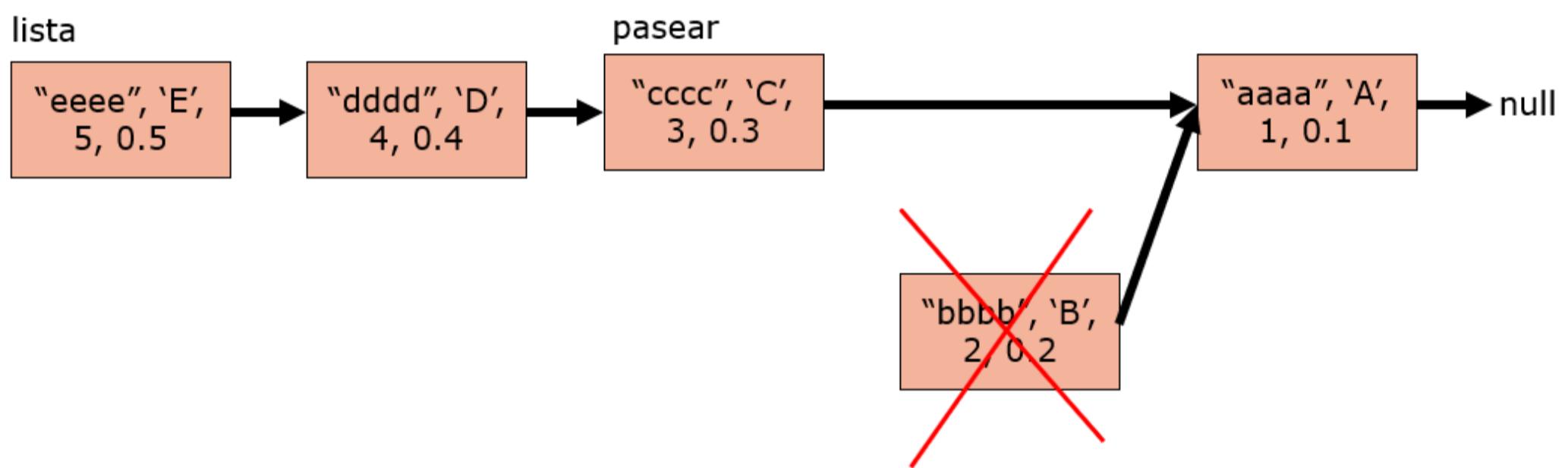
Paso 1: Lista existente e ir hasta el nodo al que se desee eliminar



Paso 2: El nodo que se va a eliminar



Paso 3: Se cambia el apuntador del nodo anterior al siguiente. El nodo sin conexión se borra automáticamente.



```

namespace Ejemplo {
    class Nodo {
        //Atributos propios
        public string Cadena { get; set; }
        public char Caracter { get; set; }
        public int Entero { get; set; }
        public double NumReal { get; set; }

        //Apuntador para lista simplemente enlazada
        public Nodo Apuntador;

        //Constructor
        public Nodo(string Cadena, char Caracter, int Entero, double NumReal, Nodo Apuntador) {
            this.Cadena = Cadena;
            this.Caracter = Caracter;
            this.Entero = Entero;
            this.NumReal = NumReal;
            this.Apuntador = Apuntador;
        }

        //Imprime Contenido
        public void Imprime() {
            Console.WriteLine("Cadena: " + Cadena + " Caracter: " + Caracter.ToString());
            Console.WriteLine("Entero: " + Entero.ToString() + " Real: " + NumReal.ToString());
        }
    }

    class Program {
        static void Main() {
            //Crea la lista
            Nodo lista = new Nodo("aaaa", 'A', 1, 0.1, null);
            lista = new Nodo("bbbb", 'B', 2, 0.2, lista);
            lista = new Nodo("cccc", 'C', 3, 0.3, lista);
            lista = new Nodo("dddd", 'D', 4, 0.4, lista);
            lista = new Nodo("eeee", 'E', 5, 0.5, lista);

            //Borra un nodo en una determinada posición
            lista = BorraNodo(lista, 3);
            ImprimeLista(lista);
        }

        //Borra nodo de una determinada posición
        static public Nodo BorraNodo(Nodo lista, int posicion) {
            //Si es al inicio de la lista
            if (posicion == 0) {
                lista = lista.Apuntador;
                return lista;
            }

            //Si es en una ubicación intermedia
            int ubicacion = 0;
            Nodo pasear = lista;
            while (pasear != null) {
                if (ubicacion + 1 == posicion) {
                    pasear.Apuntador = pasear.Apuntador.Apuntador;
                    return lista;
                }
                pasear = pasear.Apuntador;
                ubicacion++;
            }

            //Si es al final de la lista
            pasear = lista;
            while (pasear.Apuntador.Apuntador != null) pasear = pasear.Apuntador;
            pasear.Apuntador = null;
            return lista;
        }

        //Imprime la lista
        static public void ImprimeLista(Nodo pasear) {
            while (pasear != null) {
                pasear.Imprime();
                pasear = pasear.Apuntador;
            }
        }
    }
}

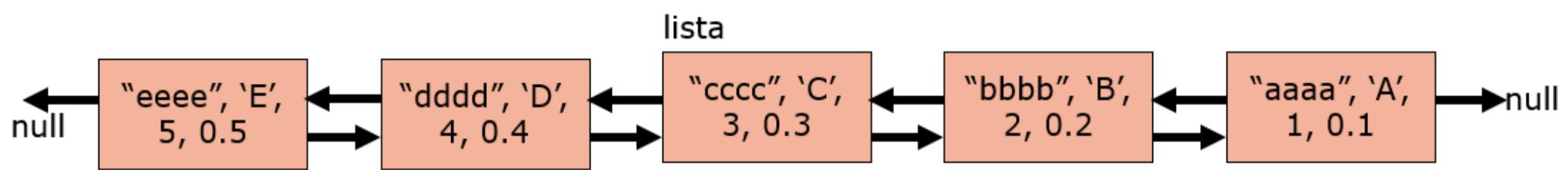
```

```
Cadena: eeee Caracter: E Entero: 5 Real: 0,5
Cadena: dddd Caracter: D Entero: 4 Real: 0,4
Cadena: cccc Caracter: C Entero: 3 Real: 0,3
Cadena: aaaa Caracter: A Entero: 1 Real: 0,1
```

Ilustración 250: Borrar un nodo de una determinada posición

La lista doblemente enlazada

Los nodos tienen doble conexión. Eso permite poder desplazare de izquierda a derecha, o de derecha a izquierda. La variable que sostiene la lista puede estar en cualquier nodo.



H/009.cs

```
namespace Ejemplo {
    class Nodo {
        //Atributos propios
        public string Cadena { get; set; }
        public char Caracter { get; set; }
        public int Entero { get; set; }
        public double NumReal { get; set; }

        //Apunadores para listas doblemente enlazadas
        public Nodo NodoIzq;
        public Nodo NodoDer;

        //Constructor
        public Nodo(string Cadena, char Caracter, int Entero, double NumReal, Nodo NodoDer) {
            this.Cadena = Cadena;
            this.Caracter = Caracter;
            this.Entero = Entero;
            this.NumReal = NumReal;
            NodoIzq = null;
            this.NodoDer = NodoDer;
            if (NodoDer != null) NodoDer.NodoIzq = this;
        }

        //Imprime Contenido
        public void Imprime() {
            Console.WriteLine("Cadena: " + Cadena + " Caracter: " + Caracter.ToString());
            Console.WriteLine("Entero: " + Entero.ToString() + " Real: " + NumReal.ToString());
        }
    }

    class Program {
        static void Main() {
            //Crea la lista
            Nodo lista = new Nodo("aaaa", 'A', 1, 0.1, null);
            lista = new Nodo("bbbb", 'B', 2, 0.2, lista);
            lista = new Nodo("cccc", 'C', 3, 0.3, lista);
            lista = new Nodo("ddd", 'D', 4, 0.4, lista);
            lista = new Nodo("eeee", 'E', 5, 0.5, lista);

            //Imprime la lista en ambos sentidos
            ImprimeIzquierdaDerecha(lista);
            ImprimeDerechaIzquierda(lista);
        }

        //Imprime la lista de izquierda a derecha
        static public void ImprimeIzquierdaDerecha(Nodo pasear) {
            Console.WriteLine("\r\nDe izquierda a derecha");

            //Debe ponerse en el primer nodo de la izquierda
            while (pasear.NodoIzq != null) {
                pasear = pasear.NodoIzq;
            }

            //Una vez en el primer nodo de la izquierda, entonces va
            //de izquierda a derecha imprimiendo
            while (pasear != null) {
                pasear.Imprime();
                pasear = pasear.NodoDer;
            }
        }

        //Imprime la lista de derecha a izquierda
    }
}
```

```

static public void ImprimeDerechaIzquierda(Nodo pasear) {
    Console.WriteLine("\r\nDe derecha a izquierda");

    //Debe ponerse en el primer nodo de la derecha
    while (pasear.NodoDer != null) {
        pasear = pasear.NodoDer;
    }

    //Una vez en el primer nodo de la derecha, entonces va
    //de derecha a izquierda imprimiendo
    while (pasear != null) {
        pasear.Imprime();
        pasear = pasear.NodoIzq;
    }
}
}

```

De izquierda a derecha

Cadena: eeee Caracter: E Entero: 5 Real: 0,5
Cadena: dddd Caracter: D Entero: 4 Real: 0,4
Cadena: cccc Caracter: C Entero: 3 Real: 0,3
Cadena: bbbb Caracter: B Entero: 2 Real: 0,2
Cadena: aaaa Caracter: A Entero: 1 Real: 0,1

De derecha a izquierda

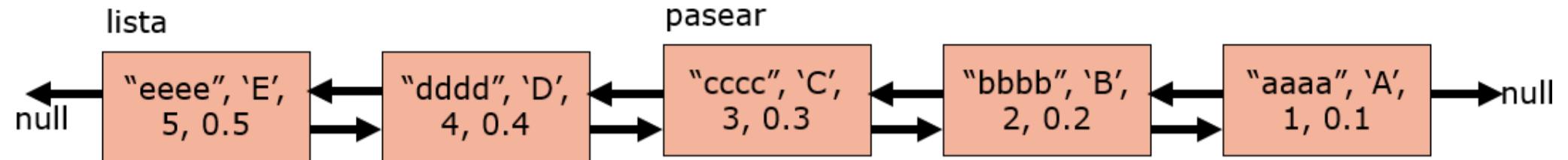
Cadena: aaaa Caracter: A Entero: 1 Real: 0,1
Cadena: bbbb Caracter: B Entero: 2 Real: 0,2
Cadena: cccc Caracter: C Entero: 3 Real: 0,3
Cadena: dddd Caracter: D Entero: 4 Real: 0,4
Cadena: eeee Caracter: E Entero: 5 Real: 0,5

Ilustración 251: La lista doblemente enlazada

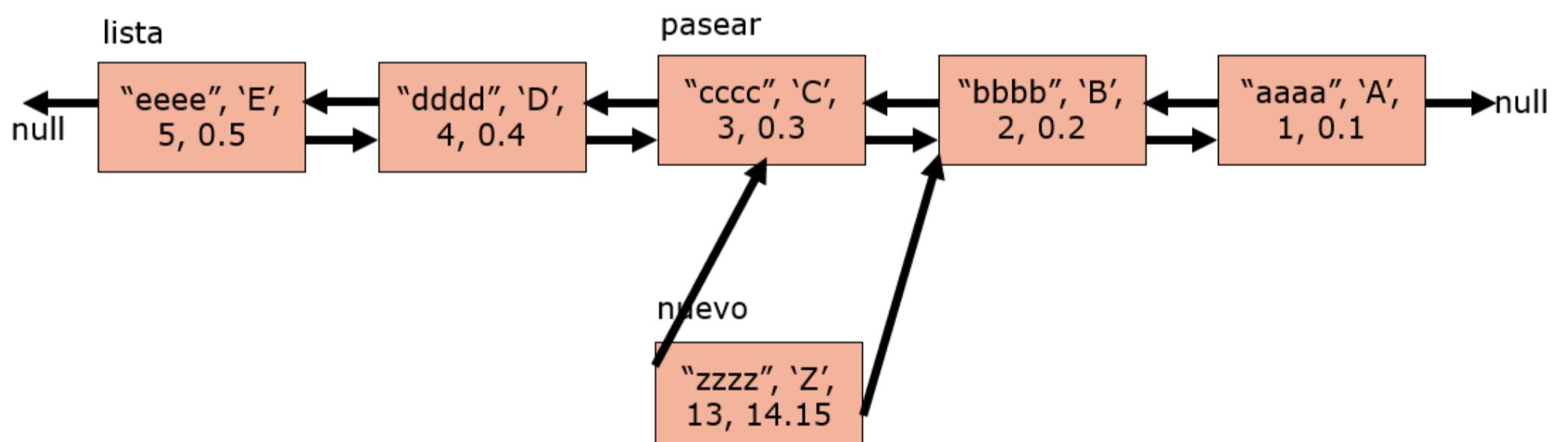
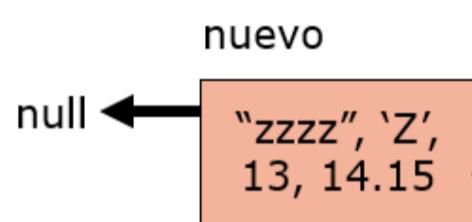
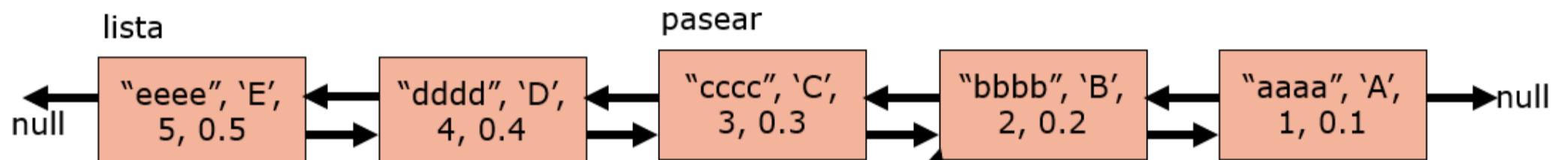
Adicionar un nodo en determinada posición

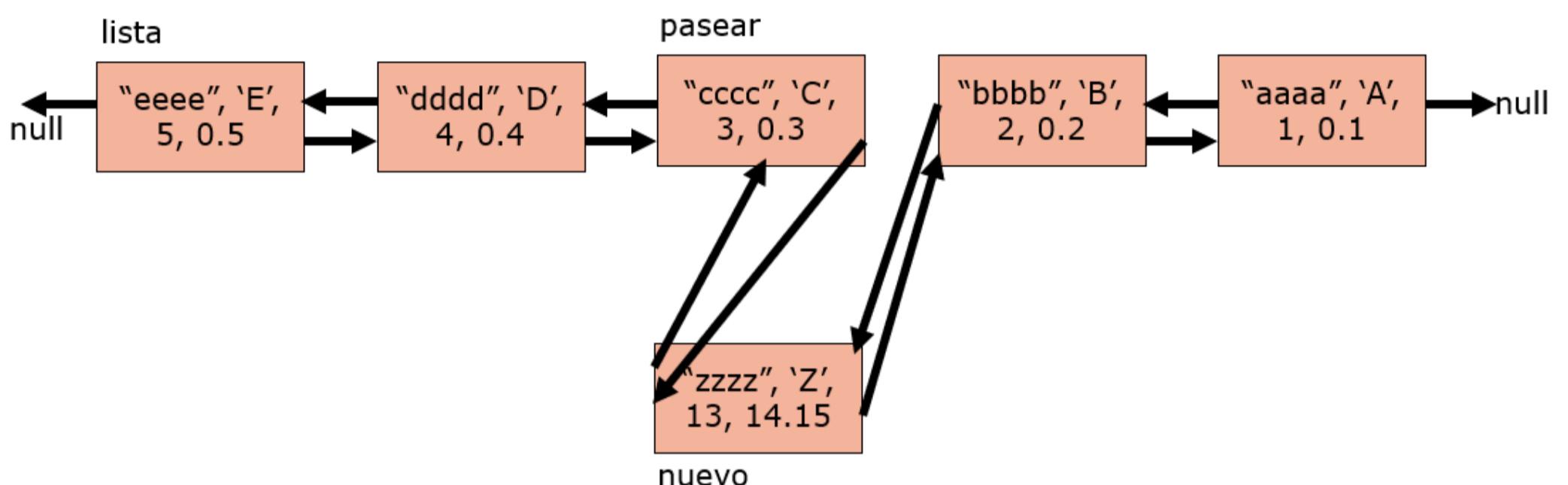
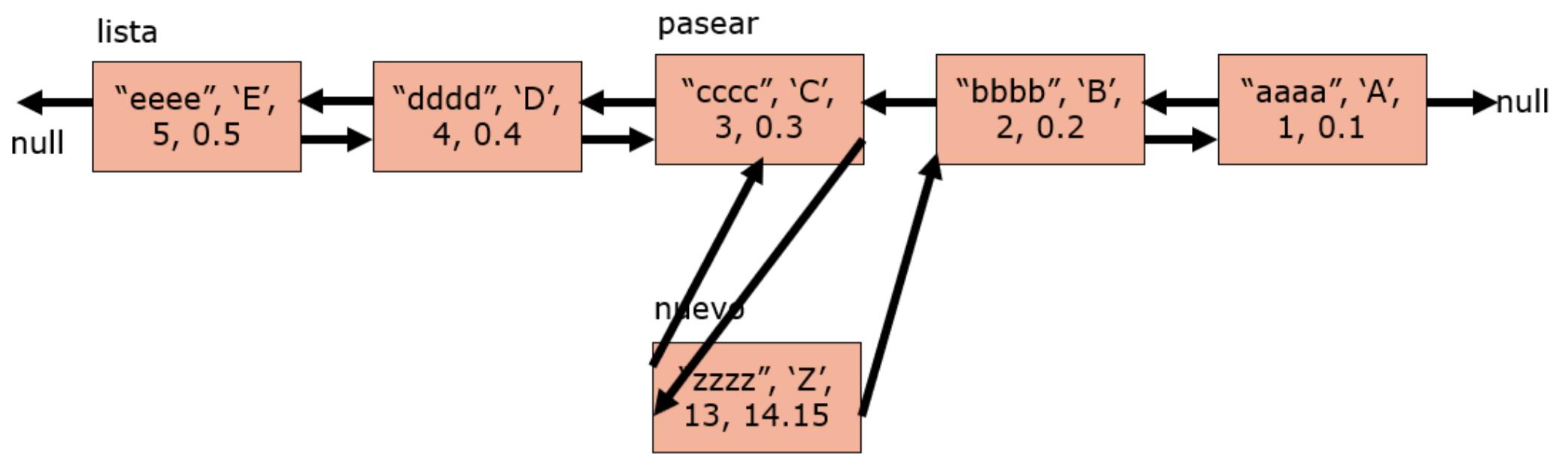
Para adicionar un nodo, se deben hacer varias operaciones:

Paso 1: Lista existente y nodo nuevo a insertar. Ir al punto de inserción.



Paso 2: Poner los enlaces





H/010.cs

```

namespace Ejemplo {
    class Nodo {
        //Atributos propios
        public string Cadena { get; set; }
        public char Caracter { get; set; }
        public int Entero { get; set; }
        public double NumReal { get; set; }

        //Apunadores para listas doblemente enlazadas
        public Nodo NodoIzq;
        public Nodo NodoDer;

        //Constructor
        public Nodo(string Cadena, char Caracter, int Entero, double NumReal, Nodo NodoDer) {
            this.Cadena = Cadena;
            this.Caracter = Caracter;
            this.Entero = Entero;
            this.NumReal = NumReal;
            NodoIzq = null;
            this.NodoDer = NodoDer;
            if (NodoDer != null) NodoDer.NodoIzq = this;
        }

        //Imprime Contenido
        public void Imprime() {
            Console.WriteLine("Cadena: " + Cadena + " Caracter: " + Caracter.ToString());
            Console.WriteLine(" Entero: " + Entero.ToString() + " Real: " + NumReal.ToString());
        }
    }

    class Program {
        static void Main() {
            //Crea la lista
            Nodo lista = new Nodo("aaaa", 'A', 1, 0.1, null);
            lista = new Nodo("bbbb", 'B', 2, 0.2, lista);
            lista = new Nodo("cccc", 'C', 3, 0.3, lista);
            lista = new Nodo("dddd", 'D', 4, 0.4, lista);
        }
    }
}
  
```

```

        lista = new Nodo("eeee", 'E', 5, 0.5, lista);

        //Imprime la lista en ambos sentidos
        ImprimeIzquierdaDerecha(lista);
        ImprimeDerechaIzquierda(lista);

        //Agrega un nodo a la lista doblemente enlazada
        Nodo nuevo = new Nodo("zzzz", 'Z', 13, 14.15, null);
        lista = AgregaNodo(nuevo, lista, 3);

        //Imprime la lista en ambos sentidos
        ImprimeIzquierdaDerecha(lista);
        ImprimeDerechaIzquierda(lista);
    }

    //Agrega un nodo a la lista doblemente enlazada
    static public Nodo AgregaNodo(Nodo nuevo, Nodo lista, int posicion) {
        //Debe asegurarse de ponerse en el primer nodo de la izquierda
        while (lista.NodoIzq != null) {
            lista = lista.NodoIzq;
        }

        //Si es al inicio de la lista
        if (posicion == 0) {
            nuevo.NodoDer = lista;
            lista.NodoIzq = nuevo;
            return nuevo;
        }

        //Si es en una ubicación intermedia
        int ubicacion = 0;
        Nodo pasear = lista;
        while (pasear != null) {
            if (ubicacion + 1 == posicion) {
                nuevo.NodoDer = pasear.NodoDer;
                pasear.NodoDer.NodoIzq = nuevo;
                pasear.NodoDer = nuevo;
                nuevo.NodoIzq = pasear;
                return lista;
            }
            pasear = pasear.NodoDer;
            ubicacion++;
        }

        //Si es al final de la lista
        pasear = lista;
        while (pasear.NodoDer != null) pasear = pasear.NodoDer;
        pasear.NodoDer = nuevo;
        nuevo.NodoIzq = pasear;
        return lista;
    }

    //Imprime la lista de izquierda a derecha
    static public void ImprimeIzquierdaDerecha(Nodo pasear) {
        Console.WriteLine("\r\nDe izquierda a derecha");

        //Debe ponerse en el primer nodo de la izquierda
        while (pasear.NodoIzq != null) {
            pasear = pasear.NodoIzq;
        }

        //Una vez en el primer nodo de la izquierda, entonces va
        //de izquierda a derecha imprimiendo
        while (pasear != null) {
            pasear.Imprime();
            pasear = pasear.NodoDer;
        }
    }

    //Imprime la lista de derecha a izquierda
    static public void ImprimeDerechaIzquierda(Nodo pasear) {
        Console.WriteLine("\r\nDe derecha a izquierda");

        //Debe ponerse en el primer nodo de la derecha
        while (pasear.NodoDer != null) {
            pasear = pasear.NodoDer;
        }

        //Una vez en el primer nodo de la derecha, entonces va

```

```
//de derecha a izquierda imprimiendo
while (pasear != null) {
    pasear.Imprime();
    pasear = pasear.NodoIzq;
}
}
```

Consola de depuración de Mi... X + ▾

De izquierda a derecha

Cadena: eeee Caracter: E Entero: 5 Real: 0,5
Cadena: dddd Caracter: D Entero: 4 Real: 0,4
Cadena: cccc Caracter: C Entero: 3 Real: 0,3
Cadena: bbbb Caracter: B Entero: 2 Real: 0,2
Cadena: aaaa Caracter: A Entero: 1 Real: 0,1

De derecha a izquierda

Cadena: aaaa Caracter: A Entero: 1 Real: 0,1
Cadena: bbbb Caracter: B Entero: 2 Real: 0,2
Cadena: cccc Caracter: C Entero: 3 Real: 0,3
Cadena: dddd Caracter: D Entero: 4 Real: 0,4
Cadena: eeee Caracter: E Entero: 5 Real: 0,5

De izquierda a derecha

Cadena: eeee Caracter: E Entero: 5 Real: 0,5
Cadena: dddd Caracter: D Entero: 4 Real: 0,4
Cadena: cccc Caracter: C Entero: 3 Real: 0,3
Cadena: zzzz Caracter: Z Entero: 13 Real: 14,15
Cadena: bbbb Caracter: B Entero: 2 Real: 0,2
Cadena: aaaa Caracter: A Entero: 1 Real: 0,1

De derecha a izquierda

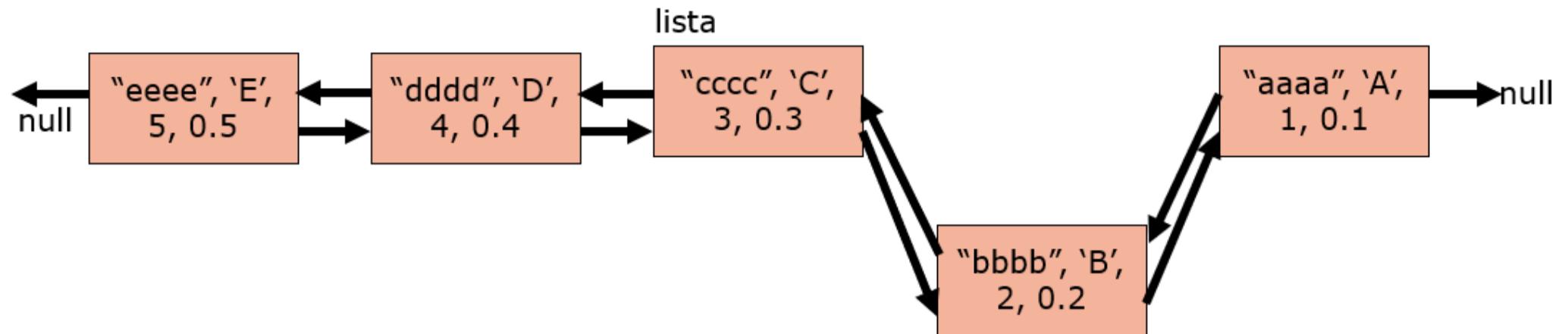
Cadena: aaaa Caracter: A Entero: 1 Real: 0,1
Cadena: bbbb Caracter: B Entero: 2 Real: 0,2
Cadena: zzzz Caracter: Z Entero: 13 Real: 14,15
Cadena: cccc Caracter: C Entero: 3 Real: 0,3
Cadena: dddd Caracter: D Entero: 4 Real: 0,4
Cadena: eeee Caracter: E Entero: 5 Real: 0,5

Ilustración 252: Adicionar un nodo en determinada posición

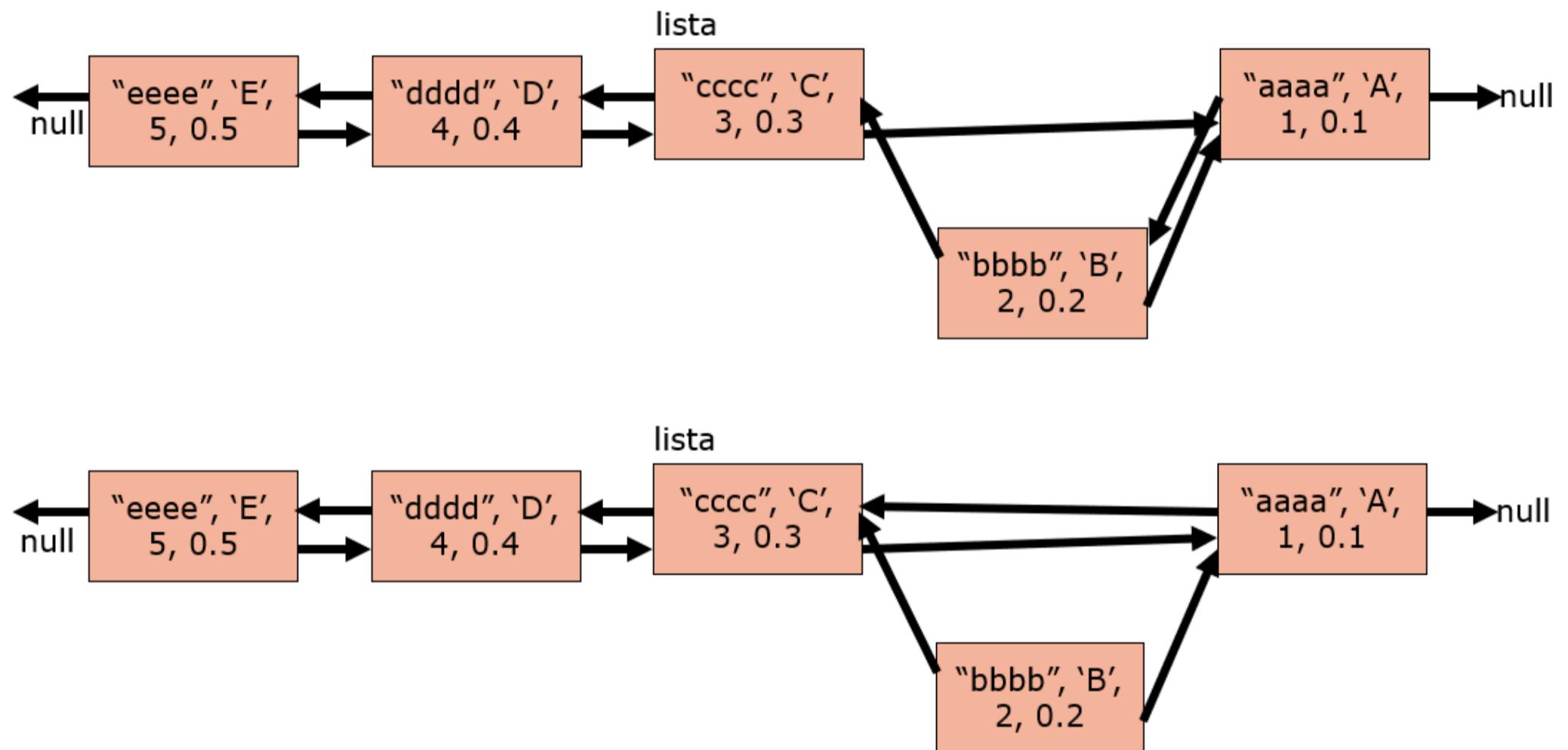
Borrar un nodo de una determinada posición

Para eliminar un nodo, se deben hacer varias operaciones:

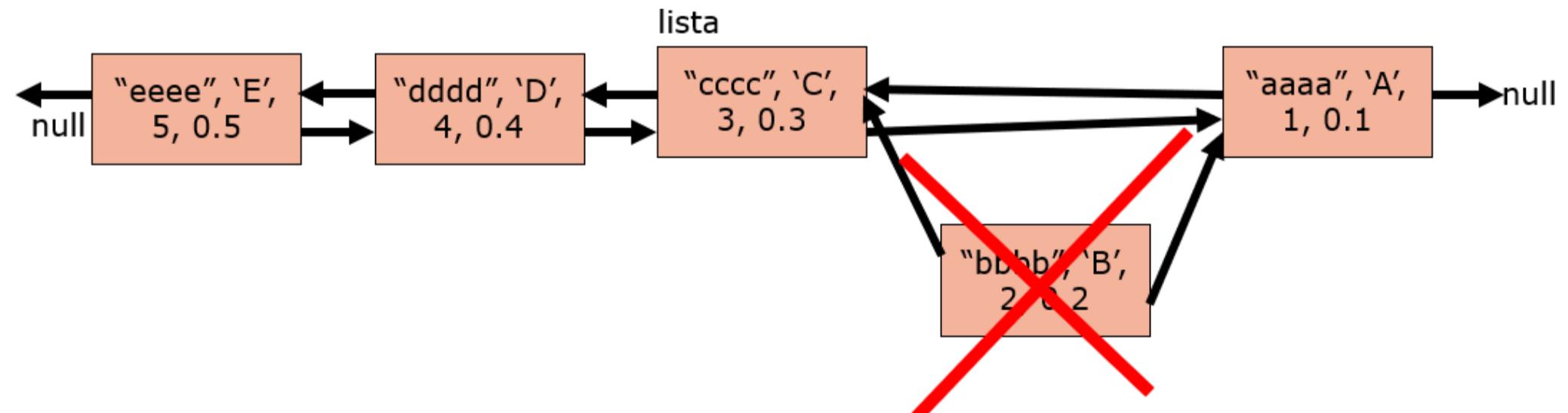
Paso 1: Lista existente e ir hasta el nodo al que se desea eliminar



Paso 2: Modificar los apuntadores



Paso 3: El nodo se destruye automáticamente al no tener quien lo sostenga



```

namespace Ejemplo {
    class Nodo {
        //Atributos propios
        public string Cadena { get; set; }
        public char Caracter { get; set; }
        public int Entero { get; set; }
        public double NumReal { get; set; }

        //Apunadores para listas doblemente enlazadas
        public Nodo NodoIzq;
        public Nodo NodoDer;

        //Constructor
        public Nodo(string Cadena, char Caracter, int Entero, double NumReal, Nodo NodoDer) {
            this.Cadena = Cadena;
            this.Caracter = Caracter;
            this.Entero = Entero;
            this.NumReal = NumReal;
            NodoIzq = null;
            this.NodoDer = NodoDer;
            if (NodoDer != null) NodoDer.NodoIzq = this;
        }

        //Imprime Contenido
        public void Imprime() {
            Console.WriteLine("Cadena: " + Cadena + " Caracter: " + Caracter.ToString());
            Console.WriteLine("Entero: " + Entero.ToString() + " Real: " + NumReal.ToString());
        }
    }

    class Program {
        static void Main() {
            //Crea la lista
            Nodo lista = new Nodo("aaaa", 'A', 1, 0.1, null);
            lista = new Nodo("bbbb", 'B', 2, 0.2, lista);
            lista = new Nodo("cccc", 'C', 3, 0.3, lista);
            lista = new Nodo("dddd", 'D', 4, 0.4, lista);
            lista = new Nodo("eeee", 'E', 5, 0.5, lista);

            //Imprime la lista en ambos sentidos
            ImprimeIzquierdaDerecha(lista);
            ImprimeDerechaIzquierda(lista);

            //Agrega un nodo a la lista doblemente enlazada
            lista = BorraNodo(lista, 3);

            //Imprime la lista en ambos sentidos
            ImprimeIzquierdaDerecha(lista);
            ImprimeDerechaIzquierda(lista);
        }

        //Borra un nodo de la lista doblemente enlazada
        static public Nodo BorraNodo(Nodo lista, int posicion) {
            //Debe asegurarse de ponerse en el primer nodo de la izquierda
            while (lista.NodoIzq != null) {
                lista = lista.NodoIzq;
            }

            //Si es al inicio de la lista
            if (posicion == 0) {
                lista = lista.NodoDer;
                lista.NodoIzq = null;
                return lista;
            }

            //Si es en una ubicación intermedia
            int ubicacion = 0;
            Nodo pasear = lista;
            while (pasear != null) {
                if (ubicacion+1 == posicion) {
                    pasear.NodoDer = pasear.NodoDer.NodoDer;
                    if (pasear.NodoDer != null) pasear.NodoDer.NodoIzq = pasear;
                    return lista;
                }
                pasear = pasear.NodoDer;
                ubicacion++;
            }
        }
    }
}

```

```

    //Si es al final de la lista
    pasear = lista;
    while (pasear.NodoDer.NodoDer != null) pasear = pasear.NodoDer;
    pasear.NodoDer = null;
    return lista;
}

//Imprime la lista de izquierda a derecha
static public void ImprimeIzquierdaDerecha(Nodo pasear) {
    Console.WriteLine("\r\nDe izquierda a derecha");

    //Debe ponerse en el primer nodo de la izquierda
    while (pasear.NodoIzq != null) {
        pasear = pasear.NodoIzq;
    }

    //Una vez en el primer nodo de la izquierda, entonces va
    //de izquierda a derecha imprimiendo
    while (pasear != null) {
        pasear.Imprime();
        pasear = pasear.NodoDer;
    }
}

//Imprime la lista de derecha a izquierda
static public void ImprimeDerechaIzquierda(Nodo pasear) {
    Console.WriteLine("\r\nDe derecha a izquierda");

    //Debe ponerse en el primer nodo de la derecha
    while (pasear.NodoDer != null) {
        pasear = pasear.NodoDer;
    }

    //Una vez en el primer nodo de la derecha, entonces va
    //de derecha a izquierda imprimiendo
    while (pasear != null) {
        pasear.Imprime();
        pasear = pasear.NodoIzq;
    }
}
}

```

De izquierda a derecha

Cadena: eeee Caracter: E Entero: 5 Real: 0,5
Cadena: dddd Caracter: D Entero: 4 Real: 0,4
Cadena: cccc Caracter: C Entero: 3 Real: 0,3
Cadena: bbbb Caracter: B Entero: 2 Real: 0,2
Cadena: aaaa Caracter: A Entero: 1 Real: 0,1

De derecha a izquierda

Cadena: aaaa Caracter: A Entero: 1 Real: 0,1
Cadena: bbbb Caracter: B Entero: 2 Real: 0,2
Cadena: cccc Caracter: C Entero: 3 Real: 0,3
Cadena: dddd Caracter: D Entero: 4 Real: 0,4
Cadena: eeee Caracter: E Entero: 5 Real: 0,5

De izquierda a derecha

Cadena: eeee Caracter: E Entero: 5 Real: 0,5
Cadena: dddd Caracter: D Entero: 4 Real: 0,4
Cadena: cccc Caracter: C Entero: 3 Real: 0,3
Cadena: aaaa Caracter: A Entero: 1 Real: 0,1

De derecha a izquierda

Cadena: aaaa Caracter: A Entero: 1 Real: 0,1
Cadena: cccc Caracter: C Entero: 3 Real: 0,3
Cadena: dddd Caracter: D Entero: 4 Real: 0,4
Cadena: eeee Caracter: E Entero: 5 Real: 0,5

Árbol binario

Primer ejemplo

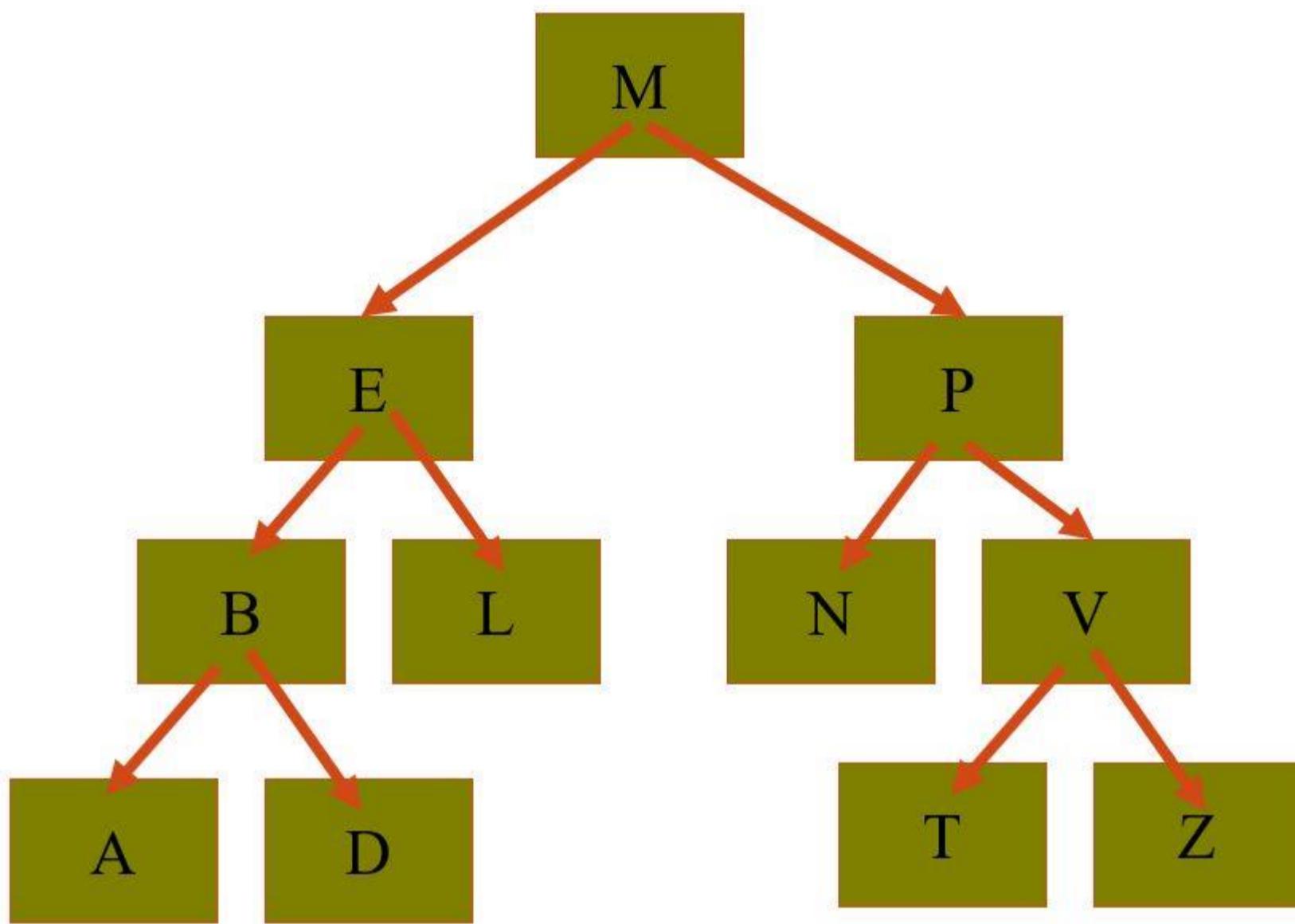


Ilustración 230: Ejemplo de un árbol binario

```
Consola de depuración de Mi... X + ▾
Recorrido preOrden (raiz, izquierdo, derecho)
M, E, B, A, D, L, P, N, V, T, Z,
Recorrido inOrden (izquierdo, raiz, derecho)
A, B, D, E, L, M, N, P, T, V, Z,
Recorrido postOrden (izquierdo, derecho, raiz)
A, D, B, L, E, N, T, Z, V, P, M,
```

Ilustración 255: Recorrido de un árbol binario

H/012.cs

```
//Recorrido de un árbol binario
namespace Ejemplo {
    class Nodo {
        public char Letra { get; set; }
        public Nodo Izquierda; //Apuntador
        public Nodo Derecha; //Apuntador

        //Constructor
        public Nodo(char Letra) {
            this.Letra = Letra;
        }
    }

    class Program {
        public static void Main() {
            //Crea el árbol
            Nodo Arbol = new Nodo('M');
            Arbol.Izquierda = new Nodo('E');
            Arbol.Derecha = new Nodo('P');
            Arbol.Izquierda.Izquierda = new Nodo('B');
```

```

Arbol.Izquierda.Derecha = new Nodo('L');
Arbol.Izquierda.Izquierda = new Nodo('A');
Arbol.Izquierda.Izquierda.Derecha = new Nodo('D');
Arbol.Derecha.Izquierda = new Nodo('N');
Arbol.Derecha.Derecha = new Nodo('V');
Arbol.Derecha.Derecha.Izquierda = new Nodo('T');
Arbol.Derecha.Derecha.Derecha = new Nodo('Z');

//Recorridos
Console.WriteLine("Recorrido preOrden (raiz, izquierdo, derecho)");
PreOrden(Arbol);

Console.WriteLine("\n\nRecorrido inOrden (izquierdo, raiz, derecho)");
InOrden(Arbol);

Console.WriteLine("\n\nRecorrido postOrden (izquierdo, derecho, raiz)");
PostOrden(Arbol);
}

static void PreOrden(Nodo Arbol) {
    if (Arbol != null) {
        Console.Write(Arbol.Letra + ", ");
        PreOrden(Arbol.Izquierda);
        PreOrden(Arbol.Derecha);
    }
}
static void InOrden(Nodo Arbol) {
    if (Arbol != null) {
        InOrden(Arbol.Izquierda);
        Console.Write(Arbol.Letra + ", ");
        InOrden(Arbol.Derecha);
    }
}
static void PostOrden(Nodo Arbol) {
    if (Arbol != null) {
        PostOrden(Arbol.Izquierda);
        PostOrden(Arbol.Derecha);
        Console.Write(Arbol.Letra + ", ");
    }
}
}

```

Segundo ejemplo

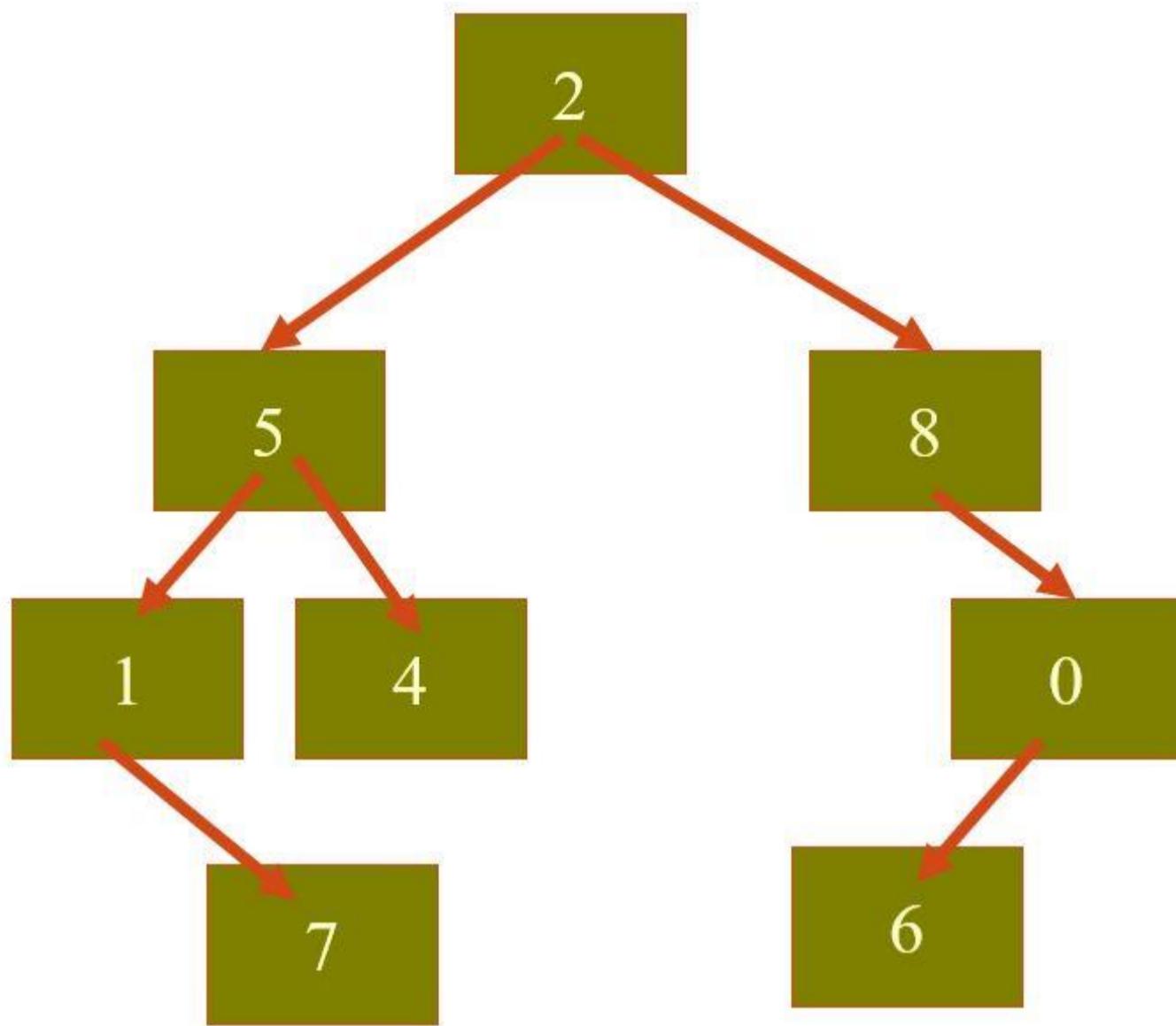


Ilustración 231: Ejemplo de árbol binario

```
Consola de depuración de Mi... X + ▾
Recorrido preOrden (raiz, izquierdo, derecho)
2, 5, 1, 7, 4, 8, 0, 6,
Recorrido inOrden (izquierdo, raiz, derecho)
1, 7, 5, 4, 2, 8, 6, 0,
Recorrido postOrden (izquierdo, derecho, raiz)
7, 1, 4, 5, 6, 0, 8, 2,
```

Ilustración 257: Recorrido de un árbol binario

H/013.cs

```
//Recorridos de un árbol binario
namespace Ejemplo {
    class Nodo {
        public char Letra { get; set; }
        public Nodo Izquierda; //Apuntador
        public Nodo Derecha; //Apuntador

        //Constructor
        public Nodo(char Letra) {
            this.Letra = Letra;
        }
    }

    class Program {
        static void Main(string[] args) {
            //Crea el árbol
            Nodo Arbol = new Nodo('2');
            Arbol.Izquierda = new Nodo('5');
            Arbol.Derecha = new Nodo('8');
            Arbol.Izquierda.Izquierda = new Nodo('1');
            Arbol.Izquierda.Derecha = new Nodo('4');
            Arbol.Izquierda.Izquierda.Derecha = new Nodo('7');
            Arbol.Derecha.Derecha = new Nodo('0');
            Arbol.Derecha.Derecha.Izquierda = new Nodo('6');
```

```

//Recorridos
Console.WriteLine("Recorrido preOrden (raiz, izquierdo, derecho)");
PreOrden(Arbol);

Console.WriteLine("\n\nRecorrido inOrden (izquierdo, raiz, derecho)");
InOrden(Arbol);

Console.WriteLine("\n\nRecorrido postOrden (izquierdo, derecho, raiz)");
PostOrden(Arbol);
}

static void PreOrden(Nodo Arbol) {
    if (Arbol != null) {
        Console.Write(Arbol.Letra + ", ");
        PreOrden(Arbol.Izquierda);
        PreOrden(Arbol.Derecha);
    }
}

static void InOrden(Nodo Arbol) {
    if (Arbol != null) {
        InOrden(Arbol.Izquierda);
        Console.Write(Arbol.Letra + ", ");
        InOrden(Arbol.Derecha);
    }
}

static void PostOrden(Nodo Arbol) {
    if (Arbol != null) {
        PostOrden(Arbol.Izquierda);
        PostOrden(Arbol.Derecha);
        Console.Write(Arbol.Letra + ", ");
    }
}
}

```

Recorrido iterativo (no recursivo)

El siguiente árbol binario será recorrido en forma iterativa. Se requiere para ello el uso de Pilas

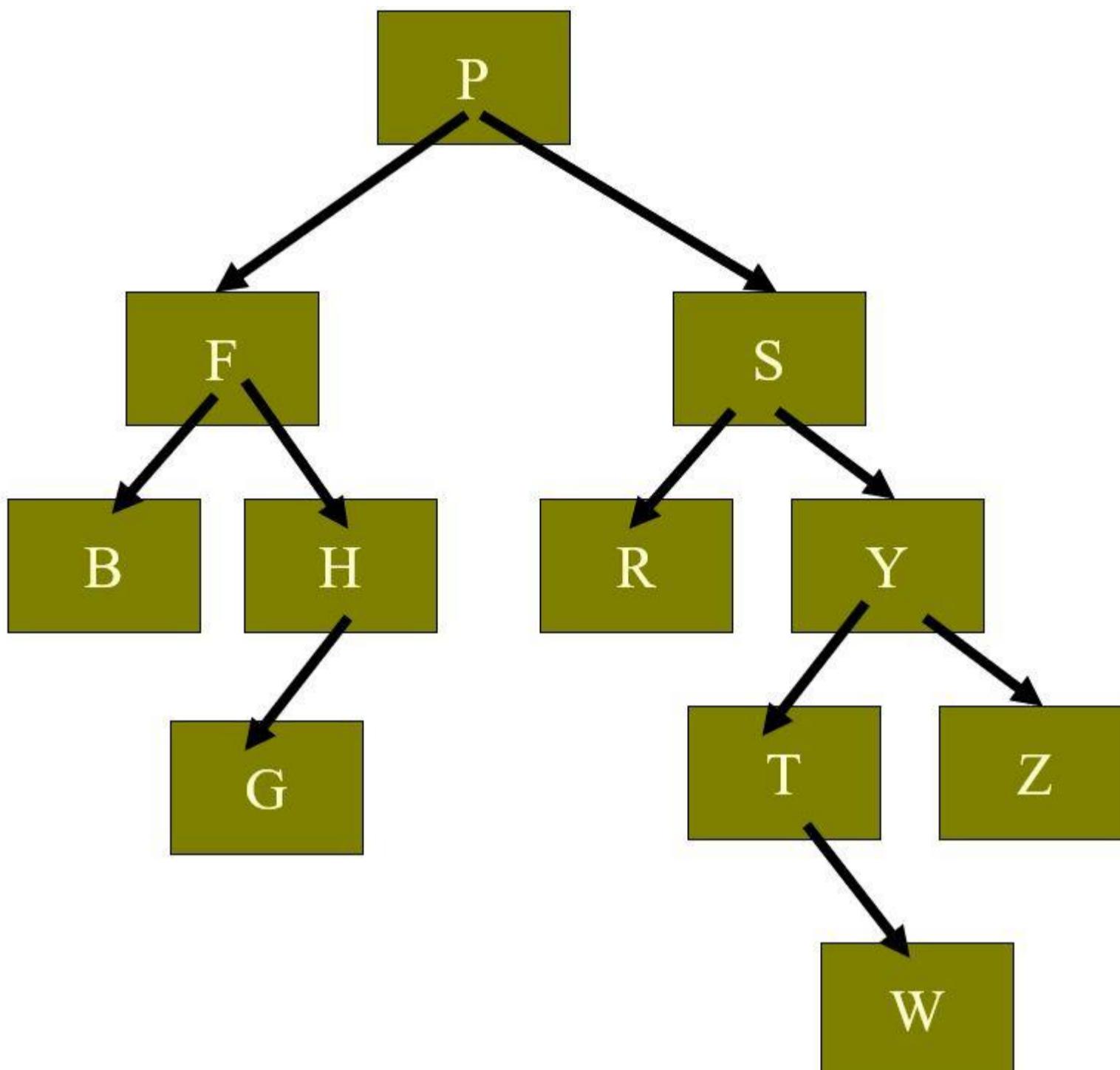


Ilustración 232: Árbol binario

```
Consola de depuración de Mi... + ▾
Recorrido Iterativo
P, S, Y, Z, T, W, R, F, H, G, B,
```

Ilustración 259: Recorrido iterativo (no recursivo)

H/014.cs

```
//Recorrido (no recursivo) de un árbol binario
namespace Ejemplo {
    //Nodo de un árbol binario
    class Nodo {
        public char Letra { get; set; }
        public Nodo Izquierda; //Apuntador
        public Nodo Derecha; //Apuntador

        //Constructor
        public Nodo(char Letra) {
            this.Letra = Letra;
        }
    }

    //Nodo de una pila para recorrer iterativamente un árbol binario
    class NodoPila {
        public NodoPila Flecha;
        public Nodo Raiz;
        public NodoPila(Nodo Raiz, NodoPila Flecha) {
            this.Raiz = Raiz;
            this.Flecha = Flecha;
        }
    }
}
```

```

}

class Program {
    static void Main(string[] args) {
        //Crea el árbol
        Nodo Arbol = new Nodo('P');
        Arbol.Izquierda = new Nodo('F');
        Arbol.Derecha = new Nodo('S');
        Arbol.Izquierda.Izquierda = new Nodo('B');
        Arbol.Izquierda.Derecha = new Nodo('H');
        Arbol.Izquierda.Derecha.Izquierda = new Nodo('G');
        Arbol.Derecha.Izquierda = new Nodo('R');
        Arbol.Derecha.Derecha = new Nodo('Y');
        Arbol.Derecha.Derecha.Izquierda = new Nodo('T');
        Arbol.Derecha.Derecha.Derecha = new Nodo('Z');
        Arbol.Derecha.Derecha.Izquierda.Derecha = new Nodo('W');

        //Recorrido iterativo
        Console.WriteLine("Recorrido Iterativo");
        Iterativo(Arbol);
    }

    public static void Iterativo(Nodo arbol) {
        //Usa una pila para guardar
        NodoPila inicio = new NodoPila(arbol, null);
        do {
            Nodo tmp = inicio.Raiz; //Una variable tmp para ver el nodo del árbol
            Console.Write(tmp.Letra + ", ");
            inicio = inicio.Flecha; //Se quita un elemento de la pila
            if (tmp.Izquierda != null) inicio = new NodoPila(tmp.Izquierda, inicio); //Si el nodo
de árbol tiene un hijo a la izquierda entonces agrega este a la pila
            if (tmp.Derecha != null) inicio = new NodoPila(tmp.Derecha, inicio); //Si el nodo de
árbol tiene un hijo a la derecha entonces agrega este a la pila
        } while (inicio != null); //Hasta que se vacíe la pila
    }
}

```

```
//Crear un árbol binario al azar
namespace Ejemplo {
    //Nodo de un árbol binario
    class Nodo {
        public int Numero { get; set; }
        public Nodo Izquierda; //Apuntador
        public Nodo Derecha; //Apuntador

        //Constructor
        public Nodo(int Numero) {
            this.Numero = Numero;
        }
    }

    class Program {
        static void Main(string[] args) {
            Random azar = new Random();
            Nodo Arbol = new Nodo(azar.Next(100));

            //Crea el árbol binario
            for (int cont = 1; cont <= 10; cont++)
                AzarNodoArbol(azar, Arbol);

            //Recorridos
            Console.WriteLine("\n\nRecorrido preOrden (raiz, izquierdo, derecho)");
            preOrden(Arbol);

            Console.WriteLine("\n\nRecorrido inOrden (izquierdo, raiz, derecho)");
            inOrden(Arbol);

            Console.WriteLine("\n\nRecorrido postOrden (izquierdo, derecho, raiz)");
            postOrden(Arbol);
        }

        //Pone un nodo en una posición al azar
        static void AzarNodoArbol(Random azar, Nodo Raiz) {
            //Por debajo de 0.5 pone una rama a la izquierda
            if (azar.NextDouble() < 0.5) {
                if (Raiz.Izquierda == null)
                    Raiz.Izquierda = new Nodo(azar.Next(100));
                else
                    AzarNodoArbol(azar, Raiz.Izquierda);
            }
            else {
                if (Raiz.Derecha == null)
                    Raiz.Derecha = new Nodo(azar.Next(100));
                else
                    AzarNodoArbol(azar, Raiz.Derecha);
            }
        }

        static void inOrden(Nodo Arbol) {
            if (Arbol != null) {
                inOrden(Arbol.Izquierda);
                Console.Write(Arbol.Numero + ", ");
                inOrden(Arbol.Derecha);
            }
        }

        static void preOrden(Nodo Arbol) {
            if (Arbol != null) {
                Console.Write(Arbol.Numero + ", ");
                preOrden(Arbol.Izquierda);
                preOrden(Arbol.Derecha);
            }
        }

        static void postOrden(Nodo Arbol) {
            if (Arbol != null) {
                postOrden(Arbol.Izquierda);
                postOrden(Arbol.Derecha);
                Console.Write(Arbol.Numero + ", ");
            }
        }
    }
}
```

```
Recorrido preOrden (raiz, izquierdo, derecho)
3, 49, 44, 54, 57, 78, 66, 49, 80, 72, 82,
```

```
Recorrido inOrden (izquierdo, raiz, derecho)
54, 44, 49, 3, 57, 66, 49, 72, 80, 78, 82,
```

```
Recorrido postOrden (izquierdo, derecho, raiz)
54, 44, 49, 72, 80, 49, 66, 82, 78, 57, 3,
```

Ilustración 260: Generar árboles binarios al azar

Ordenamiento usando un árbol binario

A medida que se van agregando nodos a un árbol binario, los acomoda de tal forma que al leerlo en InOrden aparece ordenado el contenido.

H/016.cs

```
//Ordenar con un árbol binario
namespace Ejemplo {
    //Nodo de un árbol binario
    class Nodo {
        public int Numero { get; set; }
        public Nodo Izquierda; //Apuntador
        public Nodo Derecha; //Apuntador

        //Constructor
        public Nodo(int Numero) {
            this.Numero = Numero;
        }
    }

    class Program {
        static void Main(string[] args) {
            //Va agregando nodo a nodo y los va ordenando
            Nodo Arbol = new Nodo(27);
            AgregaNodo(7, Arbol);
            AgregaNodo(17, Arbol);
            AgregaNodo(2, Arbol);
            AgregaNodo(5, Arbol);
            AgregaNodo(19, Arbol);
            AgregaNodo(15, Arbol);
            AgregaNodo(9, Arbol);
            AgregaNodo(10, Arbol);
            AgregaNodo(-1, Arbol);
            AgregaNodo(18, Arbol);
            AgregaNodo(3, Arbol);

            //Al leer en inorder el arbol, los datos salen ordenados
            Console.WriteLine("\n\nRecorrido InOrden (izquierdo, raiz, derecho)");
            InOrden(Arbol);
        }

        static void AgregaNodo(int Valor, Nodo Raiz) {
            if (Valor <= Raiz.Numero) {
                if (Raiz.Izquierda == null)
                    Raiz.Izquierda = new Nodo(Valor);
                else
                    AgregaNodo(Valor, Raiz.Izquierda);
            }
            else {
                if (Raiz.Derecha == null)
                    Raiz.Derecha = new Nodo(Valor);
                else
                    AgregaNodo(Valor, Raiz.Derecha);
            }
        }

        static void InOrden(Nodo Arbol) {
            if (Arbol != null) {
                InOrden(Arbol.Izquierda);
                Console.WriteLine(Arbol.Numero + ", ");
                InOrden(Arbol.Derecha);
            }
        }
    }
}
```

The screenshot shows a terminal window titled "Consola de depuración de Mi". The window contains the following text:

```
Recorrido InOrden (izquierdo, raiz, derecho)
-1,
2,
3,
5,
7,
9,
10,
15,
17,
18,
19,
27,
```

Ilustración 261: Ordenamiento usando un árbol binario

```
//Ordenar, buscar en árbol binario ordenado, número de nodos y altura del árbol
namespace Ejemplo {
    class Nodo {
        public int Numero { get; set; }
        public Nodo Izquierda;
        public Nodo Derecha;

        public Nodo(int Numero) {
            this.Numero = Numero;
            this.Izquierda = null;
            this.Derecha = null;
        }
    }

    class Program {
        public static void Main() {

            Nodo Arbol = new Nodo(27);
            AgregaNodo(7, Arbol);
            AgregaNodo(17, Arbol);
            AgregaNodo(2, Arbol);
            AgregaNodo(5, Arbol);
            AgregaNodo(19, Arbol);
            AgregaNodo(15, Arbol);
            AgregaNodo(9, Arbol);
            AgregaNodo(10, Arbol);
            AgregaNodo(-1, Arbol);
            AgregaNodo(18, Arbol);
            AgregaNodo(3, Arbol);

            //Al leer en inorder el arbol, los datos salen ordenados
            Console.WriteLine("Valores ordenados");
            InOrden(Arbol);

            //Ahora a buscar un determinado valor
            Console.Write("\r\n\r\nBusca el valor: 185...");
            bool encontrar = BuscaArbol(Arbol, 185);
            if (encontrar)
                Console.WriteLine(" Valor encontrado");
            else
                Console.WriteLine(" Valor NO encontrado");

            //Busca valor en el árbol
            Console.Write("\r\nBusca el valor: 19...");
            Nodo nodoEncuentra = Buscanodo(Arbol, 19);
            if (nodoEncuentra != null)
                Console.WriteLine(" Valor encontrado");
            else
                Console.WriteLine(" Valor no encontrado");

            //Contar los nodos
            Console.WriteLine("\r\nTotal nodos: " + CuentaNodosArbol(Arbol));

            //Altura del árbol
            Console.WriteLine("\nAltura del árbol: " + AlturaArbol(Arbol));
        }

        public static void AgregaNodo(int Valor, Nodo Raiz) {
            if (Valor <= Raiz.Numero) {
                if (Raiz.Izquierda == null)
                    Raiz.Izquierda = new Nodo(Valor);
                else
                    AgregaNodo(Valor, Raiz.Izquierda);
            }
            else {
                if (Raiz.Derecha == null)
                    Raiz.Derecha = new Nodo(Valor);
                else
                    AgregaNodo(Valor, Raiz.Derecha);
            }
        }

        //Recorrido del árbol en InOrden
        static void InOrden(Nodo Arbol) {
            if (Arbol != null) {

```

```

        InOrden(Arbol.Izquierda);
        Console.Write(Arbol.Numero + ", ");
        InOrden(Arbol.Derecha);
    }

//Retorna true si encuentra el valor en el árbol binario
static bool BuscaArbol(Nodo Arbol, int valor) {
    if (Arbol != null) {
        if (Arbol.Numero == valor) return true;
        bool encuentraI = BuscaArbol(Arbol.Izquierda, valor);
        bool encuentraD = BuscaArbol(Arbol.Derecha, valor);
        if (encuentraI || encuentraD) return true;
    }
    return false;
}

//Retorna el Nodo donde se encuentra el valor buscado
static Nodo Buscanodo(Nodo Raiz, int valor) {
    if (valor == Raiz.Numero) return Raiz;
    if (valor < Raiz.Numero && Raiz.Izquierda != null) return Buscanodo(Raiz.Izquierda, valor);
    if (valor > Raiz.Numero && Raiz.Derecha != null) return Buscanodo(Raiz.Derecha, valor);
    return null;
}

//Cuenta los nodos de un árbol
static int CuentaNodosArbol(Nodo Arbol) {
    if (Arbol == null) return 0;
    int contarI = CuentaNodosArbol(Arbol.Izquierda);
    int contarD = CuentaNodosArbol(Arbol.Derecha);
    return contarI + contarD + 1;
}

//Calcula la altura de un árbol
static int AlturaArbol(Nodo Arbol) {
    if (Arbol == null) return 0;
    int alturaI = AlturaArbol(Arbol.Izquierda);
    int alturaD = AlturaArbol(Arbol.Derecha);
    if (alturaI > alturaD) return alturaI + 1;
    return alturaD + 1;
}
}

```

Consola de depuración de Mi... X + ▾

Valores ordenados
-1, 2, 3, 5, 7, 9, 10, 15, 17, 18, 19, 27,

Busca el valor: 185... Valor NO encontrado

Busca el valor: 19... Valor encontrado

Total nodos: 12

Altura del árbol: 6

Ilustración 262: Buscar en árbol binario ordenado, número de nodos y altura del árbol

Dibujar un árbol binario

En <http://viz-js.com/> se encuentra este servicio:

H/018.cs

```
//Dibujar el árbol en http://viz-js.com/
namespace Ejemplo {
    class Nodo {
        public char Letra { get; set; }
        public Nodo Izquierda;
        public Nodo Derecha;

        public Nodo(char Letra) {
            this.Letra = Letra;
            this.Izquierda = null;
            this.Derecha = null;
        }
    }

    class Program {
        public static void Main() {

            //Crea el árbol
            Nodo Arbol = new Nodo('P');
            Arbol.Izquierda = new Nodo('F');
            Arbol.Derecha = new Nodo('S');
            Arbol.Izquierda.Izquierda = new Nodo('B');
            Arbol.Izquierda.Derecha = new Nodo('H');
            Arbol.Izquierda.Derecha.Izquierda = new Nodo('G');
            Arbol.Derecha.Izquierda = new Nodo('R');
            Arbol.Derecha.Derecha = new Nodo('Y');
            Arbol.Derecha.Derecha.Izquierda = new Nodo('T');
            Arbol.Derecha.Derecha.Derecha = new Nodo('Z');
            Arbol.Derecha.Derecha.Izquierda.Derecha = new Nodo('W');

            //Probarlo en: http://viz-js.com
            Console.WriteLine("digraph testgraph{");
            Dibujar(Arbol);
            Console.WriteLine("}");
        }

        static void Dibujar(Nodo Arbol) {
            if (Arbol != null) {
                if (Arbol.Izquierda != null) {
                    Console.WriteLine(Arbol.Letra + "->" + Arbol.Izquierda.Letra);
                    Dibujar(Arbol.Izquierda);
                }
                if (Arbol.Derecha != null) {
                    Console.WriteLine(Arbol.Letra + "->" + Arbol.Derecha.Letra);
                    Dibujar(Arbol.Derecha);
                }
            }
        }
    }
}
```

```
digraph testgraph{
P->F
F->B
F->H
H->G
P->S
S->R
S->Y
Y->T
T->W
Y->Z
}
```

Ilustración 263: Dibujar un árbol binario

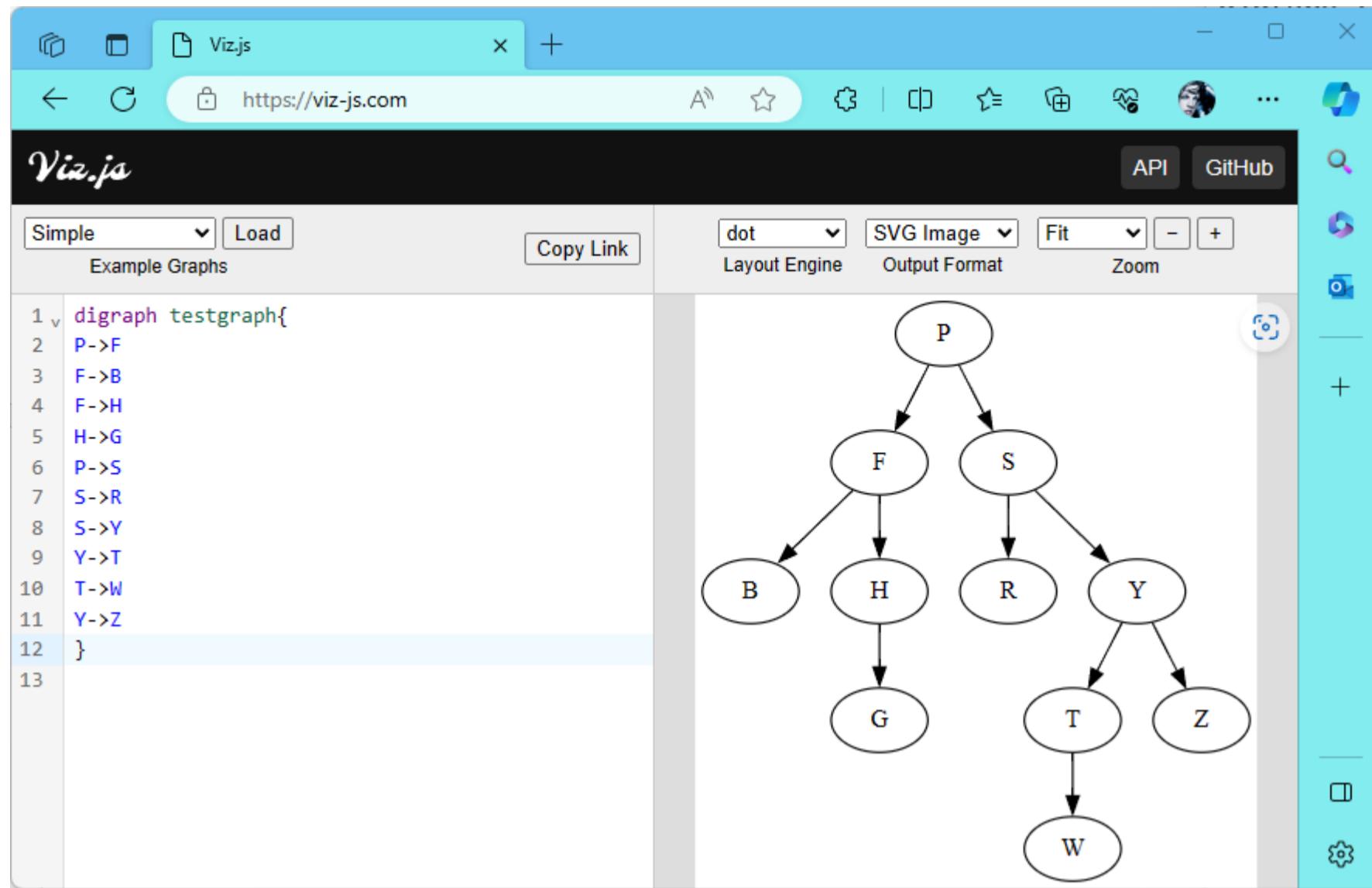


Ilustración 264: Dibujar un árbol binario

Recorrer un árbol binario por niveles

Se hace uso de una lista para almacenar dos datos de cada nodo del árbol: el nodo y su altura. Luego se recorre varias veces esa lista, mostrando los nodos de cada determinada altura.

H/019.cs

```
//Recorrido por niveles de un árbol binario
namespace Ejemplo {
    class Nodo {
        public char Letra { get; set; }
        public Nodo Izquierda; //Apuntador
        public Nodo Derecha; //Apuntador

        //Constructor
        public Nodo(char Letra) {
            this.Letra = Letra;
        }
    }

    class NodosNivel {
        public int Altura { get; set; }
        public Nodo nodo;

        public NodosNivel(int Altura, Nodo nodo) {
            this.Altura = Altura;
            this.nodo = nodo;
        }
    }

    class Program {
        static void Main(string[] args) {
            List<NodosNivel> niveles = new List<NodosNivel>();

            //Crea el árbol
            Nodo Arbol = new Nodo('P');
            Arbol.Izquierda = new Nodo('F');
            Arbol.Derecha = new Nodo('S');
            Arbol.Izquierda.Izquierda = new Nodo('B');
            Arbol.Izquierda.Derecha = new Nodo('H');
            Arbol.Izquierda.Derecha.Izquierda = new Nodo('G');
            Arbol.Derecha.Izquierda = new Nodo('R');
            Arbol.Derecha.Derecha = new Nodo('Y');
            Arbol.Derecha.Derecha.Izquierda = new Nodo('T');
            Arbol.Derecha.Derecha.Derecha = new Nodo('Z');
            Arbol.Derecha.Derecha.Izquierda.Derecha = new Nodo('W');

            //Recorrido por niveles
            Console.WriteLine("Recorrido por niveles");

            //Arma la lista con la información de Nodo y Altura
            ArmaLista(niveles, Arbol, 0);

            //Una vez armada la lista entonces la explora usando como llave la altura
            bool ExisteNivel;
            int Altura = 0;
            do {
                ExisteNivel = false;

                //Muestra los nodos de esa altura en particular
                for(int cont=0; cont<niveles.Count; cont++)
                    if (niveles[cont].Altura == Altura) {
                        Console.Write(niveles[cont].nodo.Letra + " -- ");
                        ExisteNivel = true;
                    }

                //Salta al siguiente nivel
                Console.WriteLine(" ");
                Altura++;
            } while (ExisteNivel);
        }

        //Arma la lista con la información del nodo y su altura
        public static void ArmaLista(List<NodosNivel> niveles, Nodo arbol, int altura) {
            niveles.Add(new NodosNivel(altura, arbol));
            if (arbol.Izquierda != null) ArmaLista(niveles, arbol.Izquierda, altura + 1);
            if (arbol.Derecha != null) ArmaLista(niveles, arbol.Derecha, altura + 1);
        }
    }
}
```

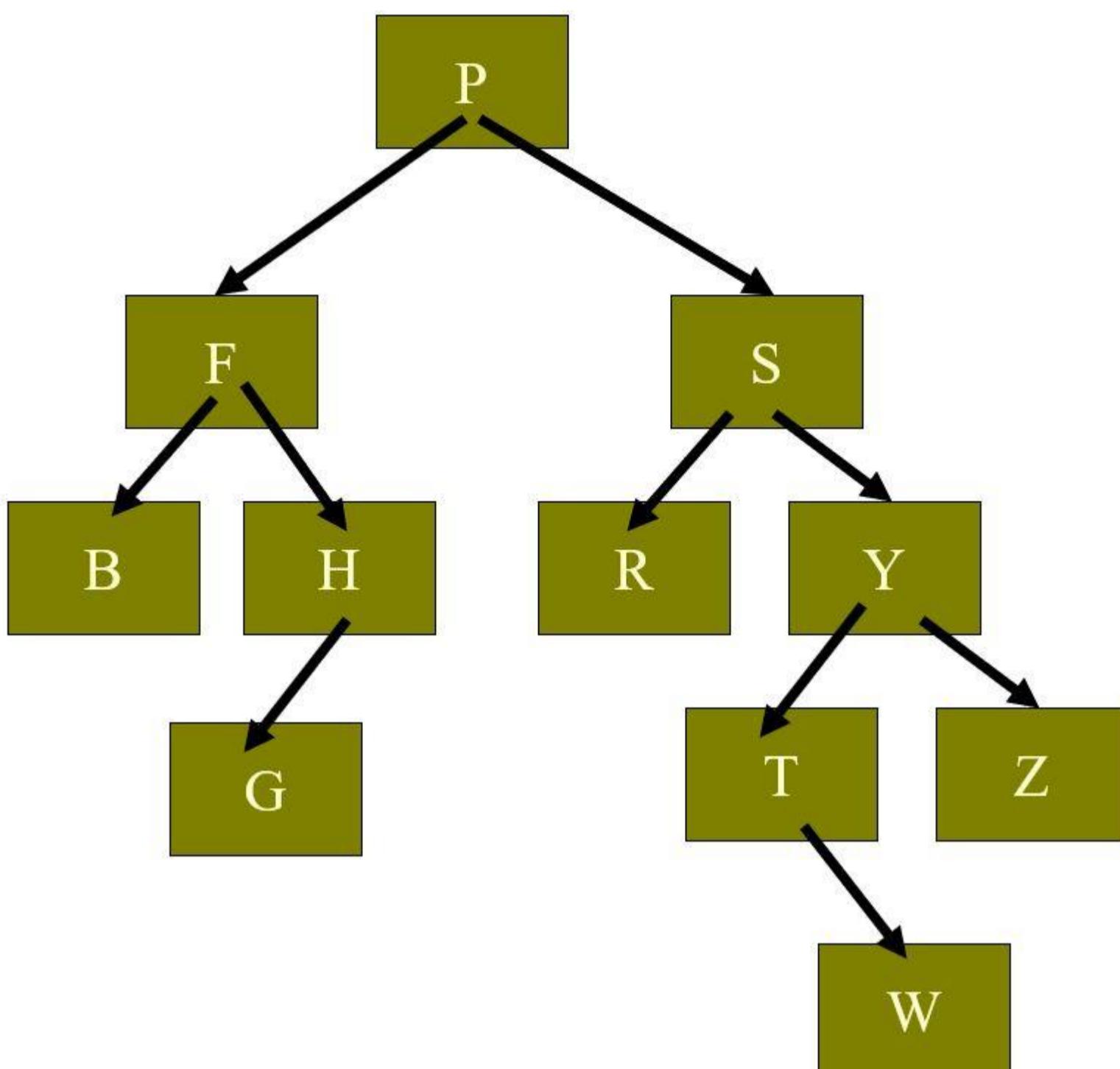


Ilustración 233: Árbol binario de ejemplo

```

Consola de depuración de Mi + 
Recorrido por niveles
P --
F -- S --
B -- H -- R -- Y --
G -- T -- Z --
W --
  
```

Ilustración 266: Recorrer un árbol binario por niveles

Árbol N-ario

Un árbol donde puede haber de 0 a N hijos por rama

H/020.cs

```
namespace Ejemplo {
    class Nodo {
        //Atributos propios
        public string Cadena { get; set; }
        public char Caracter { get; set; }
        public int Entero { get; set; }
        public double NumReal { get; set; }

        public List<Nodo> Hijos; //Uso de una Lista para sostener los hijos del nodo

        public Nodo(string Cadena, char Caracter, int Entero, double NumReal) {
            this.Cadena = Cadena;
            this.Caracter = Caracter;
            this.Entero = Entero;
            this.NumReal = NumReal;
            Hijos = new List<Nodo>(); //Crea la lista vacía
        }

        public void AgregaHijo(Nodo hijo) {
            Hijos.Add(hijo); //Agrega hijo a la lista
        }

        //Imprime Contenido
        public void Imprime() {
            Console.WriteLine("Cadena: " + Cadena + " Caracter: " + Caracter.ToString());
            Console.WriteLine(" Entero: " + Entero.ToString() + " Real: " + NumReal.ToString());
            Console.WriteLine(" Número de hijos: " + Hijos.Count + "\r\n");
        }
    }

    class Program {
        static void Main() {
            //Crea la raíz del árbol N-ario
            Nodo arbolN = new Nodo("AAAA", 'a', 1, 0.1);

            //Agrega varios hijos a esa raíz
            arbolN.AgregaHijo(new Nodo("BBBB", 'b', 2, 0.2));
            arbolN.AgregaHijo(new Nodo("CCCC", 'c', 3, 0.3));
            arbolN.AgregaHijo(new Nodo("DDDD", 'd', 4, 0.4));
            arbolN.AgregaHijo(new Nodo("EEEE", 'e', 5, 0.5));
            arbolN.AgregaHijo(new Nodo("FFFF", 'f', 6, 0.6));

            //Agrega varios hijos al nodo "BBBB"
            arbolN.Hijos[0].AgregaHijo(new Nodo("Bhhh", 'h', 7, 0.7));
            arbolN.Hijos[0].AgregaHijo(new Nodo("Biii", 'i', 8, 0.8));
            arbolN.Hijos[0].AgregaHijo(new Nodo("Bjjj", 'j', 9, 0.9));

            //Agrega varios hijos al nodo "EEEE"
            arbolN.Hijos[4].AgregaHijo(new Nodo("Ekkk", 'k', 10, 1.1));
            arbolN.Hijos[4].AgregaHijo(new Nodo("Elll", 'l', 11, 1.2));

            //Imprime el árbol
            arbolN.Imprime();
            arbolN.Hijos[0].Imprime();
            arbolN.Hijos[1].Imprime();
            arbolN.Hijos[2].Imprime();
            arbolN.Hijos[3].Imprime();
            arbolN.Hijos[4].Imprime();
            arbolN.Hijos[0].Hijos[0].Imprime();
        }
    }
}
```

```
Consola de depuración de Mi + ▾  
Cadena: AAAA Caracter: a Entero: 1 Real: 0,1 Número de hijos: 5  
Cadena: BBBB Caracter: b Entero: 2 Real: 0,2 Número de hijos: 3  
Cadena: CCCC Caracter: c Entero: 3 Real: 0,3 Número de hijos: 0  
Cadena: DDDD Caracter: d Entero: 4 Real: 0,4 Número de hijos: 0  
Cadena: EEEE Caracter: e Entero: 5 Real: 0,5 Número de hijos: 0  
Cadena: FFFF Caracter: f Entero: 6 Real: 0,6 Número de hijos: 2  
Cadena: Bhhh Caracter: h Entero: 7 Real: 0,7 Número de hijos: 0
```

Ilustración 267: Árbol N-ario

Recorriendo

Se usa un procedimiento recursivo.

H/021.cs

```
namespace Ejemplo {
    class Nodo {
        //Atributos propios
        public string Cadena { get; set; }
        public char Caracter { get; set; }
        public int Entero { get; set; }
        public double NumReal { get; set; }

        public List<Nodo> Hijos; //Uso de una Lista para sostener los hijos del nodo

        public Nodo(string Cadena, char Caracter, int Entero, double NumReal) {
            this.Cadena = Cadena;
            this.Caracter = Caracter;
            this.Entero = Entero;
            this.NumReal = NumReal;
            Hijos = new List<Nodo>(); //Crea la lista vacía
        }

        public void AgregaHijo(Nodo hijo) {
            Hijos.Add(hijo); //Agrega hijo a la lista
        }

        //Imprime Contenido
        public void Imprime() {
            Console.WriteLine("Cadena: " + Cadena + " Caracter: " + Caracter.ToString());
            Console.WriteLine(" Entero: " + Entero.ToString() + " Real: " + NumReal.ToString());
            Console.WriteLine(" Número de hijos: " + Hijos.Count);
        }
    }

    class Program {
        static void Main() {
            //Crea la raíz del árbol N-ario
            Nodo arbolN = new Nodo("AAAA", 'a', 1, 0.1);

            //Agrega varios hijos a esa raíz
            arbolN.AgregaHijo(new Nodo("BBBB", 'b', 2, 0.2));
            arbolN.AgregaHijo(new Nodo("CCCC", 'c', 3, 0.3));
            arbolN.AgregaHijo(new Nodo("DDDD", 'd', 4, 0.4));
            arbolN.AgregaHijo(new Nodo("EEEE", 'e', 5, 0.5));
            arbolN.AgregaHijo(new Nodo("FFFF", 'f', 6, 0.6));

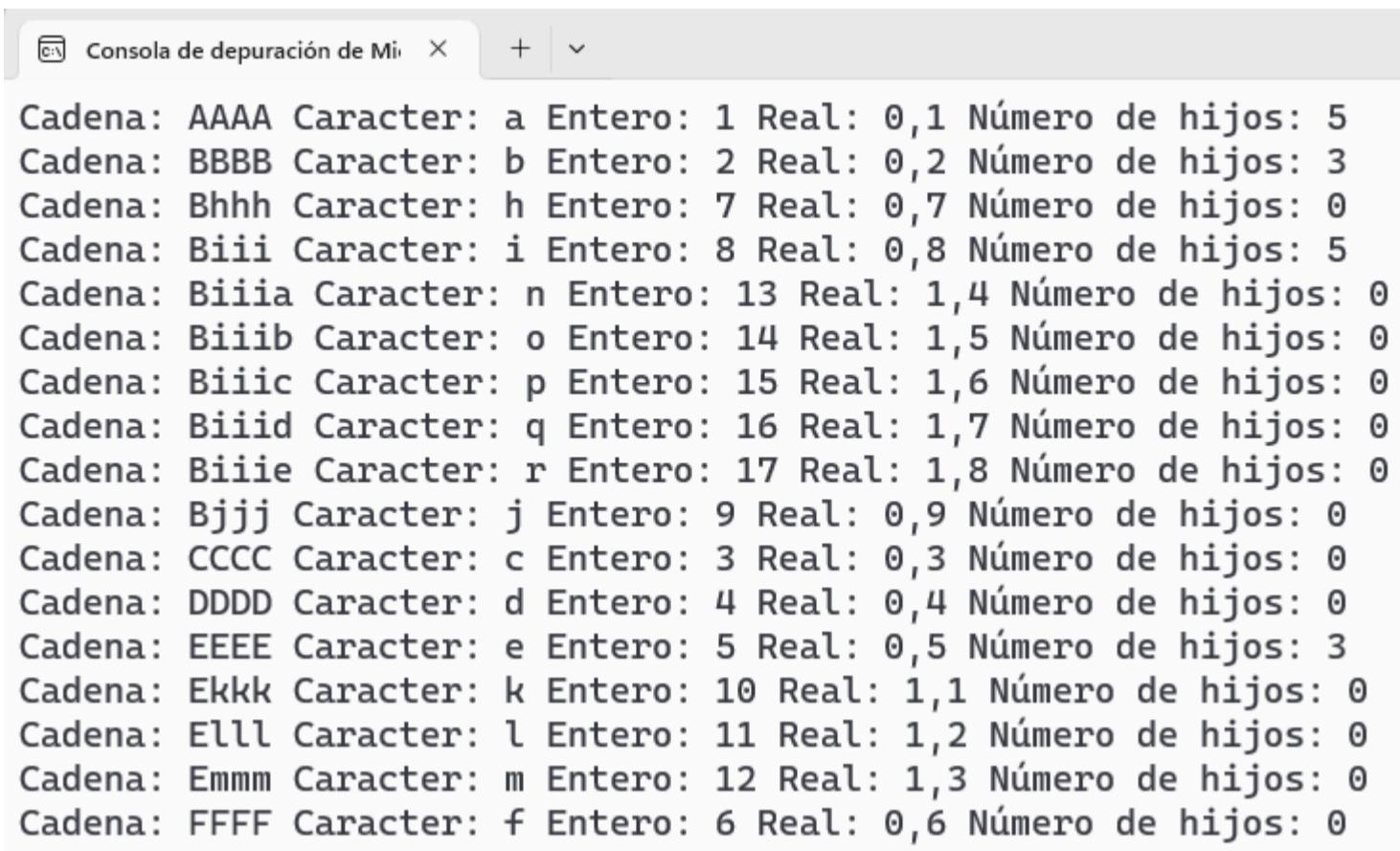
            //Agrega varios hijos al nodo "BBBB"
            arbolN.Hijos[0].AgregaHijo(new Nodo("Bhhh", 'h', 7, 0.7));
            arbolN.Hijos[0].AgregaHijo(new Nodo("Biii", 'i', 8, 0.8));
            arbolN.Hijos[0].AgregaHijo(new Nodo("Bj jj", 'j', 9, 0.9));

            //Agrega varios hijos al nodo "EEEE"
            arbolN.Hijos[3].AgregaHijo(new Nodo("Ekkk", 'k', 10, 1.1));
            arbolN.Hijos[3].AgregaHijo(new Nodo("Elll", 'l', 11, 1.2));
            arbolN.Hijos[3].AgregaHijo(new Nodo("Emmm", 'm', 12, 1.3));

            //Agrega varios hijos al nodo "Biii"
            arbolN.Hijos[0].Hijos[1].AgregaHijo(new Nodo("Biiia", 'n', 13, 1.4));
            arbolN.Hijos[0].Hijos[1].AgregaHijo(new Nodo("Biiib", 'o', 14, 1.5));
            arbolN.Hijos[0].Hijos[1].AgregaHijo(new Nodo("Biiic", 'p', 15, 1.6));
            arbolN.Hijos[0].Hijos[1].AgregaHijo(new Nodo("Biiid", 'q', 16, 1.7));
            arbolN.Hijos[0].Hijos[1].AgregaHijo(new Nodo("Biiie", 'r', 17, 1.8));

            //Imprime el árbol
            RecorreArbolN(arbolN);
        }

        //Recorre el árbol
        static void RecorreArbolN(Nodo Arbol) {
            if (Arbol != null) {
                Arbol.Imprime();
                for (int cont=0; cont<Arbol.Hijos.Count; cont++)
                    RecorreArbolN(Arbol.Hijos[cont]);
            }
        }
    }
}
```



The screenshot shows a debugger console window titled "Consola de depuración de Mi". The output area displays a series of log entries representing the traversal of an N-ary tree. Each entry consists of four fields: Cadena, Caracter, Entero, and Real. The Cadena field contains a string of characters (e.g., AAAA, BBBB, Bhhh, Bi...), the Caracter field shows the character being processed, the Entero field shows the index or depth of the node, and the Real field shows a value associated with the node. The final field, "Número de hijos", indicates the number of children for each node. The traversal starts at index 1 and moves through various characters, with some indices (like 13, 14, 15, 16, 17) appearing multiple times, suggesting a non-binary structure where some nodes have more than two children.

```
Cadena: AAAA Caracter: a Entero: 1 Real: 0,1 Número de hijos: 5
Cadena: BBBB Caracter: b Entero: 2 Real: 0,2 Número de hijos: 3
Cadena: Bhhh Caracter: h Entero: 7 Real: 0,7 Número de hijos: 0
Cadena: Bi... Caracter: i Entero: 8 Real: 0,8 Número de hijos: 5
Cadena: Bi... Caracter: n Entero: 13 Real: 1,4 Número de hijos: 0
Cadena: Bi... Caracter: o Entero: 14 Real: 1,5 Número de hijos: 0
Cadena: Bi... Caracter: p Entero: 15 Real: 1,6 Número de hijos: 0
Cadena: Bi... Caracter: q Entero: 16 Real: 1,7 Número de hijos: 0
Cadena: Bi... Caracter: r Entero: 17 Real: 1,8 Número de hijos: 0
Cadena: Bj... Caracter: j Entero: 9 Real: 0,9 Número de hijos: 0
Cadena: CCCC Caracter: c Entero: 3 Real: 0,3 Número de hijos: 0
Cadena: DDDD Caracter: d Entero: 4 Real: 0,4 Número de hijos: 0
Cadena: EEEE Caracter: e Entero: 5 Real: 0,5 Número de hijos: 3
Cadena: Ek... Caracter: k Entero: 10 Real: 1,1 Número de hijos: 0
Cadena: El... Caracter: l Entero: 11 Real: 1,2 Número de hijos: 0
Cadena: Em... Caracter: m Entero: 12 Real: 1,3 Número de hijos: 0
Cadena: FFFF Caracter: f Entero: 6 Real: 0,6 Número de hijos: 0
```

Ilustración 268: Recorriendo árbol N-ario

Grafos

Para generar cualquier otra estructura de nodos interconectados, se hace uso de grafos.

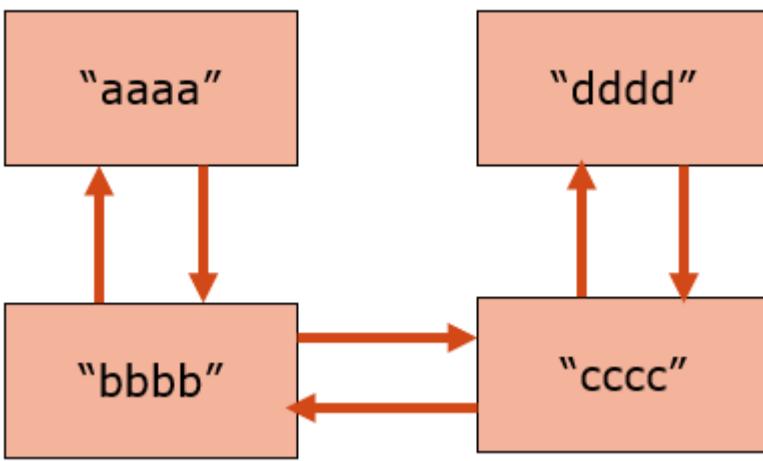


Ilustración 234: Grafo generado

H/022.cs

```
//Grafo básico
namespace Ejemplo {
    //Unidad básica en el grafo cuadrado. Apuntará Arriba, Abajo, Derecha e Izquierda
    class Nodo {
        public string Cadena { get; set; }

        //Apuntadores en las 4 direcciones
        public Nodo Arriba;
        public Nodo Abajo;
        public Nodo Derecha;
        public Nodo Izquierda;

        //Constructor
        public Nodo(string Cadena) {
            this.Cadena = Cadena;
        }
    }

    class Program {
        public static void Main() {
            //Genera los nodos
            Nodo nodoA = new Nodo("aaaa");
            Nodo nodoB = new Nodo("bbbb");
            Nodo nodoC = new Nodo("cccc");
            Nodo nodoD = new Nodo("dddd");

            //Une los nodos para crear el grafo
            nodoA.Abajo = nodoB;
            nodoB.Arriba = nodoA;

            nodoB.Derecha = nodoC;
            nodoC.Izquierda = nodoB;

            nodoC.Arriba = nodoD;
            nodoD.Abajo = nodoC;

            //Imprime
            Console.WriteLine("nodoA: " + nodoA.Cadena);
            Console.WriteLine("nodoA->Abajo: " + nodoA.Abajo.Cadena);
            Console.WriteLine("nodoA->Abajo->Derecha: " + nodoA.Abajo.Derecha.Cadena);
            Console.WriteLine("nodoA->Abajo->Derecha->Arriba: " + nodoA.Abajo.Derecha.Arriba.Cadena);
        }
    }
}
```

```
Consola de depuración de Mi... X + ▾
nodoA: aaaa
nodoA->Abajo: bbbb
nodoA->Abajo->Derecha: cccc
nodoA->Abajo->Derecha->Arriba: dddd
```

Ilustración 270: Grafos

```
//Generando un grafo aleatoriamente
namespace Ejemplo {
    //Unidad básica en el grafo cuadrado. Apuntará Arriba, Abajo, Derecha e Izquierda
    class Nodo {
        public int Numero { get; set; }

        //Apuntadores en las 4 direcciones
        public Nodo Arriba;
        public Nodo Abajo;
        public Nodo Derecha;
        public Nodo Izquierda;

        //Constructor
        public Nodo(int Numero) {
            this.Numero = Numero;
        }
    }

    class Program {
        public static void Main() {
            Random azar = new Random();

            //Usa una lista para guardar los nodos
            List<Nodo> listado = new List<Nodo>();

            //Genera los nodos dentro de un List
            int Total = azar.Next(20, 30);
            for (int cont = 1; cont <= Total; cont++) {
                listado.Add(new Nodo(cont));
            }

            //Ahora interconecta los nodos al azar
            Total = azar.Next(50, 200);
            for (int cont = 1; cont <= Total; cont++) {
                int nodoA = azar.Next(listado.Count);
                int nodoB;
                do {
                    nodoB = azar.Next(listado.Count);
                } while (nodoA == nodoB);

                switch (azar.Next(4)) {
                    case 0: listado[nodoA].Arriba = listado[nodoB]; break;
                    case 1: listado[nodoA].Abajo = listado[nodoB]; break;
                    case 2: listado[nodoA].Izquierda = listado[nodoB]; break;
                    case 3: listado[nodoA].Derecha = listado[nodoB]; break;
                }
            }

            //Imprime el grafo para ser interpretado por viz.js
            Console.WriteLine("digraph testgraph{");
            for (int cont = 0; cont < listado.Count; cont++) {
                if (listado[cont].Arriba != null) Console.WriteLine(listado[cont].Numero + "->" +
listado[cont].Arriba.Numero);
                if (listado[cont].Abajo != null) Console.WriteLine(listado[cont].Numero + "->" +
listado[cont].Abajo.Numero);
                if (listado[cont].Izquierda != null) Console.WriteLine(listado[cont].Numero + "->" +
listado[cont].Izquierda.Numero);
                if (listado[cont].Derecha != null) Console.WriteLine(listado[cont].Numero + "->" +
listado[cont].Derecha.Numero);
            }
            Console.WriteLine("}");
        }
    }
}
```

Consola de depuración de Mi X

```
digraph testgraph{\n    1->16\n    1->6\n    2->18\n    2->13\n    2->17\n    3->16\n    3->20\n    4->15\n    4->16\n    4->13\n    4->2\n}
```

Ilustración 271: Gafo generado

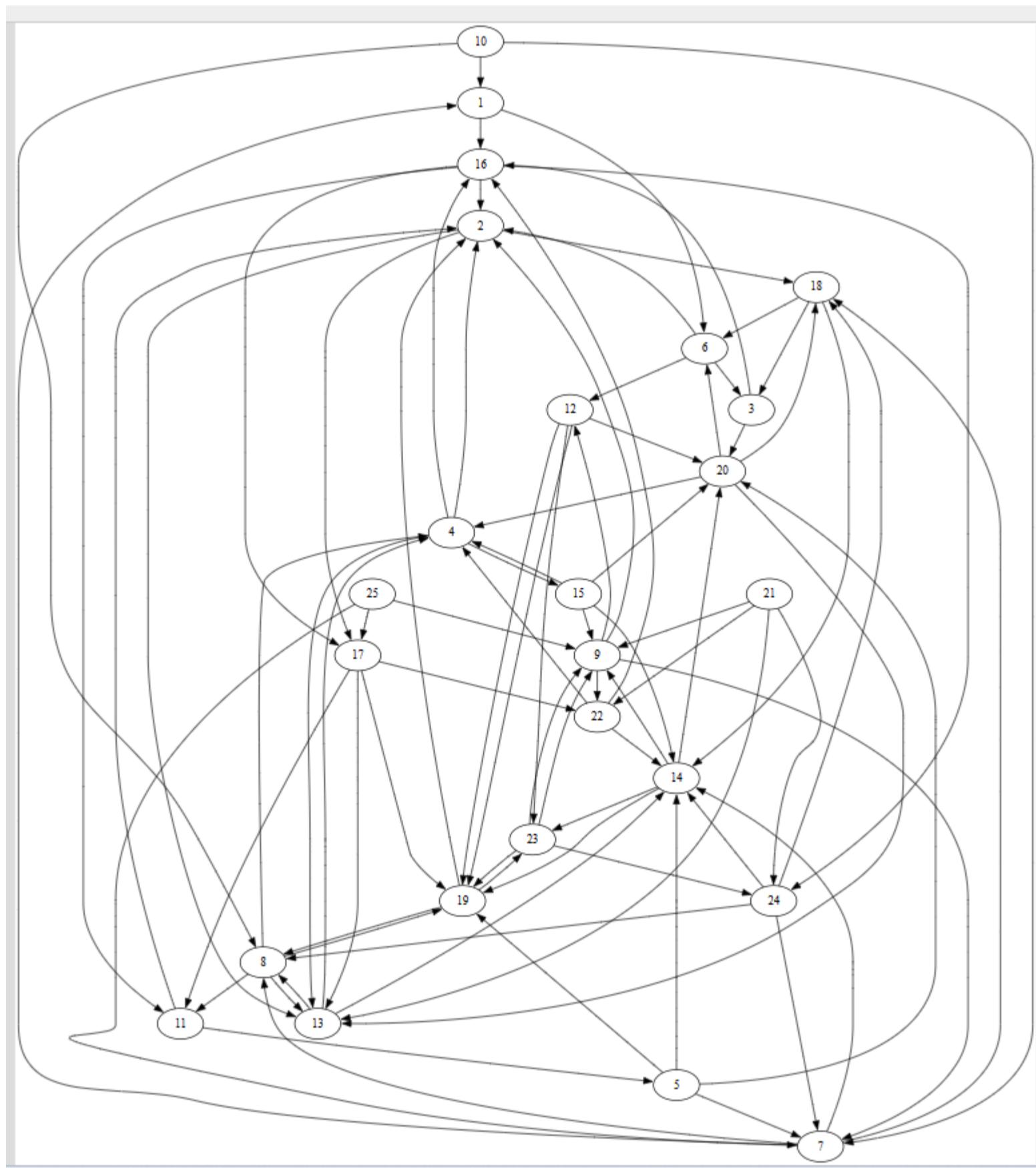


Ilustración 272: Grafo dibujado con <https://viz-js.com/>

Parte 9: Simulaciones

Números aleatorios

.NET 8 tiene su propio generador de números pseudoaleatorios (no existe la aleatoriedad perfecta construida en un lenguaje de programación, para tal propósito, se requeriría hardware especializado que genere números aleatorios). El generador de .NET 8 requiere un valor semilla, por defecto, .NET 8 usa el reloj de la máquina. Cuando se trabaja con números aleatorios en modelos de simulación, entonces son números reales de tal manera que $0 \leq r < 1$, donde r es la denominación del número aleatorio

I/001.cs

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Generando números al azar.
            Random azar = new Random(); //La semilla del generador la define .NET

            //Números enteros al azar
            Console.WriteLine("Enteros positivos: ");
            for (int Contador = 1; Contador <= 10; Contador++) {
                int numEntero = azar.Next();
                Console.WriteLine(numEntero.ToString() + " | ");
            }

            //Números entre 0 y 1 al azar
            Console.WriteLine("\r\n\r\nReales: ");
            for (int Contador = 1; Contador <= 10; Contador++) {
                double numReal = azar.NextDouble();
                Console.WriteLine(numReal.ToString() + " | ");
            }

            //Números entre 15 y 44 al azar.
            Console.WriteLine("\r\n\r\nEnteros positivos entre 15 y 44: ");
            for (int Contador = 1; Contador <= 10; Contador++) {
                //El segundo parámetro debe ser +1 del rango máximo que se busca
                int numEntero = azar.Next(15, 45);
                Console.WriteLine(numEntero.ToString() + " | ");
            }

            //Generando los mismos valores
            Console.WriteLine("\r\n\r\nGenerando los mismos valores al usar la misma semilla: ");
            Random AleatorioA = new Random(500);
            Random AleatorioB = new Random(500);
            for (int Contador = 1; Contador <= 20; Contador++) {
                int numA = AleatorioA.Next(55, 95);
                int numB = AleatorioB.Next(55, 95);
                Console.WriteLine(numA.ToString() + " y " + numB.ToString() + " | ");
            }

            Console.WriteLine(" Final");
        }
    }
}
```

```
Consola de depuración de Mi... X + - □ ×
Enteros positivos: 2111923085 | 138467953 | 482820391 | 877967970 | 1550858139 | 2
034543776 | 784953033 | 24423575 | 969063847 | 1974149762 |

Reales: 0,7112869853854452 | 0,4797890592792097 | 0,8514211978628117 | 0,539018121
0406605 | 0,8079320879260405 | 0,6909123270727379 | 0,28510259183105124 | 0,664269
7127656781 | 0,6214570462263495 | 0,7830006306838707 |

Enteros positivos entre 15 y 44: 41 | 40 | 23 | 18 | 21 | 15 | 35 | 25 | 25 | 16 |

Generando los mismos valores al usar la misma semilla: 92 y 92 | 76 y 76 | 65 y 65
| 66 y 66 | 92 y 92 | 75 y 75 | 92 y 92 | 66 y 66 | 88 y 88 | 80 y 80 | 81 y 81 |
87 y 87 | 69 y 69 | 78 y 78 | 93 y 93 | 66 y 66 | 56 y 56 | 94 y 94 | 63 y 63 | 9
1 y 91 | Final

C:\Users\engin\source\repos\Ejemplo\Ejemplo\bin\Debug\net8.0\Ejemplo.exe (proceso
```

Ilustración 273: Generando números aleatorios

Generadores de números aleatorios

Es posible hacer un propio generador de números pseudoaleatorios, hay varios algoritmos que pueden ser útiles (https://es.wikipedia.org/wiki/Generador_de_n%C3%BAmeros_pseudoaleatorios).

Generador congruencial lineal

Hace uso de la siguiente fórmula:

$$X_1 = (A * X_0 + B) \% N \quad r_1 = X_1/N$$

$$X_2 = (A * X_1 + B) \% N \quad r_2 = X_2/N$$

$$X_3 = (A * X_2 + B) \% N \quad r_3 = X_3/N$$

$$X_n = (A * X_{n-1} + B) \% N$$

A, B, X_0 y N son valores llamados “semillas” que son ingresados por el usuario, a partir de ese punto el algoritmo genera los números pseudoaleatorios. Y la operación $\%$ retorna el residuo de la división.

I/002.cs

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Generador congruencial lineal
            long X0, A, B, N;

            //Valores de inicio. Se los da el usuario.
            X0 = 302;
            A = 278;
            B = 435;
            N = 871;

            for (int contador = 1; contador <= 100; contador++) {
                X0 = (A * X0 + B) % N;
                double r = (double) X0 / N;
                Console.WriteLine("Número pseudo-aleatorio: " + X0 + " r: " + r);
            }
        }
    }
}
```



The screenshot shows a terminal window titled "Consola de depuración de Mi". It displays 100 lines of output, each consisting of a pseudo-random number followed by its corresponding decimal representation. The numbers range from 210 to 775.

Número pseudo-aleatorio	r
775	0,8897818599311137
748	0,8587830080367393
210	0,24110218140068887
458	0,5258323765786452
593	0,6808266360505166
670	0,7692307692307693
301	0,3455797933409874
497	0,5706084959816303

Ilustración 274: Números aleatorios

Hace uso de la siguiente fórmula:

$$X_1 = (X_0)^2 \% M \quad r_1 = X_1/M$$

$$X_2 = (X_1)^2 \% M \quad r_2 = X_2/M$$

$$X_3 = (X_2)^2 \% M \quad r_3 = X_3/M$$

$$X_n = (X_{n-1})^2 \% M$$

Donde M es el producto de la multiplicación de dos números primos muy grandes.

I/003.cs

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Generador Blum Blum Shub
            long X0, M;

            //Valores de inicio. Se los da el usuario.
            X0 = 3;
            M = 11*19;

            for (int contador = 1; contador <= 100; contador++) {
                X0 = (X0*X0) % M;
                double r = (double) X0 / M;
                Console.WriteLine("Número pseudo-aleatorio: " + X0 + " r: " + r);
            }
        }
    }
}
```

En el código se probó como demostración con números primos muy pequeños, eso se debe cambiar.

Pruebas a generadores de números aleatorios

¿Cómo saber si un generador de números aleatorios es aceptable? Hay diversas pruebas estadísticas para probar. A continuación, se muestran unas cuantas:

1. Los números generados deben dar como promedio un número muy cercano a 0.5
2. La varianza debe ser cercana a 1/12
3. Los números deben estar uniformemente distribuidos en el rango 0 a 1
4. Cada número es independiente del anterior y no afecta al siguiente

I/004.cs

```
namespace Ejemplo {
    internal class Program {
        static void Main(string[] args) {

            List<double> Numeros = new List<double>();

            //Generador congruencial lineal
            long X0, A, B, Ndiv;

            //Valores de inicio. Se los da el usuario.
            X0 = 7302; //debe ser menor de Ndiv
            A = 5278;
            B = 3435;
            Ndiv = 20000;

            for (int contador = 1; contador <= 2000; contador++) {
                X0 = (A * X0 + B) % Ndiv;
                double r = (double)X0 / Ndiv;
                Numeros.Add(r);
            }

            //=====
            //Prueba de promedio
            //=====
            double Acumula = 0;
            for (int cont = 0; cont < Numeros.Count; cont++) {
                Acumula += Numeros[cont];
            }
            double Promedio = (double)Acumula / Numeros.Count;
            Console.WriteLine("Promedio es: {0:0.00000} y debe ser similar a 0,5", Promedio);

            //=====
            //Prueba de varianza
            //=====
            double Sumatoria = 0;
            for (int cont = 0; cont < Numeros.Count; cont++) {
                Sumatoria += (Numeros[cont] - Promedio) * (Numeros[cont] - Promedio);
            }
            double Varianza = Sumatoria / Numeros.Count;
            Console.WriteLine("Promedio es: {0:0.00000} y debe ser similar a {1:0.00000}", Varianza, (1.0 /
12.0));

            //=====
            //Prueba de Uniformidad
            //=====
            int[] Rango = new int[10];
            for (int cont = 0; cont < Numeros.Count; cont++) {
                if (Numeros[cont] < 0.1) Rango[0]++;
                else if (Numeros[cont] < 0.2) Rango[1]++;
                else if (Numeros[cont] < 0.3) Rango[2]++;
                else if (Numeros[cont] < 0.4) Rango[3]++;
                else if (Numeros[cont] < 0.5) Rango[4]++;
                else if (Numeros[cont] < 0.6) Rango[5]++;
                else if (Numeros[cont] < 0.7) Rango[6]++;
                else if (Numeros[cont] < 0.8) Rango[7]++;
                else if (Numeros[cont] < 0.9) Rango[8]++;
                else Rango[9]++;
            }

            Console.WriteLine("\n\nPrueba de Uniformidad");
            Console.WriteLine("Números por rango");
            Console.WriteLine(" Rango\t\tFrecuencia Obtenida Fo\tFrecuencia Esperada Fe\t\t((Fe-Fo)^2)/Fe");

            double FrecuenciaEsperada = Numeros.Count / 10;
            double SumaRango = 0;
            for (int cont = 0; cont < Rango.Length; cont++) {
                double Minimo = cont / 10.0;
```

```

        double Maximo = (cont + 1) / 10.0;
        double Diferencia = (double)(FrecuenciaEsperada - Rango[cont]) * (FrecuenciaEsperada -
Rango[cont]) / FrecuenciaEsperada;
        Console.WriteLine(" {0:0.0} y {1:0.0}\t\t{2:#}\t\t{3:#}\t\t{4:0.00000}", Minimo, Maximo,
Rango[cont], FrecuenciaEsperada, Diferencia);
        SumaRango += Diferencia;
    }

    if (SumaRango < 16.9)
        Console.WriteLine("Pasa la prueba de uniformidad porque " + SumaRango.ToString() + " debe ser
menor a 16,9");
    else
        Console.WriteLine("NO aprobó la prueba de uniformidad porque " + SumaRango.ToString() + " fue
mayor o igual a 16,9");

    //=====
    //Prueba de Uniformidad de Kolmogorov-Smirnov
    //=====
    double[] ProbabilidadAcumula = new double[10];
    ProbabilidadAcumula[0] = (double)Rango[0] / Numeros.Count;
    for (int cont = 1; cont < Rango.Length; cont++) {
        ProbabilidadAcumula[cont] = (double)ProbabilidadAcumula[cont - 1] + (double)Rango[cont] /
Numeros.Count;
    }

    Console.WriteLine("\n\nPrueba de Uniformidad de Kolmogorov-Smirnov");
    Console.WriteLine("Probabilidad acumulada");
    Console.WriteLine(" Rango\t\t\tEsperada\t\tObtenida\t\tDiferencia");
    double MaximaDiferencia = 0;
    for (int cont = 0; cont < ProbabilidadAcumula.Length; cont++) {
        double Minimo = cont / 10.0;
        double Maximo = (cont + 1) / 10.0;
        double Espera = (double)(cont + 1) / 10;
        double Diferencia = Math.Abs(ProbabilidadAcumula[cont] - Espera);
        if (Diferencia > MaximaDiferencia) MaximaDiferencia = Diferencia;
        Console.WriteLine(" {0:0.0} y {1:0.0}\t\t{2:0.00000}\t\t{3:0.00000}\t\t{4:0.00000}", Minimo,
Maximo, Espera, ProbabilidadAcumula[cont], Diferencia);
    }

    double Compara = 1.36 / Math.Sqrt(Numeros.Count);
    if (MaximaDiferencia < Compara)
        Console.WriteLine("Pasa la prueba de uniformidad porque " + MaximaDiferencia.ToString() + " <
" + Compara.ToString());
    else
        Console.WriteLine("NO aprobó la prueba de uniformidad porque " + MaximaDiferencia.ToString()
+ " >= " + Compara.ToString());

    //=====
    // Prueba de Independencia - Wald-Wolfowitz
    //=====
    Console.WriteLine("\n\nPrueba de Independencia - Wald-Wolfowitz");

    //Deduce el valor de R, N1, N2, N
    double N1 = 0, N2 = 0, R = 0;
    int Bloque = 0;
    for (int cont = 0; cont < Numeros.Count; cont++) {
        if (Numeros[cont] < Promedio) {
            if (Bloque != 1) R++;
            Bloque = 1;
            N1++;
        }
        else {
            if (Bloque != 2) R++;
            Bloque = 2;
            N2++;
        }
    }

    double N = Numeros.Count;

    Console.WriteLine("N = " + N);
    Console.WriteLine("N1 = " + N1);
    Console.WriteLine("N2 = " + N2);
    Console.WriteLine("R = " + R);

    //Deduce la media
    double Media = (2 * N1 * N2) / (N + 1);

```

```

        double Variar = (Media - 1) * (Media - 2) / (N - 1);
        double Z = (R - Media) / Math.Sqrt(Variar);

        Console.WriteLine("Media = " + Media);
        Console.WriteLine("Variación = " + Variar);
        Console.WriteLine("Z = " + Z);

        if (Z >= -1.96 && Z <= 1.96)
            Console.WriteLine("Pasa la prueba de independencia porque Z está entre -1,96 y 1,96");
        else
            Console.WriteLine("NO pasó la prueba de independencia porque Z está fuera del rango de -1,96 y 1,96");
    }
}
}

```

Consola de depuración de Mi... X + ▾

Promedio es: 0,50078 y debe ser similar a 0,5
 Promedio es: 0,08345 y debe ser similar a 0,08333

Prueba de Uniformidad

Números por rango

Rango	Frecuencia Obtenida Fo	Frecuencia Esperada Fe	((Fe-Fo)^2)/Fe
0,0 y 0,1	197	200	0,04500
0,1 y 0,2	206	200	0,18000
0,2 y 0,3	196	200	0,08000
0,3 y 0,4	203	200	0,04500
0,4 y 0,5	194	200	0,18000
0,5 y 0,6	204	200	0,08000
0,6 y 0,7	196	200	0,08000
0,7 y 0,8	204	200	0,08000
0,8 y 0,9	197	200	0,04500
0,9 y 1,0	203	200	0,04500

Pasa la prueba de uniformidad porque 0,86 debe ser menor a 16,9

Prueba de Uniformidad de Kolmogorov-Smirnov

Probabilidad acumulada

Rango	Esperada	Obtenida	Diferencia
0,0 y 0,1	0,10000	0,09850	0,00150
0,1 y 0,2	0,20000	0,20150	0,00150
0,2 y 0,3	0,30000	0,29950	0,00050
0,3 y 0,4	0,40000	0,40100	0,00100
0,4 y 0,5	0,50000	0,49800	0,00200
0,5 y 0,6	0,60000	0,60000	0,00000
0,6 y 0,7	0,70000	0,69800	0,00200
0,7 y 0,8	0,80000	0,80000	0,00000
0,8 y 0,9	0,90000	0,89850	0,00150
0,9 y 1,0	1,00000	1,00000	0,00000

Pasa la prueba de uniformidad porque 0,002000000000000018 < 0,03041052449399714

Ilustración 275: Prueba del generador de números aleatorios

Prueba de Independencia - Wald-Wolfowitz

N = 2000

N1 = 1000

N2 = 1000

R = 999

Media = 999,5002498750624

Variación = 498,25125000034365

Z = -0,022411080232072486

Pasa la prueba de independencia porque Z está entre -1,96 y 1,96

Ilustración 276: Prueba del generador de números aleatorios

Variables aleatorias

Una variable aleatoria es el resultado de una ecuación que hace uso de números aleatorios, se representa así:

$$V = F(r)$$

Donde r es el número aleatorio entre 0 y 1. La variable aleatoria es usada dentro de los modelos de simulación, como la parte no controlable, pero si estimable. Por ejemplo, lanzar una moneda, puede generar cara o cruz. Ese valor es la variable aleatoria y puede expresarse de esta forma en C#:

```
Random Azar = new Random(); //Para el número aleatorio entre 0 y 1  
  
//La variable aleatoria  
if (Azar.NextDouble() < 0.5)  
    return "Cara";  
else  
    return "Cruz";
```

Las funciones que pueden tener las variables aleatorias pueden ser discretas como la mostrada anteriormente con solo dos valores devueltos, que es "cara" o "cruz", o también pueden ser continuas, es decir, retornan un número real. A esas funciones continuas se les conocen como distribuciones. Ejemplo:

$$V = \operatorname{seno}(r)$$

Distribuciones

Distribución Normal

La distribución normal es una de las más utilizadas porque ciertos fenómenos tienen un comportamiento que sigue esta distribución:

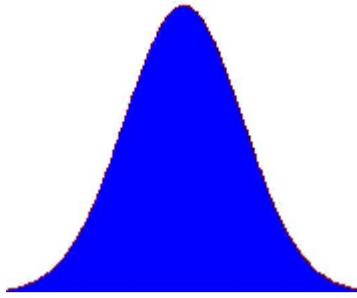


Ilustración 277: Distribución Normal

Los parámetros recibidos para poder usar esta distribución son M (media) y D (Desviación)

Para generar la variable aleatoria que utilice estos parámetros se usa la ecuación:

$$V = M + D * c$$

Donde c es un valor aleatorio que se deduce con la siguiente fórmula:

$$c = \cos(2\pi r_2) * \sqrt{-2 * \ln(r_1)}$$

Donde r_1 y r_2 son números aleatorios.

I/005.cs

```
namespace Ejemplo {
    internal class Program {
        static void Main(string[] args) {
            /* Variable aleatoria
               Distribución Normal. Generar una variable aleatoria con media M y desviación D
               variable = M + D * c
               donde c = cos(2*PI*r2)*raizcuadrada(-2*LogarimoNatural(r1))
               r1 y r2 son números aleatorios */
            Random Azar = new Random();
            double M = 100;
            double D = 7;
            for (int cont = 1; cont <= 100; cont++) {
                double r1 = Azar.NextDouble();
                double r2 = Azar.NextDouble();
                double c = Math.Cos(2 * Math.PI * r2) * Math.Sqrt(-2 * Math.Log(r1));
                double variable = M + D * c;
                Console.WriteLine(variable + " ");
            }
        }
    }
}
```

```
91,09273509318925; 97,44044168349961; 96,5749131291491; 97,49507318386851; 92,93431301403827; 96,59831383697994; 113,359
00647648799; 108,82073392878436; 96,17281722107435; 99,59304816131315; 99,7267788887046; 109,69657058709257; 95,88560911
493428; 106,9492559292143; 84,33046442695398; 96,60900849552849; 94,65528805930252; 91,00534988086203; 99,87411192018213
; 91,49156300179015; 97,56121121821361; 91,58237027118349; 91,04485155867883; 101,61740514354081; 106,54815963223855; 93
,32497847362987; 99,41600743841813; 92,4378916170819; 100,24482652635668; 98,00692137165389; 93,80066464019494; 89,30695
903819958; 90,13565732692831; 89,20742612926568; 102,2247389961108; 98,80531070112356; 103,2674022714811; 109,5071439110
4725; 98,34878543274395; 93,3134901884384; 102,44648187657376; 87,41859654446549; 103,33300603826059; 106,39048892753584
; 105,43451360538772; 109,06005021430859; 99,64033833225344; 110,59070030027003; 102,7001226748231; 99,32177711956038; 9
8,53647068318647; 97,6604425424962; 97,15927205374693; 106,64543313761894; 100,48289903672728; 96,2402919053433; 98,4475
1132578081; 101,9612839215683; 106,89640716601917; 106,25169968354072; 99,66062394409292; 92,12099877398896; 107,9178283
6669323; 99,9572866620369; 91,74307421049421; 94,27243781160706; 92,9384593590587; 104,45408598096505; 86,77342256058854
; 95,04469383150162; 102,87384819939824; 100,57637422083039; 104,63227758940657; 102,28458990761193; 108,57669658869006;
109,53855969201723; 93,02156636136128; 93,29738390575287; 111,4659475748849; 101,43788955788929; 97,70535669156044; 114
,2073607617365; 103,01700662099124; 109,55295362287553; 111,72341231174859; 101,01893119377506; 104,84774887732819; 101,
71230925217712; 101,0505482725827; 90,0891989543054; 97,1332721264091; 95,7240766331298; 98,98567759816731; 100,27974829
545761; 104,4880873444565; 106,06914811028268; 86,3809025393002; 98,54847865623142; 100,08576282175427; 100,885134904295
91;
```

Ilustración 278: Distribución Normal

Distribución Triangular

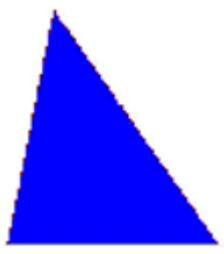


Ilustración 279: Distribución Triangular

Los parámetros recibidos para poder usar esta distribución son un valor mínimo (a), un valor más probable (b) y un valor máximo (c)

Para calcular el valor de la variable aleatorio está el siguiente algoritmo:

Si $r < \frac{b-a}{c-a}$ entonces

$$V = a + \sqrt{r * (c - a) * (b - a)}$$

De lo contrario

$$V = c - \sqrt{(1 - r) * (c - a) * (c - b)}$$

Donde r es un número aleatorio.

I/006.cs

```
namespace Ejemplo {
    internal class Program {
        static void Main(string[] args) {
            /* Distribución Triangular. Generar una variable aleatoria con valor mínimo A,
               valor más probable B y valor máximo C
               Si r < (B-A) / (C-A)
                   Variable = A + raizcuadrada(r * (C-A) * (B-A))
               de lo contrario
                   Variable = C - raizcuadrada((1-r) * (C-A) * (C-B))
               */
            Random Azar = new Random();

            double A = 150;
            double B = 190;
            double C = 230;
            double variable = 0;
            for (int cont = 1; cont <= 100; cont++) {
                double r = Azar.NextDouble();
                if (r < (B - A) / (C - A))
                    variable = A + Math.Sqrt(r * (C - A) * (B - A));
                else
                    variable = C - Math.Sqrt((1 - r) * (C - A) * (C - B));
                Console.WriteLine(variable + "; ");
            }
        }
    }
}
```

Consola de depuración de Mi - + ▾

```
195,34711517747394; 203,4416867899472; 210,78373998742856; 156,67002894393372; 201,97677162857684; 167,32496459252542; 1  
95,55580262184566; 168,15495126476281; 219,3049209020694; 220,45109642508507; 202,63227769186398; 171,05189329648698; 18  
6,7067991665697; 190,9749940270029; 210,20761420114084; 191,30926409338866; 182,09141160423775; 199,17745016954947; 215,  
23503832869423; 196,33786197312853; 204,01313954703434; 204,47492723826525; 203,16623706416192; 172,32520537379224; 200,  
50932293482592; 196,67266692797014; 204,92165217934524; 201,6972093981238; 199,42984189472867; 179,93969364378597; 223,4  
5699136449235; 202,90644264231088; 175,24164926421332; 182,05820884967977; 193,91236927049403; 186,35553644049998; 183,3  
0557962816948; 184,654494497482; 197,48352716651624; 194,83790247874575; 184,5163311927075; 192,43896801536465; 186,5677  
7970540276; 221,26462393701195; 180,36398937083058; 209,2243247255419; 183,68653666816027; 200,90221084912213; 213,16273  
874852484; 196,17324861656815; 225,79703150439192; 192,69202812116237; 195,14329621917; 185,74616757655434; 199,96574597  
420027; 168,07608566539952; 190,39509177325496; 170,19912021481767; 188,17659014483382; 196,9417416653822; 184,927206349  
13434; 197,30391137377214; 179,9932927636756; 185,010406956728; 201,44477765797797; 196,59361523865957; 212,015189166516  
6; 220,83325064307311; 185,85571129738133; 213,26739634261472; 161,49240807281262; 199,94409568231936; 197,9023998443545  
; 193,50802418991123; 189,61736357392982; 199,8977289444255; 164,53011652744445; 186,40843578391804; 212,71426189018368;  
204,54800550216814; 168,41797359850895; 185,26145442537992; 190,49195281051928; 172,50023825635725; 186,6063410337001;  
201,9783672157434; 216,69107372463972; 196,05705985275733; 172,1615038366602; 190,46316083653542; 202,05520813907387; 20  
9,575735479553; 192,23597194205533; 200,30277090676623; 163,66875527228865; 177,07679906078843; 197,00489739475296; 220,  
82994741676163; 193,09064807003122; 221,95722188779433;
```

Ilustración 280: Distribución Triangular

Distribución Uniforme

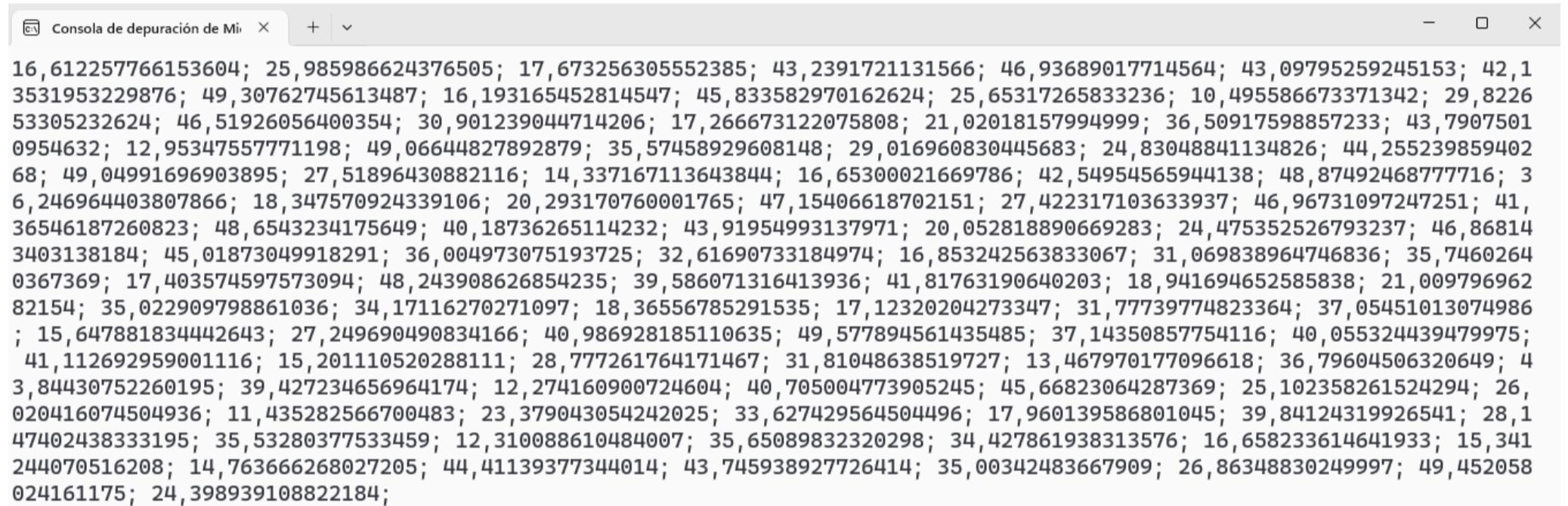
Se requiere un valor aleatorio entre A y B, y que su ocurrencia sea uniforme entre esos dos valores.

$$V = r * (B - A) + A$$

Donde r es un número aleatorio.

I/007.cs

```
namespace Ejemplo {
    internal class Program {
        static void Main(string[] args) {
            /* Variable aleatoria
               Distribución uniforme. Generar un número entero entre A y B
               variable = r * (B - A) + A */
            Random Azar = new Random();
            double A = 10;
            double B = 50;
            for (int cont = 1; cont <= 100; cont++) {
                double r = Azar.NextDouble();
                double variable = r * (B - A) + A;
                Console.WriteLine(variable + " ");
            }
        }
    }
}
```



```
16,612257766153604; 25,985986624376505; 17,673256305552385; 43,2391721131566; 46,93689017714564; 43,09795259245153; 42,1
3531953229876; 49,30762745613487; 16,193165452814547; 45,833582970162624; 25,65317265833236; 10,495586673371342; 29,8226
53305232624; 46,51926056400354; 30,901239044714206; 17,266673122075808; 21,02018157994999; 36,50917598857233; 43,7907501
0954632; 12,95347557771198; 49,06644827892879; 35,57458929608148; 29,016960830445683; 24,83048841134826; 44,255239859402
68; 49,04991696903895; 27,51896430882116; 14,337167113643844; 16,65300021669786; 42,54954565944138; 48,8749246877716; 3
6,246964403807866; 18,347570924339106; 20,293170760001765; 47,15406618702151; 27,422317103633937; 46,96731097247251; 41,
36546187260823; 48,6543234175649; 40,18736265114232; 43,91954993137971; 20,052818890669283; 24,475352526793237; 46,86814
3403138184; 45,01873049918291; 36,004973075193725; 32,61690733184974; 16,853242563833067; 31,069838964746836; 35,7460264
0367369; 17,403574597573094; 48,243908626854235; 39,586071316413936; 41,81763190640203; 18,941694652585838; 21,009796962
82154; 35,022909798861036; 34,17116270271097; 18,36556785291535; 17,12320204273347; 31,77739774823364; 37,05451013074986
; 15,64788183442643; 27,249690490834166; 40,986928185110635; 49,577894561435485; 37,14350857754116; 40,055324439479975;
41,112692959001116; 15,201110520288111; 28,777261764171467; 31,81048638519727; 13,467970177096618; 36,79604506320649; 4
3,84430752260195; 39,427234656964174; 12,274160900724604; 40,705004773905245; 45,66823064287369; 25,102358261524294; 26,
020416074504936; 11,435282566700483; 23,379043054242025; 33,627429564504496; 17,960139586801045; 39,84124319926541; 28,1
47402438333195; 35,53280377533459; 12,310088610484007; 35,65089832320298; 34,427861938313576; 16,658233614641933; 15,341
244070516208; 14,763666268027205; 44,41139377344014; 43,745938927726414; 35,00342483667909; 26,86348830249997; 49,452058
024161175; 24,398939108822184;
```

Ilustración 281: Distribución Uniforme

Problema del viajero

Hay varias ciudades por visitar, sólo puede visitarse una por vez y cada viaje de una ciudad a otra tiene un costo, inclusive ir de la ciudad C a D puede tener un valor distinto de ir de D a C. El viajero debe planificar el viaje de tal manera que visite todas las ciudades, sólo una vez por ciudad y que le salga lo más económico posible.

A continuación, una tabla de costos de viaje de ir de una ciudad (vertical) a otra ciudad (horizontal). Por ejemplo, ir de F a B cuesta \$37, o ir de K a E cuesta \$24

Ciudad	A	B	C	D	E	F	G	H	I	J	K
A	\$ 0	\$ 74	\$ 13	\$ 10	\$ 75	\$ 49	\$ 21	\$ 60	\$ 60	\$ 97	\$ 89
B	\$ 43	\$ 0	\$ 31	\$ 86	\$ 41	\$ 92	\$ 75	\$ 56	\$ 72	\$ 41	\$ 10
C	\$ 67	\$ 39	\$ 0	\$ 64	\$ 88	\$ 75	\$ 82	\$ 50	\$ 68	\$ 14	\$ 38
D	\$ 12	\$ 52	\$ 28	\$ 0	\$ 16	\$ 32	\$ 51	\$ 77	\$ 85	\$ 67	\$ 28
E	\$ 22	\$ 85	\$ 19	\$ 51	\$ 0	\$ 51	\$ 19	\$ 11	\$ 37	\$ 62	\$ 63
F	\$ 10	\$ 37	\$ 13	\$ 35	\$ 12	\$ 0	\$ 43	\$ 53	\$ 49	\$ 98	\$ 77
G	\$ 18	\$ 75	\$ 62	\$ 64	\$ 32	\$ 58	\$ 0	\$ 81	\$ 86	\$ 93	\$ 46
H	\$ 22	\$ 37	\$ 51	\$ 41	\$ 12	\$ 71	\$ 76	\$ 0	\$ 92	\$ 99	\$ 15
I	\$ 36	\$ 33	\$ 71	\$ 25	\$ 92	\$ 10	\$ 93	\$ 52	\$ 0	\$ 98	\$ 61
J	\$ 38	\$ 94	\$ 39	\$ 52	\$ 34	\$ 10	\$ 58	\$ 92	\$ 87	\$ 0	\$ 26
K	\$ 59	\$ 32	\$ 30	\$ 92	\$ 24	\$ 12	\$ 51	\$ 36	\$ 38	\$ 98	\$ 0

Una idea de un trayecto sería este: A → B → C → D → E → F → G → H → I → J → K

El costo del viaje sería: \$74 + \$31 + \$64 + \$16 + \$51 + \$43 + \$81 + \$92 + \$98 + \$26 = \$576

Si se modificara el viaje a: B → A → C → D → E → F → G → H → I → J → K

El costo del viaje sería: \$43 + \$13 + \$64 + \$16 + \$51 + \$43 + \$81 + \$92 + \$98 + \$26 = \$527

Una ruta más económica.

¿Cuántas rutas posibles habría? Es la función factorial del total de ciudades, en el ejemplo serían 11! Rutas = 39.916.800 rutas. Y de todas esas rutas habrá una que será la más económica, pero hay que hacer cálculo de costos por cada ruta. Si fuesen unas 20 ciudades, las rutas posibles serían: 2.432.902.008.176.640.000, haciendo cálculos, y teniendo un computador que sea capaz de calcular el costo de 1000 millones de rutas por segundo, tardaría 2.432.902.008 segundos aproximadamente en evaluar todas las rutas, que equivalen a aproximadamente 77 años. Eso es muchísimo tiempo.

El método propuesto es mucho más rápido y da con una ruta con un costo bajo (no el más bajo, pero si lo suficiente). Tarda pocos segundos. ¿En qué consiste? Parte de una ruta inicial a la cual se le calcula su costo, al azar se toman dos ciudades de la ruta y se intercambian, se vuelve a calcular el nuevo costo, si es menor, entonces se deja la nueva ruta, caso contrario, se repone como estaba anteriormente. Se repite ese proceso por el tiempo deseado, y la última ruta hallada será lo mejor encontrado por el momento.

I/008.cs

```
namespace Ejemplo {
    internal class Program {
        static void Main(string[] args) {
            /* Hay N ciudades a recorrer (0 a N-1). Sólo se puede visitar una ciudad por vez.
             * En la tabla aparece cuanto cuesta ir de una ciudad (origen) a otra ciudad (destino).
             * ¿Qué ruta tomar para visitar todas las ciudades con el mínimo costo? */
            Random Azar = new Random();

            int ciudad = 20; //Número de ciudades
            int minValor = 15; //Valor mínimo que tendrá ir de una ciudad a otra
            int maxValor = 85; //Valor máximo que tendrá ir de una ciudad a otra

            int dest1, dest2; //Ciudad origen a Ciudad destino

            //Genera valores de viaje al azar
            int[,] valorviajes = InicializaValorViajes(ciudad, minValor, maxValor);

            //Imprime los valores
            Imprime(valorviajes);

            //Inicia con una ruta predeterminada 0, 1, 2, 3, ..., N
            int[] ruta = IniciaRuta(ciudad);

            //Deduce el costo de esa ruta predeterminada
            int costo = DeduceCosto(ruta, valorviajes);

            Console.WriteLine("\r\nRutas:");
            ImprimeRuta(ruta, costo);

            //Usando el método Monte Carlo se buscarán otras rutas con menor costo
            for (int pruebas = 1; pruebas <= 700000; pruebas++) {
                dest1 = (int)Math.Floor(Azar.NextDouble() * ruta.Length);
```

```

        do {
            dest2 = (int)Math.Floor(Azar.NextDouble() * ruta.Length);
        } while (dest2 == dest1);
        ModificaRuta(ruta, dest1, dest2);
        int costoNuevo = DeducesCosto(ruta, valorviajes);
        if (costoNuevo < costo) {
            costo = costoNuevo;
            ImprimeRuta(ruta, costo);
            //Console.WriteLine(string.Join(", ", ruta) + " => $" + costo);
        }
        else
            ModificaRuta(ruta, dest1, dest2); //Dejar la ruta como antes
    }
}

//Modifica la ruta de viaje
static void ModificaRuta(int[] ruta, int dest1, int dest2) {
    int tmp = ruta[dest1];
    ruta[dest1] = ruta[dest2];
    ruta[dest2] = tmp;
}

//Inicia el arreglo bidimensional de rutas
static int[,] IniciaRuta(int limite) {
    int[] ruta = new int[limite];
    for (int cont = 0; cont < limite; cont++) ruta[cont] = cont;
    return ruta;
}

//Deduces el costo de la ruta de viaje
static int DeducesCosto(int[] ruta, int[,] costos) {
    int acum = 0;
    for (int cont = 0; cont < ruta.Length - 1; cont++)
        acum += costos[ruta[cont], ruta[cont + 1]];
    return acum;
}

//Llena de valores al azar la tabla de costos de viajes de una ciudad a otra
static int[,] InicializaValorViajes(int ciudad, int minValor, int maxValor) {
    int[,] tablero = new int[ciudad, ciudad];
    Random Azar = new Random();
    for (int fila = 0; fila < ciudad; fila++) {
        for (int columna = 0; columna < ciudad; columna++) {
            tablero[fila, columna] = Azar.Next(minValor, maxValor);
        }
        tablero[fila, fila] = 0;
    }
    return tablero;
}

//Imprime el tablero
static void Imprime(int[,] tablero) {
    Console.WriteLine("Tabla de costos");
    Console.Write("  ");
    for (int col = 0; col < tablero.GetLength(1); col++)
        Console.Write((char)(col + 65) + " ");
    Console.WriteLine(" ");

    for (int fila = 0; fila < tablero.GetLength(0); fila++) {
        Console.Write((char)(fila + 65) + ": ");
        for (int col = 0; col < tablero.GetLength(1); col++)
            Console.Write(tablero[fila, col] + " ");
        Console.WriteLine(" ");
    }
}

//Imprime la ruta y su costo
static void ImprimeRuta(int[] ruta, int costo) {
    for (int col = 0; col < ruta.Length; col++) {
        Console.Write((char)(ruta[col] + 65) + "->");
    }
    Console.WriteLine(" $" + costo);
}
}
}

```

La ejecución del programa genera una tabla de costos al azar para unas 20 ciudades, luego muestra el costo de una ruta inicial y el proceso para encontrar rutas de menor costo cada vez.

Tabla de costos

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
A:	0	56	80	26	64	47	51	27	38	58	58	42	15	20	62	54	62	58	75	34
B:	36	0	20	58	59	47	18	65	23	33	36	46	25	33	58	32	45	41	21	44
C:	71	48	0	32	20	50	55	83	56	21	62	48	48	43	56	47	33	77	34	19
D:	63	37	76	0	16	21	65	60	44	17	81	24	43	71	43	53	64	28	49	66
E:	16	50	64	71	0	79	57	82	78	84	20	84	66	39	76	61	49	77	28	58
F:	60	61	65	61	24	0	27	57	68	25	61	48	81	82	20	56	82	17	34	53
G:	53	84	51	39	15	51	0	27	38	44	36	35	30	63	57	75	19	45	50	83
H:	27	60	77	54	66	53	57	0	62	79	32	74	47	21	19	16	28	69	52	82
I:	73	49	67	80	32	48	37	69	0	43	70	56	57	54	38	27	73	20	32	65
J:	55	64	28	73	41	28	72	49	41	0	59	38	74	47	26	21	17	77	53	71
K:	16	80	70	39	61	16	18	19	73	52	0	83	74	79	79	31	38	40	24	73
L:	61	54	47	71	76	27	24	73	37	40	18	0	42	82	33	23	68	78	75	17
M:	15	82	17	71	33	52	70	31	54	36	71	17	0	53	44	39	55	47	17	73
N:	16	65	73	21	30	45	25	15	80	82	39	81	74	0	73	19	82	65	56	16
O:	68	22	77	81	22	51	17	70	22	48	68	47	66	74	0	37	16	40	63	39
P:	52	64	50	77	30	72	57	51	22	46	38	22	19	26	43	0	45	43	61	50
Q:	41	27	48	84	29	26	78	81	32	52	40	44	47	64	41	27	0	81	46	19
R:	49	53	62	28	61	15	68	34	74	70	33	33	42	63	56	80	52	0	74	48
S:	19	38	44	63	42	22	72	41	80	28	15	81	44	34	70	84	22	19	0	65
T:	38	30	43	56	67	71	28	36	58	18	66	24	57	82	63	23	72	16	38	0

Rutas:

A->B->C->D->E->F->G->H->I->J->K->L->M->N->O->P->Q->R->S->T-> \$974
 A->B->C->D->E->F->G->M->I->J->K->L->H->N->O->P->Q->R->S->T-> \$968
 A->B->C->D->E->F->G->M->I->J->K->L->O->N->H->P->Q->R->S->T-> \$902
 A->B->C->D->E->F->G->M->I->J->S->L->O->N->H->P->Q->R->K->T-> \$861
 A->B->C->D->E->F->G->M->I->J->S->K->O->N->H->P->Q->R->L->T-> \$785
 A->B->C->D->E->T->G->M->I->J->S->K->O->N->H->P->Q->R->L->F-> \$775
 A->B->C->D->E->T->G->M->J->I->S->K->O->N->H->P->Q->R->L->F-> \$734
 A->B->C->D->E->T->G->M->J->I->S->R->O->N->H->P->Q->R->L->F-> \$724
 A->B->C->D->E->T->L->M->J->I->S->R->O->N->H->P->Q->K->G->F-> \$691
 J->B->C->D->E->T->L->M->A->I->S->R->O->N->H->P->Q->K->G->F-> \$675
 J->B->C->D->E->T->L->M->A->I->S->R->F->N->H->P->Q->K->G->O-> \$648
 N->B->C->D->E->T->L->M->A->I->S->R->F->J->H->P->Q->K->G->O-> \$626
 N->B->C->D->E->T->L->M->A->I->S->R->F->J->H->P->Q->K->G->O-> \$605
 N->T->C->D->E->B->L->M->A->I->S->R->F->J->H->P->Q->O->G->K-> \$593
 N->T->C->D->E->B->L->M->A->I->S->R->F->J->Q->P->H->O->G->K-> \$556
 N->T->C->D->E->B->L->M->A->I->S->R->F->J->Q->P->H->O->G->K-> \$555
 N->T->C->D->E->B->L->H->A->M->S->R->F->J->Q->P->I->O->G->K-> \$551
 N->T->C->D->E->B->P->H->A->M->S->R->F->J->Q->L->I->O->G->K-> \$547
 N->T->C->D->E->L->P->H->A->M->S->R->F->J->Q->B->I->O->G->K-> \$541
 N->T->C->D->G->L->P->H->A->M->S->R->F->J->Q->B->I->O->E->K-> \$530
 N->D->C->T->G->L->P->H->A->M->S->R->F->J->Q->B->I->O->E->K-> \$518
 N->D->C->T->P->L->G->H->A->M->S->R->F->J->Q->B->I->O->E->K-> \$477

```

namespace Ejemplo {
    internal class Program {
        static void Main() {
            Random Azar = new Random();

            /* SUDOKU original, los ceros son los números que se deben hallar */
            int[,] Original = new int[9, 9] {
                {1, 0, 0, 0, 0, 7, 0, 9, 0},
                {0, 3, 0, 0, 2, 0, 0, 0, 8},
                {0, 0, 9, 6, 0, 0, 5, 0, 0},
                {0, 0, 5, 3, 0, 0, 9, 0, 0},
                {0, 1, 0, 0, 8, 0, 0, 0, 2},
                {6, 0, 0, 0, 0, 4, 0, 0, 0},
                {3, 0, 0, 0, 0, 0, 0, 1, 0},
                {0, 4, 0, 0, 0, 0, 0, 0, 7},
                {0, 0, 7, 0, 0, 0, 3, 0, 0}
            };

            /* Mantiene el ciclo hasta que resuelva el Sudoku */
            bool Finalizar;

            /* Lleva el número de iteraciones */
            int Ciclos = 0;

            /* Copia el SUDOKU original para trabajar en esta copia */
            int[,] Copia = new int[9, 9];

            /* Cada cuantos ciclos borra números para destrabar */
            int DESTRUYE = 700;

            /* Ciclo que llenará el sudoku completamente */
            do {
                /* Copia el sudoku original */
                for (int FilaCopia = 0; FilaCopia < 9; FilaCopia++) {
                    for (int ColumnaCopia = 0; ColumnaCopia < 9; ColumnaCopia++)
                        if (Original[FilaCopia, ColumnaCopia] != 0)
                            Copia[FilaCopia, ColumnaCopia] = Original[FilaCopia, ColumnaCopia];
                }

                bool numValido;
                int Fila, Columna, Numero;
                do {
                    /* Busca un número al azar para colocar en alguna celda */
                    Fila = Azar.Next(9); /* Una posición X de 0 a 8 */
                    Columna = Azar.Next(9); /* Una columna de 0 a 8 */
                    Numero = Azar.Next(1, 10); /* Un número al azar de 1 a 9 */
                    numValido = true;

                    /* Chequea si el número no se repite ni vertical ni horizontalmente */
                    for (int Contador = 0; Contador < 9; Contador++)
                        if (Copia[Contador, Columna] == Numero || Copia[Fila, Contador] == Numero)
                            numValido = false;

                } while (!numValido);

                /* Si el número no se repite entonces valida que no se repita dentro del cuadro interno */
                int cuadroFila = Fila - Fila % 3;
                int cuadroColumna = Columna - Columna % 3;
                for (int i = cuadroFila; i < cuadroFila + 3; i++) {
                    for (int j = cuadroColumna; j < cuadroColumna + 3; j++) {
                        if (Copia[i, j] == Numero)
                            numValido = false;
                    }
                }
            }

            /* Si todo está bien, entonces se pone el número */
            if (numValido)
                Copia[Fila, Columna] = Numero;

            /* Chequea si se completó el sudoku completamente */
            Finalizar = true;
            for (Fila = 0; Fila < 9; Fila++)
                for (Columna = 0; Columna < 9; Columna++)
                    if (Copia[Fila, Columna] == 0)
                        Finalizar = false;
        }
    }
}

```

```

/* Cada cierto número de ciclos, para desatrabar, borra la tercera parte del tablero */
Ciclos++;
if (Ciclos % DESTRUYE == 0)
    for (Fila = 0; Fila < 9; Fila++)
        for (Columna = 0; Columna < 9; Columna++)
            if (Azar.NextDouble() < 0.34)
                Copia[Fila, Columna] = 0;
} while (!Finalizar);

//Imprime el sudoku original
Console.WriteLine("Sudoku Original");
for (int Fila = 0; Fila < 9; Fila++) {
    for (int Columna = 0; Columna < 9; Columna++)
        if (Original[Fila, Columna] == 0)
            Console.Write("_ ");
        else
            Console.Write(Original[Fila, Columna] + " ");
    Console.WriteLine(" ");
}

//Imprime el sudoku resuelto
Console.WriteLine("\r\nCiclos totales usados: " + Ciclos);
Console.WriteLine("Sudoku Resuelto");
for (int Fila = 0; Fila < 9; Fila++) {
    for (int Columna = 0; Columna < 9; Columna++)
        Console.Write(Copia[Fila, Columna] + " ");
    Console.WriteLine(" ");
}
}
}
}

```

Sudoku Original

1	-	-	-	-	7	-	9	-
-	3	-	-	2	-	-	-	8
-	-	9	6	-	-	5	-	-
-	-	-	5	3	-	-	9	-
-	1	-	-	8	-	-	-	2
6	-	-	-	-	4	-	-	-
3	-	-	-	-	-	-	1	-
-	4	-	-	-	-	-	-	7
-	-	7	-	-	-	3	-	-

Ciclos totales usados: 5307473

Sudoku Resuelto

1	6	2	8	5	7	4	9	3
5	3	4	1	2	9	6	7	8
7	8	9	6	4	3	5	2	1
4	7	5	3	1	2	9	8	6
9	1	3	5	8	6	7	4	2
6	2	8	7	9	4	1	3	5
3	5	6	4	7	8	2	1	9
2	4	1	9	3	5	8	6	7
8	9	7	2	6	1	3	5	4

Ilustración 282: Resuelve el Sudoku

El problema de las tres puertas

El problema de las tres puertas, también conocido como el problema de Monty Hall, es un acertijo matemático de probabilidad que se presenta en diferentes variantes. En una de las variantes, se tienen tres puertas etiquetadas como "coche", "cabra 1" y "cabra 2". Detrás de una de las puertas se encuentra un coche, mientras que detrás de las otras dos hay cabras. El objetivo del acertijo es determinar qué puerta contiene el coche, utilizando solo una elección de puerta y sin abrir ninguna de las puertas.

La mecánica del acertijo es la siguiente: el concursante debe elegir una puerta entre tres (todas cerradas); el premio consiste en llevarse lo que se encuentra detrás de la elegida. Se sabe con certeza que tras una de ellas se oculta un automóvil, y tras las otras dos hay cabras. Una vez que el concursante haya elegido una puerta y comunicado su elección a los presentes, el presentador, que sabe lo que hay detrás de cada puerta, abrirá una de las otras dos, en la que habrá una cabra. A continuación, le da la opción al concursante de cambiar, si lo desea, de puerta (tiene dos opciones). ¿Debe el concursante mantener su elección original o escoger la otra puerta? ¿Hay alguna diferencia?

La respuesta correcta es que el concursante debe cambiar de puerta. Si el concursante cambia de puerta, la probabilidad de ganar el coche es de 2/3, mientras que, si mantiene su elección original, la probabilidad de ganar el coche es de 1/3.

I/010.cs

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            /* Generador de números pseudo-aleatorios */
            Random Azar = new();

            /* Contador de éxitos si el jugador cambia su decisión inicial */
            int CambiaDecision = 0;

            /* Contador de éxitos si el jugador insiste en mantener su decisión inicial */
            int InsisteNOCambiar = 0;

            for (int prueba = 1; prueba <= 1000000; prueba++) {
                /* Selecciona una puerta al azar que será la ganadora */
                int puertaGanadora = Azar.Next(3);

                /* El jugador selecciona una puerta al azar */
                int puertaJugador = Azar.Next(3);

                /* Si el jugador escogió por casualidad la puerta ganadora,
                   el dueño escoge al azar alguna de las dos puertas restantes */
                int puertaDueño;
                if (puertaGanadora == puertaJugador)
                    do {
                        puertaDueño = Azar.Next(3);
                    } while (puertaDueño == puertaGanadora);
                else
                    /* El dueño sólo puede escoger la puerta que no tiene premio */
                    do {
                        puertaDueño = Azar.Next(3);
                    } while (puertaDueño == puertaGanadora || puertaDueño == puertaJugador);

                /* El jugador NO cambia su elección */
                if (puertaGanadora == puertaJugador) InsisteNOCambiar++;

                /* El jugador SI cambia su elección */
                int nuevapuerta;
                do {
                    nuevapuerta = Azar.Next(3);
                } while (nuevapuerta == puertaDueño || nuevapuerta == puertaJugador);
                if (nuevapuerta == puertaGanadora) CambiaDecision++;

            }

            Console.WriteLine("Número de aciertos");
            Console.WriteLine("SIN cambiar la elección inicial: " + InsisteNOCambiar);
            Console.WriteLine("CAMBIANDO la elección inicial: " + CambiaDecision);
        }
    }
}
```

Número de aciertos

SIN cambiar la elección inicial: 333322

CAMBIANDO la elección inicial: 666678

Ilustración 283: Problema de las tres puertas

Número de aciertos

SIN cambiar la elección inicial: 333986

CAMBIANDO la elección inicial: 666014

Ilustración 284: Problema de las tres puertas

Área bajo la curva

El área bajo la curva es un concepto matemático que se utiliza para calcular el área de una región limitada por una curva y el eje x en un intervalo cerrado. Para encontrar el área bajo una curva, se utilizan integrales definidas. En general, el proceso para encontrar el área bajo una curva implica los siguientes pasos:

1. Formar una integral definida con la ecuación $Y=F(X)$ dada.
2. Obtener la integral de la función.
3. Se evalúan los límites superior e inferior en la expresión integrada.
4. Se obtiene la diferencia de los valores obtenidos del límite superior y límite inferior.

¿Pero en el caso que la integral definida sea muy compleja de resolver? En ese caso, se hace uso del siguiente método

Paso 1: Tiene la ecuación $Y=F(X)$

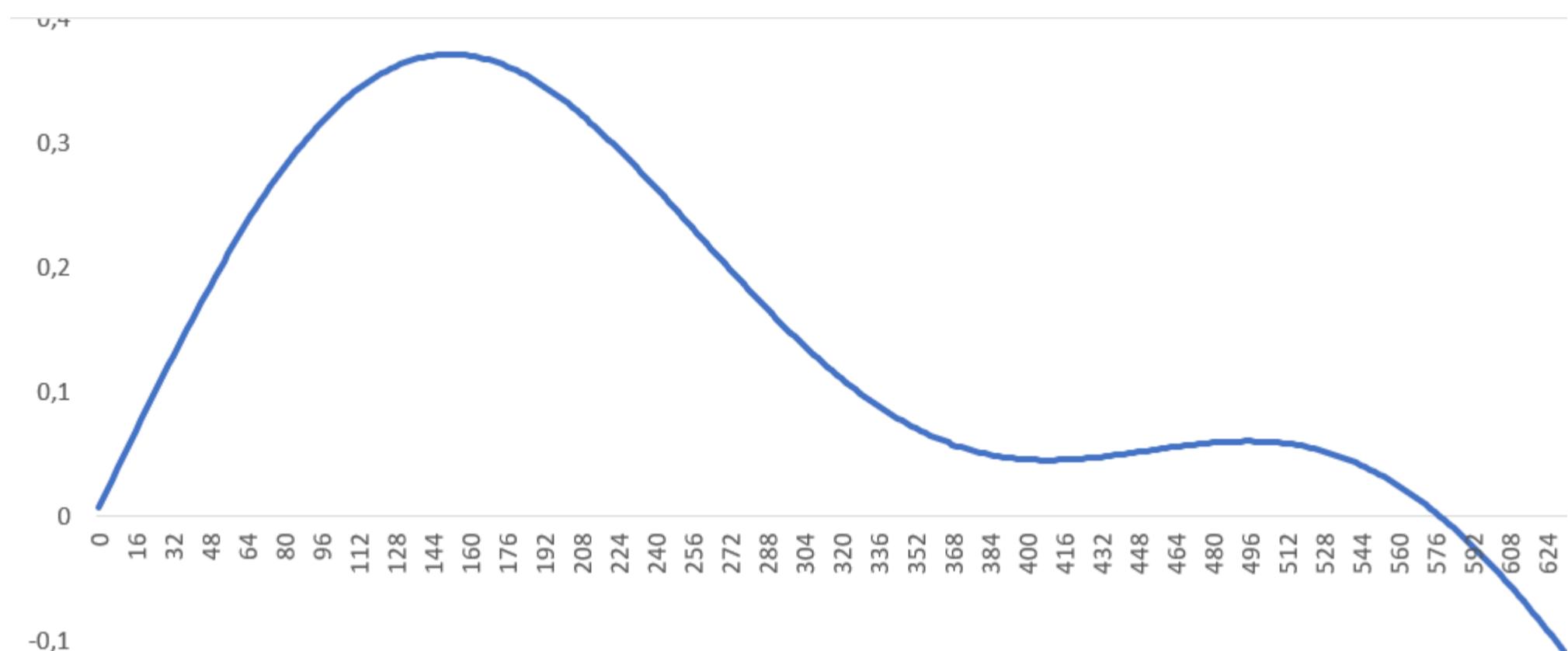


Ilustración 285: Curva $Y = F(X)$

Paso 2: Se dibujan los límites X mínimo y X máximo para calcular el área, con eso se tiene el tamaño de la base

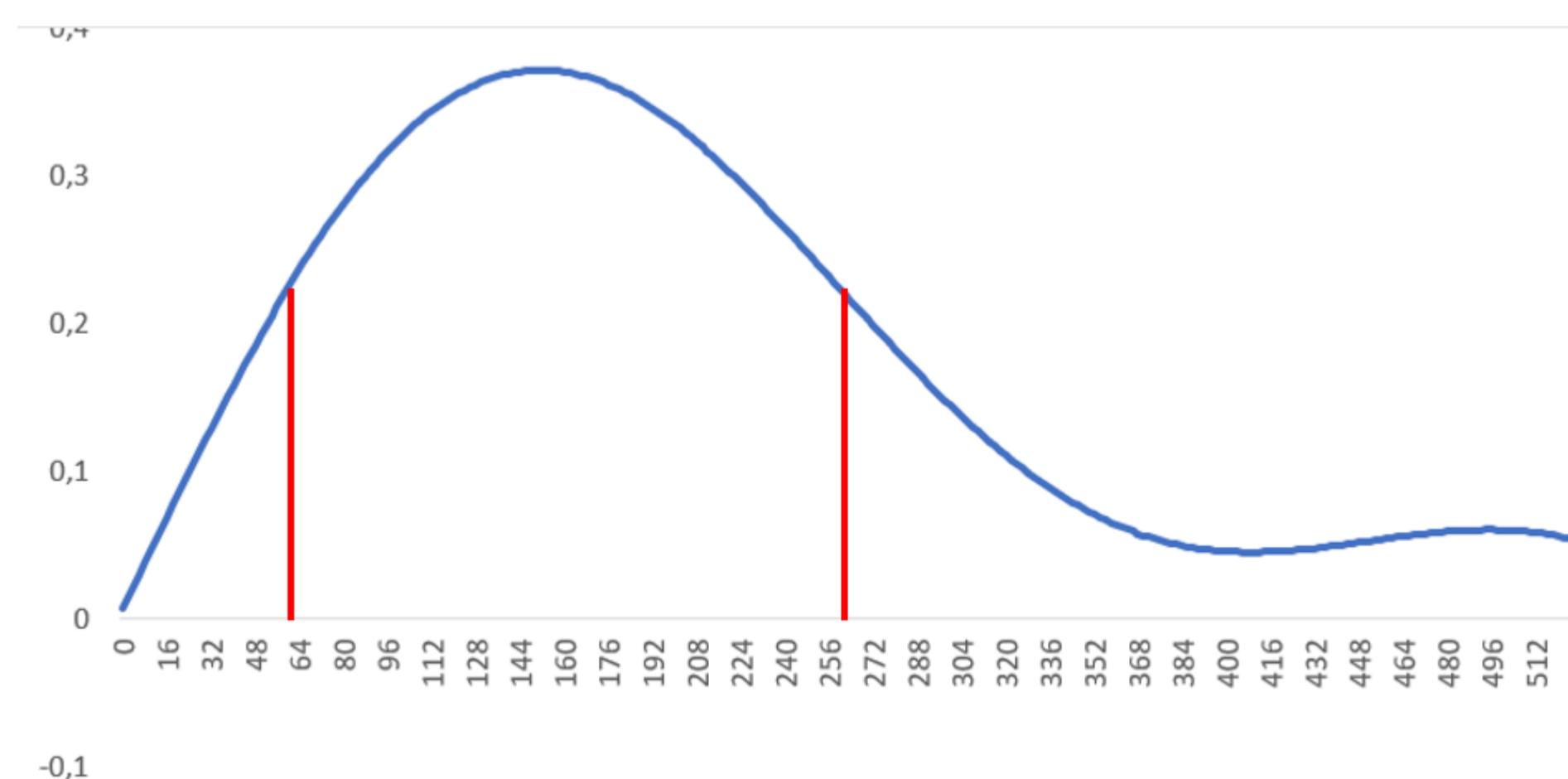


Ilustración 286: Los límites X_{\min} y X_{\max} para hallar el área interna

Paso 3: Se dibuja un rectángulo que contenga la curva en su interior. Luego ya se tiene el valor del tamaño de la base y la altura del rectángulo (que sería el mayor valor de Y entre X mínimo y Xmáximo)

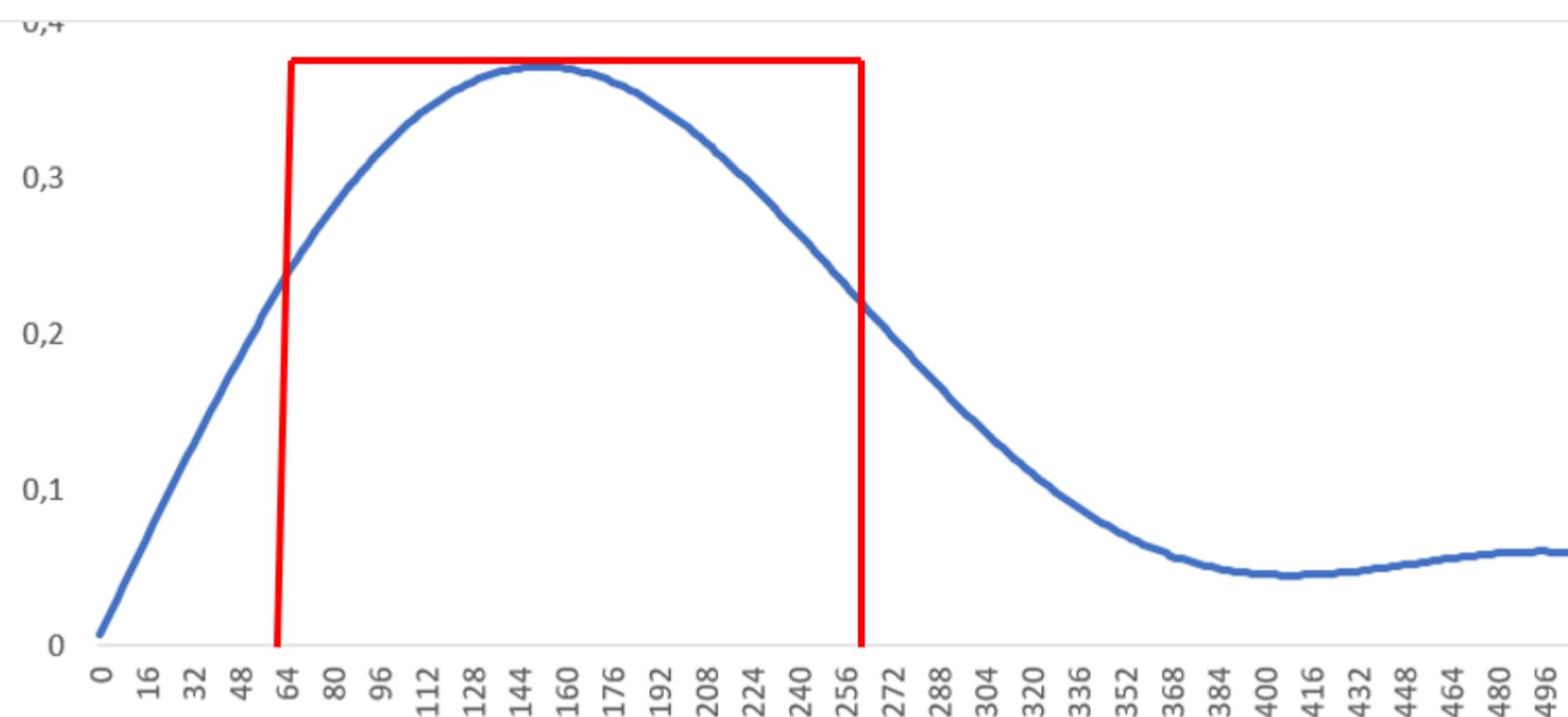


Ilustración 287: Se dibuja un rectángulo que cubra la curva

Paso 4: Se lanzan muchos puntos al azar al interior del rectángulo

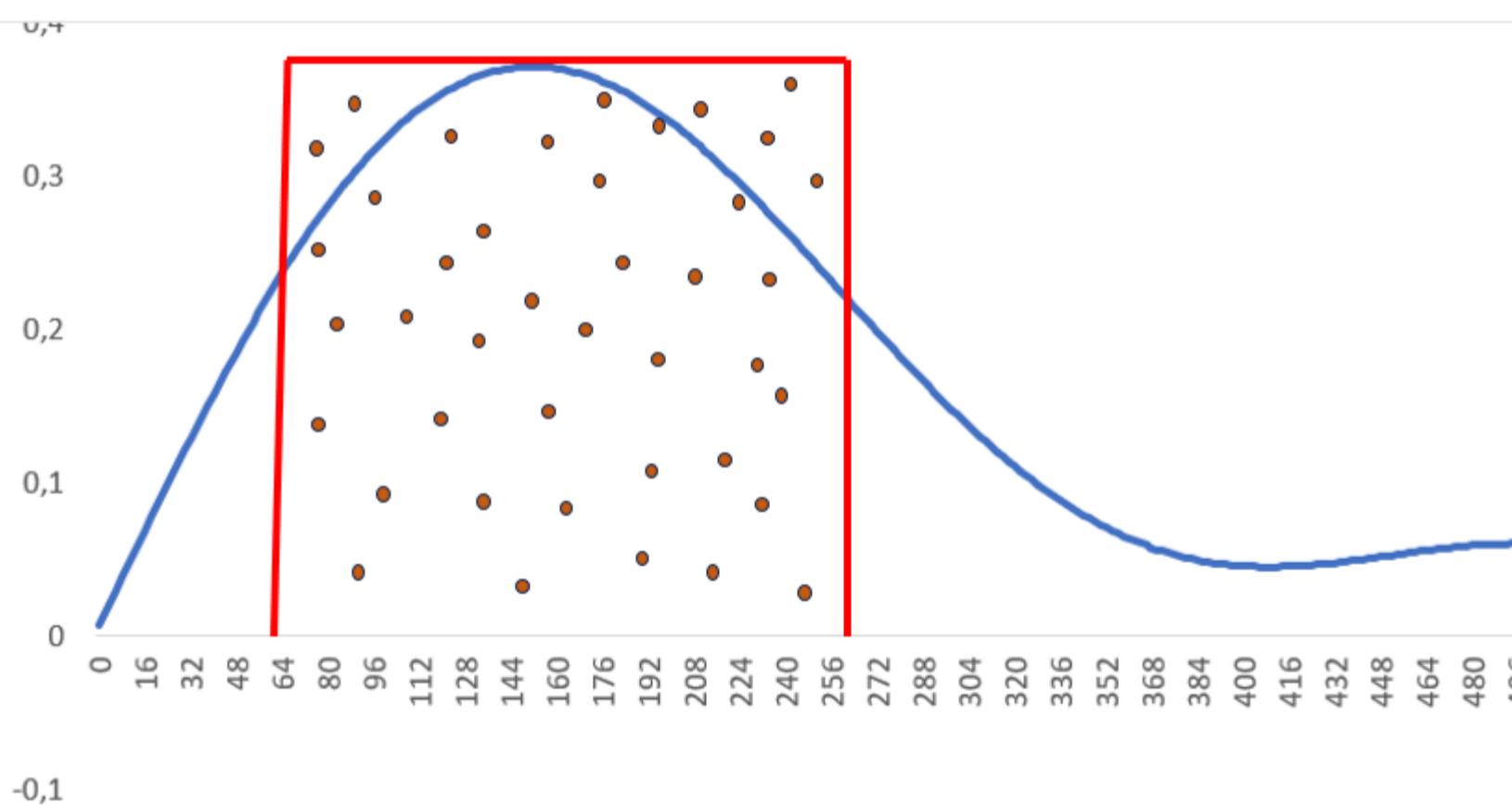


Ilustración 288: Se lanzan puntos al azar al interior del rectángulo

Paso 5: Se calcula el área bajo la curva con la siguiente ecuación:

$$\text{Área bajo la curva} = \text{Área del rectángulo} * \frac{\text{Total puntos que estén al interior de la curva}}{\text{Total puntos lanzados}}$$

```

namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Área bajo la curva

            //Mínimo valor en X, máximo valor en X
            double MinValX = -5;
            double MaxValX = 5;

            //Validar que no hayan puntos de corte
            if (HayPuntosCorte(MinValX, MaxValX))
                Console.WriteLine("Hay puntos de corte, no es válido para hallar el área");
            else {
                double ExtremoY = MaxMinY(MinValX, MaxValX);
                double Area = CalculaArea(MinValX, MaxValX, ExtremoY);
                Console.WriteLine("Área es: " + Area);
            }
        }

        //Retorna true si hay puntos de corte entre los valores de X mínimo y máximo dados
        static public bool HayPuntosCorte(double MinX, double MaxX) {
            int Positivos = 0;
            int Negativos = 0;
            for (double X = MinX; X <= MaxX; X += 0.001) {
                double Y = Ecuacion(X);
                if (Y > 0) Positivos++;
                if (Y < 0) Negativos++;
            }
            if (Positivos != 0 && Negativos != 0) return true;
            return false;
        }

        //Retorna el mínimo valor de Y (si el área está por debajo del eje X)
        //o el máximo valor de Y (si el área está por encima del eje X)
        static public double MaxMinY(double MinX, double MaxX) {
            double MaximoY = double.MinValue;
            double MinimoY = double.MaxValue;
            bool Orienta = false;
            for (double X = MinX; X <= MaxX; X += 0.001) {
                double Y = Ecuacion(X);
                if (Y > 0) Orienta = true;
                if (Y > MaximoY) MaximoY = Y;
                if (Y < MinimoY) MinimoY = Y;
            }
            if (Orienta) return MaximoY;
            return MinimoY;
        }

        //Calcula el área bajo la curva usando el método Monte Carlo. Siguiendo las directrices
        matemáticas,
        //el área tendrá valor positivo si la curva está por encima del eje X,
        //el área tendrá valor negativo si la curva está por debajo del eje X
        static public double CalculaArea(double MinX, double MaxX, double ExtremoY) {
            Random Azar = new();
            int PuntosDentro = 0;
            int PuntosTotal = 7000000;
            for (int puntos = 1; puntos <= PuntosTotal; puntos++) {
                double Xazar = Azar.NextDouble() * (MaxX - MinX) + MinX;
                double Yazar = Azar.NextDouble() * ExtremoY;
                double Y = Ecuacion(Xazar);
                if (Yazar <= Y && ExtremoY > 0) PuntosDentro++;
                if (Yazar >= Y && ExtremoY < 0) PuntosDentro++;
            }
            double AreaTotal = (MaxX - MinX) * ExtremoY * (double)PuntosDentro / PuntosTotal;
            return AreaTotal;
        }

        static public double Ecuacion(double X) {
            double Y = -1 * Math.Abs(Math.Cos(X) - Math.Sin(X) * X);
            return Y;
        }
    }
}

```

Área es: -18,23635566437144

DE LOS CREADORES DE WOLFRAM LANGUAGE Y MATHEMATICA



integrate Y = -1 * abs(cos(X) - sin(X) * X) from X= -5 to 5



LENGUAJE NATURAL

ENTRADA MATEMÁTICA

TECLADO EXTENDIDO

EJEMPLOS

CARGAR

ALEATORIO

Integral definida

Más dígitos

Solución paso a paso

$$\begin{aligned} \int_{-5}^5 -|\cos(x) - \sin(x)x| dx &= \\ -2\left(\cos\left(\sqrt{\cos(x) - x \sin(x)} \text{ cercana a } x = -3,42562\right)\right. & \\ \left. - \cos\left(\sqrt{\cos(x) - x \sin(x)} \text{ cercana a } x = -0,860334\right)\right. & \\ \left. + \cos\left(\sqrt{\cos(x) - x \sin(x)} \text{ cercana a } x = 0,860334\right)\right. & \\ \left. - \cos\left(\sqrt{\cos(x) - x \sin(x)} \text{ cercana a } x = 3,42562\right)\right. & \\ \cos(5) \approx -18,234 & \end{aligned}$$

Dos robots se encuentran

En un arreglo bidimensional se ubican dos robots, cada uno en una casilla seleccionada al azar. No tienen sensores visuales, luego cada robot no sabe donde está el otro robot. La simulación es averiguar en cuál escenario es más probable que haya un encuentro de los dos robots:

1. Un robot quieto mientras el otro se mueve al azar por el tablero.
2. Los dos robots moviéndose al tiempo al azar por el tablero.

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            /* Encuentro de dos robots sin sensor visual en un tablero.
             * ¿Cuál es la mejor estrategia?
             * 1. Un robot quieto y el otro moviéndose.
             * 2. Los dos robots moviéndose.
            */

            Random Azar = new();

            //Tamaño del tablero
            int Filas = 50;
            int Columnas = 50;

            int MueveEstrategia1 = 0;
            int MueveEstrategia2 = 0;
            for (int Pruebas = 1; Pruebas <= 1000; Pruebas++) {
                //Robot A
                int FilRobotA = Azar.Next(Filas);
                int ColRobotA = Azar.Next(Columnas);

                //Robot B
                int FilRobotB, ColRobotB;
                do {
                    FilRobotB = Azar.Next(Filas);
                    ColRobotB = Azar.Next(Columnas);
                } while (FilRobotA == FilRobotB || ColRobotA == ColRobotB);

                MueveEstrategia1 += Estrategia1(Azar, FilRobotA, ColRobotA, FilRobotB, ColRobotB,
                    Filas, Columnas);
                MueveEstrategia2 += Estrategia2(Azar, FilRobotA, ColRobotA, FilRobotB, ColRobotB,
                    Filas, Columnas);
            }

            Console.WriteLine("Total movimientos para: Un robot quieto (B) y el otro moviéndose (A): {0:0,0}", MueveEstrategia1);
            Console.WriteLine("Total movimientos para: Los dos robots movéndose: {0:0,0}" , MueveEstrategia2);
        }

        static int Estrategia1(Random Azar, int FilRobotA, int ColRobotA, int FilRobotB, int ColRobotB,
            int Filas, int Columnas) {
            //Estrategia 1. Un robot quieto (B) y el otro moviéndose (A)
            int TotalMovimientos = 0;
            do {
                TotalMovimientos++;
                int FilaMueve, ColumnaMueve;
                do {
                    FilaMueve = Azar.Next(-1, 2);
                    ColumnaMueve = Azar.Next(-1, 2);
                } while (FilRobotA + FilaMueve < 0 || FilRobotA + FilaMueve >= Filas ||
                    ColRobotA + ColumnaMueve < 0 || ColRobotA + ColumnaMueve >= Columnas ||
                    (FilaMueve == 0 && ColumnaMueve == 0));
                FilRobotA += FilaMueve;
                ColRobotA += ColumnaMueve;
            } while (FilRobotA != FilRobotB || ColRobotA != ColRobotB);
            return TotalMovimientos;
        }

        static int Estrategia2(Random Azar, int FilRobotA, int ColRobotA, int FilRobotB, int ColRobotB,
            int Filas, int Columnas) {
            //Estrategia 2. Los dos robots movéndose
            int TotalMovimientos = 0;
            do {
                TotalMovimientos++;
                //Mueve Robot A
                int FilaMueve, ColumnaMueve;
                do {
                    FilaMueve = Azar.Next(-1, 2);
                    ColumnaMueve = Azar.Next(-1, 2);
                } while (FilRobotA + FilaMueve < 0 || FilRobotA + FilaMueve >= Filas ||
                    ColRobotA + ColumnaMueve < 0 || ColRobotA + ColumnaMueve >= Columnas ||
                    (FilaMueve == 0 && ColumnaMueve == 0));
                FilRobotA += FilaMueve;
                ColRobotA += ColumnaMueve;
            } while (FilRobotA != FilRobotB || ColRobotA != ColRobotB);
            return TotalMovimientos;
        }
    }
}
```

```

int A_FilaMueve, A_ColumnaMueve;
do {
    A_FilaMueve = Azar.Next(-1, 2);
    A_ColumnaMueve = Azar.Next(-1, 2);
} while (FilRobotA + A_FilaMueve < 0 || FilRobotA + A_FilaMueve >= Filas ||
         ColRobotA + A_ColumnaMueve < 0 || ColRobotA + A_ColumnaMueve >= Columnas ||
         (A_FilaMueve == 0 && A_ColumnaMueve == 0));
FilRobotA += A_FilaMueve;
ColRobotA += A_ColumnaMueve;

//Mueve RobotB
int B_FilaMueve, B_ColumnaMueve;
do {
    B_FilaMueve = Azar.Next(-1, 2);
    B_ColumnaMueve = Azar.Next(-1, 2);
} while (FilRobotB + B_FilaMueve < 0 || FilRobotB + B_FilaMueve >= Filas ||
         ColRobotB + B_ColumnaMueve < 0 || ColRobotB + B_ColumnaMueve >= Columnas ||
         (B_FilaMueve == 0 && B_ColumnaMueve == 0));
FilRobotB += B_FilaMueve;
ColRobotB += B_ColumnaMueve;

} while (FilRobotA != FilRobotB || ColRobotA != ColRobotB);
return TotalMovimientos;
}
}
}

```

Consola de depuración de Microsoft Visual Studio

Total movimientos para: Un robot quieto (B) y el otro moviéndose (A): 6.473.617
Total movimientos para: Los dos robots moviéndose: 4.752.886

Ilustración 289: Dos robots moviéndose

Parte 10: Algoritmos evolutivos

El problema

Está la siguiente cadena:

Un ejemplo de cadena de caracteres

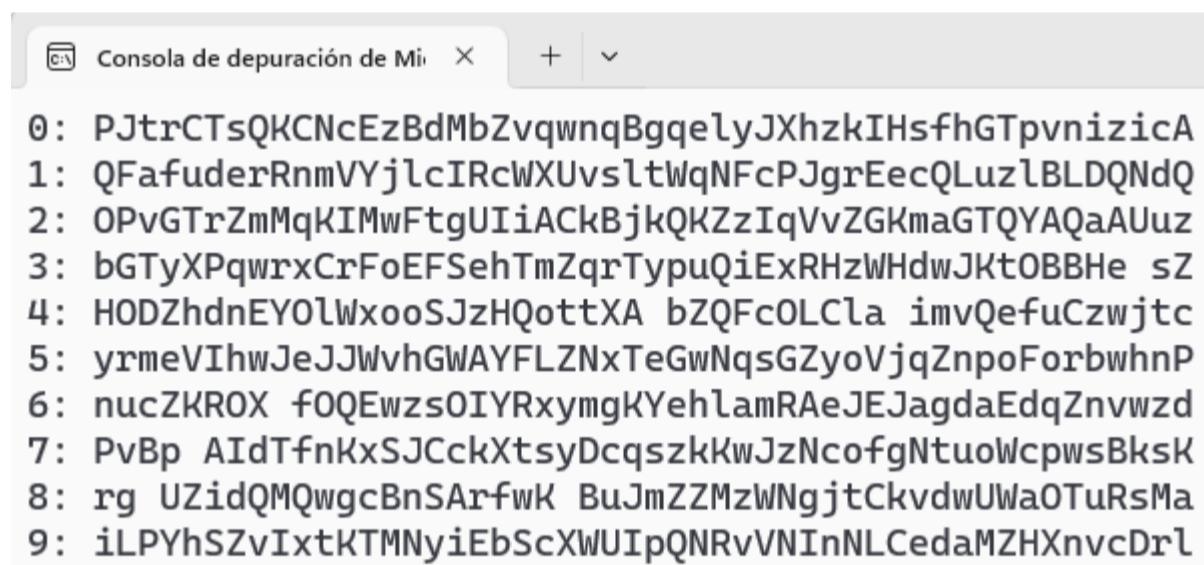
¿Es posible generar cadenas de caracteres al azar de tal manera que coincida con la cadena anterior? En realidad, sí, pero es altamente improbable que la cadena generada al azar acierte a esa cadena en particular. Igualmente se prueba.

El primer paso es generar cadenas al azar. Este sería el código:

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            Random Azar = new();

            //Imprime 10 cadenas al azar de 50 caracteres cada una
            for (int cont = 0; cont < 10; cont++)
                Console.WriteLine(cont.ToString() + ":" + CadenaAzar(Azar, 50));
        }

        //Retorna una cadena al azar
        static string CadenaAzar(Random Azar, int Longitud) {
            string Permitido = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ ";
            string Cadena = "";
            for (int Contador = 1; Contador <= Longitud; Contador++)
                Cadena += Permitido[Azar.Next(Permitido.Length)];
            return Cadena;
        }
    }
}
```



The screenshot shows a command prompt window titled 'Consola de depuración de Mi'. It displays 10 lines of randomly generated strings, each starting with a number from 0 to 9 followed by a colon and a 50-character string of lowercase letters and uppercase letters. The strings are generated using the provided C# code.

```
0: PJtrCTsQKCNCezBdMbZvqwnqBqgelyJXhzkIHsfhGTpvnizicA
1: QFafuderRnmVYjlcIRcWXUvsltWqNFCPJgrEecQLuzlBLDQNdQ
2: OPvGTrZmMqKIMwFtgUIiACKBjkQKZzIqVvZGKmaGTQYAQaAUuz
3: bGTyXPqwrxCrFoEFSehTmZqrTypuQiExRHzzWHzJktOBBAHe sZ
4: HODZhdnEYOlWxooSJzHQottXA bZQFcOLCla imvQefuCzwjtc
5: yrmeVIhwJeJJWvhGWAYFLZNxTeGwNqsGZyoVjqZnpoForbwhnP
6: nucZKROX fOQEwzsOIYRxymgKYehlamRAeJEJagdaEdqZnvwzd
7: PvBp AIdTfnKxSJCckXtsyDcqszkKwJzNcofgNtuowcpwsBksK
8: rg UZidQMwgcBnSArfwK BuJmZZMzWNgtCkvdwUWaOTuRsMa
9: iLPYhSzvIxtKTMNyieBScXWUIpQRvVNInNLCedaMZHXnvcDrl
```

Ilustración 290: Generar cadenas al azar

En la función CadenaAzar se generan las cadenas con minúsculas, mayúsculas y el espacio, pero no la letra Ñ o ñ. Luego es una limitante porque si la cadena original tuviese ese carácter Ñ, nunca se habría encontrado una coincidencia. Podría decirse que se usara el ASCII, pero de nuevo, si la cadena original tiene un carácter en otro idioma como el japonés (que no aparece en el ASCII) entonces jamás se encontraría una coincidencia.

Para el algoritmo de dar con la cadena original generando cadenas al azar, este sería el código:

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Único generador de números pseudo-aleatorios
            Random Azar = new();

            //La cadena original
            string cadenaOriginal = "Esta es una prueba de texto";

            //Genera cadenas al azar y ver si en algún momento hay coincidencia
            int longitud = cadenaOriginal.Length;

            //Ciclo para generar cadenas al azar
            for (int Contador = 1; Contador <= 10000; Contador++) {
                string nuevaCadena = CadenaAzar(Azar, longitud);
                if (nuevaCadena == cadenaOriginal)
                    Console.WriteLine("Acertó");
            }

            Console.WriteLine("Finaliza");
        }

        //Retorna una cadena al azar
        static string CadenaAzar(Random Azar, int Longitud) {
            string Permitido = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ ";
            string Cadena = "";
            for (int Contador = 1; Contador <= Longitud; Contador++)
                Cadena += Permitido[Azar.Next(Permitido.Length)];
            return Cadena;
        }
    }
}
```

No importa que tan grande ponga el ciclo (en el ejemplo, llega a 10000), es supremamente improbable que acierte la cadena generada al azar a la cadena original. En otras palabras, es muy improbable que salga el aviso "Acertó" en la ejecución del programa.

Los algoritmos evolutivos solucionan ese problema e inspirados con la Teoría de la Evolución de las Especies, se agrega un nuevo ingrediente: la selección natural.

El operador Mutación

El algoritmo sería así:

```
Algoritmo Evolutivo
Inicio
    Generar población de N individuos al azar
    Inicio ciclo
        Seleccionar al azar dos individuos de esa población: A y B
        Evaluar adaptación de A
        Evaluar adaptación de B
        Si adaptación de A es mejor que adaptación de B entonces
            Eliminar individuo B
            Duplicar individuo A
            Modificar levemente al azar el nuevo duplicado
        de lo contrario
            Eliminar individuo A
            Duplicar individuo B
            Modificar levemente al azar el nuevo duplicado
        Fin Si
    Fin ciclo
    Buscar individuo con mejor adaptación de la población
    Imprimir individuo
Fin
```

Explicación del algoritmo

“Generar población de N individuos al azar”: Generar N cadenas al azar y guardarlas en un arreglo o lista

“Evaluar adaptación de”: Evaluar cuantitativamente esa cadena con respecto a la original. En ese caso, una medida puede ser sumar un punto por cada letra que coincida con la cadena original. A mayor puntaje, mejor es la adaptación.

Al principio se crea una población con varios individuos generados al azar, por ejemplo, con 10 individuos:

```
1: oTGBVáDgzD!óxëVoÓsbYAOVVWkTj
2: Áh?JÄgÉmmÍ; !óQUbFtdfymNtNSjmF
3: rEVáa Z;oaÓC;rxy?Äx;pÑjíHmfGÓ
4: ÜkJñzU;mCoQEZe! ÄmúÖN;Ghj zoá
5: tWsLótMRFhoB; iÄpkkDgYq; LrXëÓU
6: LFopAïéÓüYj !ëÖÄvÚMyccÄíïFPLÜ?
7: ÑyérMQwlgMXaYTgúGsFfQÑIQBÜsib
8: jOYnÚtAEH; È; HVvPB ÁëÜáx; fuÜch
9: ñróQEx; MqöúüEKÜiNTpRBóBWúofsö
10: èWÄUPLúiïBVëÁÑLaómpKPÓ! ïWXYÚQ
```

Se seleccionan dos individuos al azar, por ejemplo, el 3 y el 9:

```
1: oTGBVáDgzD!óxëVoÓsbYAOVVWkTj
2: Áh?JÄgÉmmÍ; !óQUbFtdfymNtNSjmF
3: rEVáa Z;oaÓC;rxy?Äx;pÑjíHmfGÓ
4: ÜkJñzU;mCoQEZe! ÄmúÖN;Ghj zoá
5: tWsLótMRFhoB; iÄpkkDgYq; LrXëÓU
6: LFopAïéÓüYj !ëÖÄvÚMyccÄíïFPLÜ?
7: ÑyérMQwlgMXaYTgúGsFfQÑIQBÜsib
8: jOYnÚtAEH; È; HVvPB ÁëÜáx; fuÜch
9: ñróQEx; MqöúüEKÜiNTpRBóBWúofsö
10: èWÄUPLúiïBVëÁÑLaómpKPÓ! ïWXYÚQ
```

Ahora se evalúa cual individuo, de los dos seleccionados, es el mejor. Una forma de hacerlo es comparar letra por letra entre la cadena del individuo y la cadena original, si coinciden entonces suma 1 al puntaje.

Evaluar: rEVáa Z;oaÓC;rxy?Äx;pÑjíHmfGÓ vs Esta es una prueba de texto Es 1

Evaluar: ñróQEx; MqöúüEKÜiNTpRBóBWúofsö vs Esta es una prueba de texto Es 0

El individuo ganador es rEVáa Z;oaÓC;rxy?Äx;pÑjíHmfGÓ con puntaje de 1, el otro pierde con un puntaje de 0

El individuo perdedor es eliminado de la población

```

1: oTGBVáDgzD!óxëVoÓsbYAOVVWkTj
2: Áh?JÄgÉmmÍ; !óQUbFtdfymNtNSjmF
3: rEVáa Z;oaÓC;rxy?Äx;pÑjíHmfGÓ
4: ÜkJñzU;mCoQEZe! ÄmúÖN;Ghj zoá
5: tWsLótMRFhoB; iÄpkkDgYq; LrXëÓU
6: LFopAïéÓüYj !ëÖAvÚMyccÄíïFPLÜ?
7: ÑyérMQwlgMXaYTgúGsFfQÑIQBÜsib
8: jOYnÚtAEH;E;HVvPB ÁëÜáx;fuÜch
9: ñréQEx;MqöúïEKÜiNTpRBóBWúefsö
10: ëWÄUPLúiïBVëÁÑLaómpKPÓ! ïWXYÚQ

```

El mejor individuo se premia creando una copia de sí mismo

```
3: rEVáa Z;oaÓC;rxy?Äx;pÑjíHmfGÓ
```

Ahora esa copia se modifica en algún punto al azar

```
?: rEVáa ZhaoÓC;rxy?Äx;pÑjíHmfGÓ
```

Esa copia modificada ahora hace parte de la población ocupando el espacio dejado por el perdedor

```

1: oTGBVáDgzD!óxëVoÓsbYAOVVWkTj
2: Áh?JÄgÉmmÍ; !óQUbFtdfymNtNSjmF
3: rEVáa Z;oaÓC;rxy?Äx;pÑjíHmfGÓ
4: ÜkJñzU;mCoQEZe! ÄmúÖN;Ghj zoá
5: tWsLótMRFhoB; iÄpkkDgYq; LrXëÓU
6: LFopAïéÓüYj !ëÖAvÚMyccÄíïFPLÜ?
7: ÑyérMQwlgMXaYTgúGsFfQÑIQBÜsib
8: jOYnÚtAEH;E;HVvPB ÁëÜáx;fuÜch
9: rEVáa ZhaoÓC;rxy?Äx;pÑjíHmfGÓ
10: ëWÄUPLúiïBVëÁÑLaómpKPÓ! ïWXYÚQ

```

El proceso se repite varias veces hasta que se encuentra el mejor individuo que soluciona el problema.

J/001.cs

```

using System.Text;

namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Sólo operador mutación
            Poblacion pobl = new();
            String Builder Cadena = new();
            Cadena.Append("esta es una prueba de algoritmo evolutivo");

            int TotalPoblacion = 100; //Una población muy grande hace que tarde más en encontrar la
solución
            int TotalCiclos = 90000;
            pobl.Proceso(Cadena, TotalPoblacion, TotalCiclos);
        }
    }

    //Cómo es el individuo
    internal class Individuo {
        public String Builder Genotipo;
        private readonly String Builder Letras = new
String Builder("abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ áéúíóúÁÉÍÓÚÑñ¿;!äëöüÄËÏÖÜ");
        //Al nacer, tendrá una serie de caracteres al azar
        public Individuo(Random Azar, int Longitud) {
            Genotipo = new String Builder();
            for (int Contador = 1; Contador <= Longitud; Contador++)

```

```

        Genotipo.Append(Letras[Azar.Next(Letras.Length)]);
    }

    //Cambia una letra del genotipo al azar
    public void Muta(Random Azar) {
        int Posicion = Azar.Next(Genotipo.Length);
        Genotipo[Posicion] = Letras[Azar.Next(Letras.Length)];
    }

    //Compara el genotipo con la cadena a la que debe parecerse
    public int Evalua(StringBuilder Original) {
        int Puntaje = 0;
        for (int Contador = 0; Contador < Original.Length; Contador++)
            if (Original[Contador] == Genotipo[Contador])
                Puntaje++;
        return Puntaje;
    }
}

//La población
internal class Poblacion {
    public List<Individuo> Individuos = new List<Individuo>();
    private Random Azar = new Random();

    public void Proceso(StringBuilder Original, int TotalIndividuos, int TotalCiclos) {
        //Genera la población
        Individuos.Clear();
        for (int Contador = 0; Contador <= TotalIndividuos; Contador++)
            Individuos.Add(new Individuo(Azar, Original.Length));

        //El proceso evolutivo
        for (int Contador = 0; Contador <= TotalCiclos; Contador++) {
            //Seleccionar al azar dos individuos de esa población: A y B
            int PosA = Azar.Next(Individuos.Count);
            int PosB;
            do {
                PosB = Azar.Next(Individuos.Count);
            } while (PosB == PosA);

            int PuntajeA = Individuos[PosA].Evalua(Original); //Evaluar adaptación de A
            int PuntajeB = Individuos[PosB].Evalua(Original); //Evaluar adaptación de B

            //Si adaptación de A es mejor que adaptación de B entonces
            if (PuntajeA > PuntajeB) {
                //Eliminar individuo B y duplicar individuo A
                Individuos[PosB].Genotipo = new StringBuilder(Individuos[PosA].Genotipo.ToString());
                //Modificar levemente al azar el nuevo duplicado
                Individuos[PosB].Muta(Azar);
            }
            else {
                //Eliminar individuo A y duplicar individuo B
                Individuos[PosA].Genotipo = new StringBuilder(Individuos[PosB].Genotipo.ToString());
                //Modificar levemente al azar el nuevo duplicado
                Individuos[PosA].Muta(Azar);
            }
        }

        //Buscar individuo con mejor adaptación de la población
        int MejorPuntaje = 0;
        int MejorIndivid = 0;
        for (int indiv = 0; indiv < Individuos.Count; indiv++) {
            int Puntaje = Individuos[indiv].Evalua(Original);
            if (Puntaje > MejorPuntaje) {
                MejorPuntaje = Puntaje;
                MejorIndivid = indiv;
            }
        }

        //Imprime el mejor individuo
        Console.WriteLine("Mejor individuo: " + Individuos[MejorIndivid].Genotipo);
        Console.WriteLine("Puntaje: " + MejorPuntaje);
    }
}
}

```

```
Consola de depuración de Mi... + ▾
Mejor individuo: esta es una prueba de algoritmo evolutivo
Puntaje: 41

C:\Users\engin\source\repos\Ejemplo\Ejemplo\bin\Debug\net8.0\E...
Para cerrar automáticamente la consola cuando se detiene la depuración.
Cerrar la consola automáticamente al detenerse la depuración.
Presione cualquier tecla para cerrar esta ventana. . .
```

Ilustración 291: Algoritmo evolutivo con el operador mutación

Después de varios ciclos, se logró dar con la cadena.

El operador Cruce

Dada la cadena de destino:

```
String original = "¡Esta es una prueba de texto!";
```

Se genera una población donde cada individuo es una cadena, por ejemplo:

```
1: oTGBVáDgzD!óxëVoÓsbYAOVVRwkTj
2: Áh?JÄgÉmmÍ; !óQUbFtdfymNtNSjmF
3: rEVáaqZ;oaÓC;rxy?Äx;pÑjíHmfGÓ
4: ÜkJñzU;mCoQEZe! ÄmúÖN;Ghj zoá
5: tWsLótMRFhoB;iÄpkkDgYq;LrXéÓU
6: LFopAïéÓüYj!ëÖÄvÚMyCÄÍïFPLÜ?
7: ÑyérMQwlglgMXaYTgúGsFfQÑIQBÜsib
8: jOYnÚtAEH;Ë;HVvPB ÁeÜáx;fuÜch
9: ñróQEx;MqöúiEKÜiNTpRBÓBWúofsö
10: ëWÄUPLúiïBVëÁÑLaómpKPÓ! ïWXYÚQ
```

Se seleccionan dos individuos al azar

```
1: oTGBVáDgzD!óxëVoÓsbYAOVVRwkTj
2: Áh?JÄgÉmmÍ; !óQUbFtdfymNtNSjmF
3: rEVáaqZ;oaÓC;rxy?Äx;pÑjíHmfGÓ
4: ÜkJñzU;mCoQEZe! ÄmúÖN;Ghj zoá
5: tWsLótMRFhoB;iÄpkkDgYq;LrXéÓU
6: LFopAïéÓüYj!ëÖÄvÚMyCÄÍïFPLÜ?
7: ÑyérMQwlglgMXaYTgúGsFfQÑIQBÜsib
8: jOYnÚtAEH;Ë;HVvPB ÁeÜáx;fuÜch
9: ñróQEx;MqöúiEKÜiNTpRBÓBWúofsö
10: ëWÄUPLúiïBVëÁÑLaómpKPÓ! ïWXYÚQ
```

Esos individuos generan un tercero con la operación de cruce: Al azar se escoge un punto de corte, la primera parte la pone un individuo y el resto el otro individuo seleccionado (en fondo verde son las partes de los individuos que constituirán el nuevo individuo)

rEVáaqZ;oaÓC;rxy?Äx;pÑjíHmfGÓ Y ñróQEx;MqöúiEKÜiNTpRBÓBWúofsö

Nuevo individuo al unir las dos piezas: rEVáax;MqöúiEKÜiNTpRBÓBWúofsö

Este nuevo individuo es evaluado y si es mejor que alguno de los dos escogidos anteriormente, entonces el nuevo individuo reemplaza al que le gane a esos dos progenitores. El proceso se repite varias veces hasta que se encuentra el mejor individuo que soluciona el problema.

J/002.cs

```
using System.Text;

namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Sólo operador cruce
            Poblacion pobl = new();
            StringBuilder Cadena = new();
            Cadena.Append("Estoy probando un algoritmo evolutivo con el operador cruce");

            int TotalPoblacion = 3000; //La población debe ser grande para que haya una gran variedad
            int TotalCiclos = 130000;
            pobl.Proceso(Cadena, TotalPoblacion, TotalCiclos);
        }
    }

    //Cómo es el individuo
    internal class Individuo {
        public StringBuilder Genotipo;
        private readonly StringBuilder Letras = new
        StringBuilder("abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ áéúíóúÁÉÍÓÚÑñ¿;!äëïöüÄËÏÖÜ");
        //Al nacer, tendrá una serie de caracteres al azar
        public Individuo(Random Azar, int Longitud) {
    }
```

```

Genotipo = new StringBuilder();
for (int Contador = 1; Contador <= Longitud; Contador++)
    Genotipo.Append(Letras[Azar.Next(Letras.Length)]);
}

//Nace de un proceso de cruce, luego recibe dos genotipos
public Individuo(Random Azar, StringBuilder GenotipoA, StringBuilder GenotipoB) {
    //Una posición al azar de dónde corta de un genotipo y otro
    int Pos = Azar.Next(GenotipoA.Length);

    Genotipo = new StringBuilder();

    //Agrega el genotipo del primer progenitor
    for (int Contador = 0; Contador < Pos; Contador++)
        Genotipo.Append(GenotipoA[Contador]);

    //Agrega el genotipo del segundo progenitor
    for (int Contador = Pos; Contador < GenotipoB.Length; Contador++)
        Genotipo.Append(GenotipoB[Contador]);
}

//Compara el genotipo con la cadena a la que debe parecerse
public int Evalua(StringBuilder Original) {
    int Puntaje = 0;
    for (int Contador = 0; Contador < Original.Length; Contador++)
        if (Original[Contador] == Genotipo[Contador])
            Puntaje++;
    return Puntaje;
}

//La población
internal class Poblacion {
    public List<Individuo> Individuos = new List<Individuo>();
    private Random Azar = new();

    public void Proceso(StringBuilder Original, int TotalIndividuos, int TotalCiclos) {
        //Genera la población
        Individuos.Clear();
        for (int Contador = 1; Contador <= TotalIndividuos; Contador++)
            Individuos.Add(new Individuo(Azar, Original.Length));

        //El proceso evolutivo
        for (int Contador = 1; Contador <= TotalCiclos; Contador++) {
            //Seleccionar al azar dos individuos de esa población: A y B
            int PosA = Azar.Next(Individuos.Count);
            int PosB;
            do {
                PosB = Azar.Next(Individuos.Count);
            } while (PosB == PosA);

            //Generan un hijo que nace del cruce
            Individuo Hijo = new Individuo(Azar, Individuos[PosA].Genotipo,
Individuos[PosB].Genotipo);

            int PuntajeA = Individuos[PosA].Evalua(Original); //Evaluar adaptación de A
            int PuntajeB = Individuos[PosB].Evalua(Original); //Evaluar adaptación de B
            int PuntajeHijo = Hijo.Evalua(Original); //Evalúa al hijo

            if (PuntajeHijo > PuntajeA) Individuos[PosA].Genotipo = new
StringBuilder(Hijo.Genotipo.ToString());
            if (PuntajeHijo > PuntajeB) Individuos[PosB].Genotipo = new
StringBuilder(Hijo.Genotipo.ToString());
        }

        //Buscar individuo con mejor adaptación de la población
        int MejorPuntaje = 0;
        int MejorIndivid = 0;
        for (int indiv = 0; indiv < Individuos.Count; indiv++) {
            int Puntaje = Individuos[indiv].Evalua(Original);
            if (Puntaje > MejorPuntaje) {
                MejorPuntaje = Puntaje;
                MejorIndivid = indiv;
            }
        }

        //Imprime el mejor individuo
        Console.WriteLine("Mejor individuo: " + Individuos[MejorIndivid].Genotipo);
        Console.WriteLine("Puntaje: " + MejorPuntaje);
    }
}

```

```
} }
```

Consola de depuración de Mi... + ▾

Mejor individuo: Estoy probando un algoritmo evolutivo con el operador cruce
Puntaje: 59

C:\Users\engin\source\repos\Ejemplo\Ejemplo\bin\Debug\net8.0\Ejemplo.exe (pro
Para cerrar automáticamente la consola cuando se detiene la depuración, habil
Cerrar la consola automáticamente al detenerse la depuración.
Presione cualquier tecla para cerrar esta ventana. . .|

Ilustración 292: Algoritmo evolutivo con el operador cruce

Combinando los operadores cruce y mutación

Otra estrategia es usar los dos operadores, se cruzan dos individuos y luego el hijo resultante se le aplica el operador mutación.

J/003.cs

```
using System.Text;

namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Combina operador cruce y mutación
            Poblacion pobl = new();
            StringBuider Cadena = new();
            Cadena.Append("Probando algoritmo evolutivo con operador cruce y mutación");

            int TotalPoblacion = 100;
            int TotalCiclos = 90000;
            pobl.Proceso(Cadena, TotalPoblacion, TotalCiclos);
        }
    }

    //Cómo es el individuo
    internal class Individuo {
        public StringBuider Genotipo;
        private readonly StringBuider Letras = new
StringBuider("abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ áéúíóúÁÉÍÓÚñ¿?¡!äëöüÄËÖÜ");

        //Al nacer, tendrá una serie de caracteres al azar
        public Individuo(Random Azar, int Longitud) {
            Genotipo = new StringBuider();
            for (int Contador = 1; Contador <= Longitud; Contador++)
                Genotipo.Append(Letras[Azar.Next(Letras.Length)]);
        }

        //Nace de un proceso de cruce, luego recibe dos genotipos
        public Individuo(Random Azar, StringBuider GenotipoA, StringBuider GenotipoB) {
            //Una posición al azar de dónde corta de un genotipo y otro
            int Pos = Azar.Next(GenotipoA.Length);

            Genotipo = new StringBuider();

            //Agrega el genotipo del primer progenitor
            for (int Contador = 0; Contador < Pos; Contador++)
                Genotipo.Append(GenotipoA[Contador]);

            //Agrega el genotipo del segundo progenitor
            for (int Contador = Pos; Contador < GenotipoB.Length; Contador++)
                Genotipo.Append(GenotipoB[Contador]);
        }

        //Cambia una letra del genotipo al azar
        public void Muta(Random Azar) {
            int Posicion = Azar.Next(Genotipo.Length);
            Genotipo[Posicion] = Letras[Azar.Next(Letras.Length)];
        }

        //Compara el genotipo con la cadena a la que debe parecerse
        public int Evalua(StringBuider Original) {
            int Puntaje = 0;
            for (int Contador = 0; Contador < Original.Length; Contador++)
                if (Original[Contador] == Genotipo[Contador])
                    Puntaje++;
            return Puntaje;
        }
    }

    //La población
    internal class Poblacion {
        public List<Individuo> Individuos = new List<Individuo>();
        private Random Azar = new();

        public void Proceso(StringBuider Original, int TotalIndividuos, int TotalCiclos) {
            //Genera la población
            Individuos.Clear();
            for (int Contador = 1; Contador <= TotalIndividuos; Contador++)
                Individuos.Add(new Individuo(Azar, Original.Length));

            //El proceso evolutivo
            for (int Contador = 1; Contador <= TotalCiclos; Contador++) {
```

```

//Seleccionar al azar dos individuos de esa población: A y B
int PosA = Azar.Next(Individuos.Count);
int PosB;
do {
    PosB = Azar.Next(Individuos.Count);
} while (PosB == PosA);

//Generan un hijo que nace del cruce
Individuo Hijo = new Individuo(Azar, Individuos[PosA].Genotipo,
Individuos[PosB].Genotipo);

//Además el hijo muta
Hijo.Muta(Azar);

int PuntajeA = Individuos[PosA].Evalua(Original); //Evaluar adaptación de A
int PuntajeB = Individuos[PosB].Evalua(Original); //Evaluar adaptación de B
int PuntajeHijo = Hijo.Evalua(Original); //Evalúa al hijo

if (PuntajeHijo > PuntajeA) Individuos[PosA].Genotipo = new
StringBuilder(Hijo.Genotipo.ToString());
if (PuntajeHijo > PuntajeB) Individuos[PosB].Genotipo = new
StringBuilder(Hijo.Genotipo.ToString());
}

//Buscar individuo con mejor adaptación de la población
int MejorPuntaje = 0;
int MejorIndivid = 0;
for (int indiv = 0; indiv < Individuos.Count; indiv++) {
    int Puntaje = Individuos[indiv].Evalua(Original);
    if (Puntaje > MejorPuntaje) {
        MejorPuntaje = Puntaje;
        MejorIndivid = indiv;
    }
}

//Imprime el mejor individuo
Console.WriteLine("Mejor individuo: " + Individuos[MejorIndivid].Genotipo);
Console.WriteLine("Puntaje: " + MejorPuntaje);
}
}
}

```

```

Mejor individuo: Probando algoritmo evolutivo con operador cruce y mutación
Puntaje: 58

C:\Users\engin\source\repos\Ejemplo\Ejemplo\bin\Debug\net8.0\Ejemplo.exe (pr
Para cerrar automáticamente la consola cuando se detiene la depuración, habi
Cerrar la consola automáticamente al detenerse la depuración.
Presione cualquier tecla para cerrar esta ventana. . .

```

Ilustración 293: Algoritmo evolutivo: Combinando operador cruce y mutación

Consideraciones sobre los operadores

1. Hay muchas partes en las cuáles esos algoritmos se pueden cambiar y observar si son mejores o más rápidos.
2. En el algoritmo de mutación pueden hacerse estos cambios:
 - a. Aumentar o disminuir el número de ciclos.
 - b. Validar que cuando logre dar con la cadena, el programa se detenga.
 - c. Seleccionar tres o más individuos y que compitan entre estos para ver cuál es el mejor para generar un hijo.
 - d. Se puede escoger sólo un individuo, duplicarlo y mutarlo, repetir hasta que la copia mutada sea mejor que el progenitor y reemplazar el progenitor.
 - e. Mutar dos o más partes del hijo.
 - f. Se puede optimizar, almacenando el valor de adaptación y así no es necesario calcularlo constantemente.
 - g. Siguiendo el punto anterior, se calcularía la adaptación de toda la población y en vez de seleccionar un individuo al azar, se escoge el mejor para generar hijo y se elimina el de peor puntaje.
3. En el algoritmo de cruce pueden hacerse estos cambios:
 - a. Aumentar o disminuir el número de ciclos.
 - b. Validar que cuando logre dar con la cadena, el programa se detenga.
 - c. Seleccionar tres y que se logre un cruce entre estos con dos puntos de corte.
 - d. Generar dos o más hijos.
4. En el algoritmo de cruce y mutación se pueden aplicar los cambios propuestos de los dos anteriores.
5. ¿Por qué mantener la población en un número constante y no permitir que crezca o disminuya?
6. Hay partes que se pueden optimizar del código haciendo uso de APIs más rápidas en C#.
7. También podría cambiarse el generador de números pseudoaleatorios.
8. Un paso sería hacer pruebas sobre cuál algoritmo logra el más rápido acercamiento.

Buscar el máximo en una función

En este segundo problema, dada una función $y=f(x)$, un valor inicial para $x=a$ y un valor final para $x=b$ donde $a < b$, se pide encontrar en cuál valor de "x", se obtiene el máximo valor de "y" en ese rango $[a,b]$. Hallarlo sería derivar la función $y=f(x)$, igualar a cero y así se obtiene el valor de x . Pero puede que $y=f(x)$ sea una ecuación bastante compleja, que complique el derivarla y resolverla a $y'=0$. En estos casos es posible aplicar los algoritmos evolutivos.

Por ejemplo, se tiene esta curva:

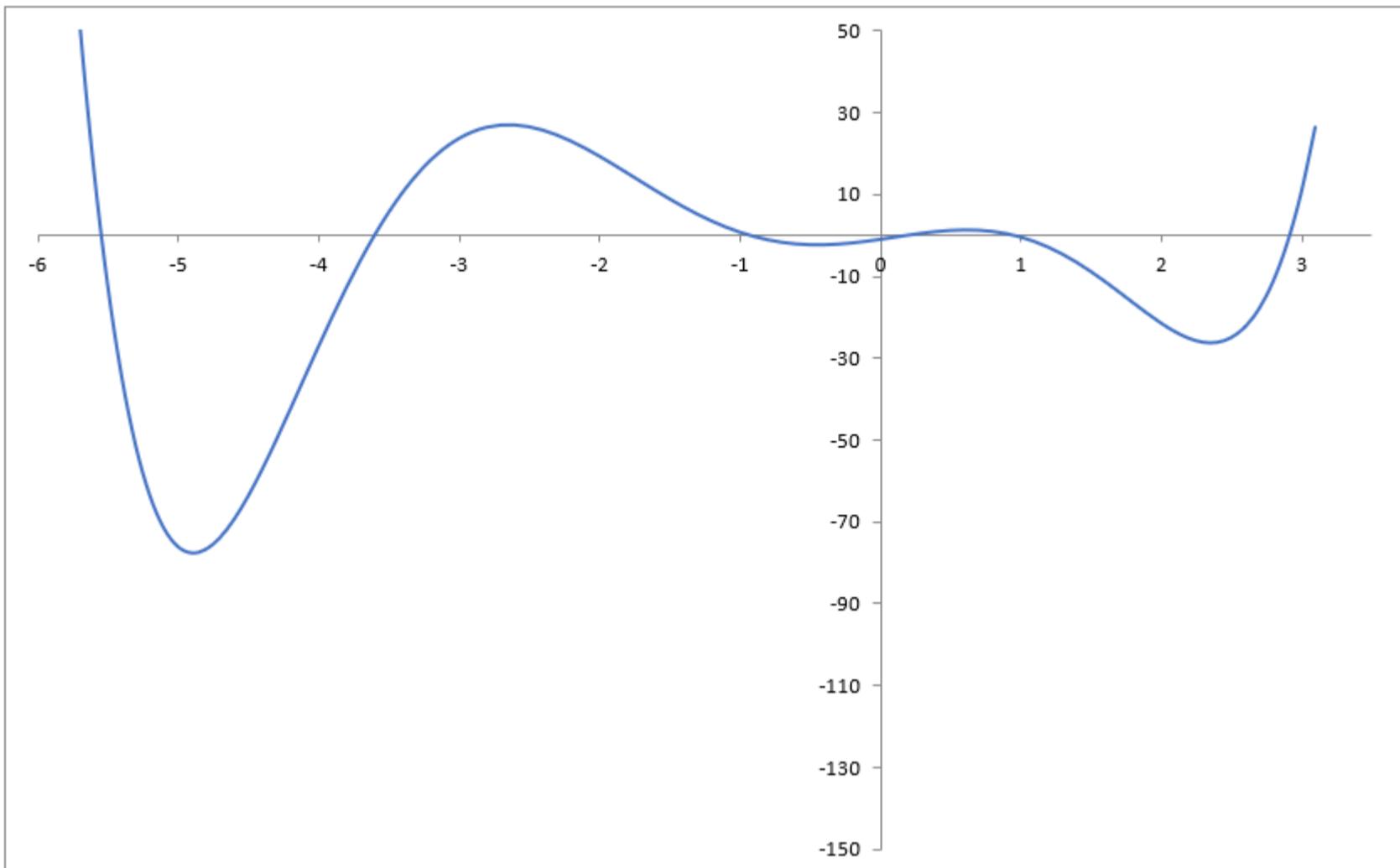


Ilustración 294: Polinomio de grado 6

Para hallar el valor de X que obtenga el mayor valor de Y en el rango Xmin y Xmax. La curva es:

$$Y = 0.1 * X^6 + 0.6 * X^5 - 0.9 * X^4 - 6.2 * X^3 + 2 * X^2 + 5 * X - 1$$

El primer paso con los algoritmos evolutivos es determinar cómo construir al individuo que represente una solución. Una forma de hacerlo es que los individuos sean números reales de doble precisión. Así que se crea una población de varios individuos como números reales aleatorios entre Xmin y Xmax y para este ejemplo, se usa la distribución uniforme.

En cada ciclo del proceso, se escogen dos individuos al azar y aquel que ofrezca el valor más alto de "Y", es el que se "reproduce" y varía con el operador mutación. ¿Cómo aplicar el operador mutación? Podría ser sumarle o restarle 0.1 o 0.01 o 0.001 o 0.0001 o 0.00001 o 0.000001 y así hasta donde alcance los decimales.

A continuación, el código en C# con el algoritmo:

1. Generar una población de individuos al azar (cada individuo es un número de tipo double entre Xmin y Xmax usando la distribución uniforme).
2. Se escogen dos individuos al azar de esa población.
3. Se califica cada individuo.
4. El mejor individuo, es decir, el que genere un mayor valor de Y es seleccionado y copiado.
5. La copia se modifica ya sea sumando o restándole 0.1 o 0.01 o 0.001 o 0.0001 o 0.00001 o 0.000001.
6. La copia sobrescribe al individuo perdedor.

El código es el siguiente:

J/004.cs

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Buscar el mayor valor de una ecuación modificando números de tipo double
            Poblacion pobl = new Poblacion();

            int TotalPoblacion = 100;
            int TotalCiclos = 90000;
            double ValorXminimo = -4;
            double ValorXmaximo = 1;
            pobl.Proceso(TotalPoblacion, TotalCiclos, ValorXminimo, ValorXmaximo);
        }
    }
}
```

```

}

//Cómo es el individuo
internal class Individuo {
    public double Genotipo;

    //Al nacer, tendrá un valor double entre 0 y 1
    public Individuo(Random Azar) {
        Genotipo = Azar.NextDouble();
    }

    //Cambia el valor en algún decimal validando que no se salga del rango 0 a 1
    public void Mutar(Random Azar) {
        double Copia;
        do {
            Copia = Genotipo;
            double Divide = Math.Pow(10, Azar.Next(2, 7));
            if (Azar.NextDouble() < 0.5)
                Copia -= 1 / Divide;
            else
                Copia += 1 / Divide;
        } while (Copia < 0 || Copia > 1);
        Genotipo = Copia;
    }
}

//La población
internal class Poblacion {
    public List<Individuo> Individuos = new List<Individuo>();
    private Random Azar = new Random();

    public void Proceso(int TotalIndividuos, int TotalCiclos, double Xmin, double Xmax) {
        //Genera la población
        Individuos.Clear();
        for (int Contador = 1; Contador <= TotalIndividuos; Contador++)
            Individuos.Add(new Individuo(Azar));

        //El proceso evolutivo
        for (int Contador = 1; Contador <= TotalCiclos; Contador++) {
            //Seleccionar al azar dos individuos de esa población: A y B
            int PosA = Azar.Next(Individuos.Count);
            int PosB;
            do {
                PosB = Azar.Next(Individuos.Count);
            } while (PosB == PosA);

            double ValorXA = Individuos[PosA].Genotipo * (Xmax - Xmin) + Xmin;
            double PuntajeA = Ecuacion(ValorXA); //Evaluar adaptación de A

            double ValorXB = Individuos[PosB].Genotipo * (Xmax - Xmin) + Xmin;
            double PuntajeB = Ecuacion(ValorXB); //Evaluar adaptación de B

            //Si adaptación de A es mejor que adaptación de B entonces
            if (PuntajeA > PuntajeB) {
                //Eliminar individuo B y duplicar individuo A
                Individuos[PosB].Genotipo = Individuos[PosA].Genotipo;
                //Modificar levemente al azar el nuevo duplicado
                Individuos[PosB].Mutar(Azar);
            }
            else {
                //Eliminar individuo A y duplicar individuo B
                Individuos[PosA].Genotipo = Individuos[PosB].Genotipo;
                //Modificar levemente al azar el nuevo duplicado
                Individuos[PosA].Mutar(Azar);
            }
        }

        //Buscar individuo con mejor adaptación de la población
        double MejorPuntaje = double.MinValue;
        int MejorIndividuo = 0;
        for (int indiv = 0; indiv < Individuos.Count; indiv++) {
            double ValorX = Individuos[indiv].Genotipo * (Xmax - Xmin) + Xmin;
            double Puntaje = Ecuacion(ValorX);
            if (Puntaje > MejorPuntaje) {
                MejorPuntaje = Puntaje;
                MejorIndividuo = indiv;
            }
        }
    }
}

```

```

//Imprime el mejor individuo
double MejorValorX = Individuos[MejorIndivid].Genotipo * (Xmax - Xmin) + Xmin;

Console.WriteLine("Búsqueda del mayor valor Y de una ecuación");
Console.WriteLine("Y = 0.1 * Math.Pow(x, 6) + 0.6 * Math.Pow(x, 5) - 0.9 * Math.Pow(x, 4) -
6.2 * Math.Pow(x, 3) + 2 * x * x + 5 * x - 1");
Console.WriteLine("Entre Xmin = " + Xmin + " y Xmax = " + Xmax);
Console.WriteLine("Valor X: " + MejorValorX);
Console.WriteLine("Valor Y: " + Ecuacion(MejorValorX));
}

public double Ecuacion(double x) {
    return 0.1 * Math.Pow(x, 6) + 0.6 * Math.Pow(x, 5) - 0.9 * Math.Pow(x, 4) - 6.2 * Math.Pow(x,
3) + 2 * x * x + 5 * x - 1;
}
}

```

Búsqueda del mayor valor Y de una ecuación
 $Y = 0.1 * \text{Math.Pow}(x, 6) + 0.6 * \text{Math.Pow}(x, 5) - 0.9 * \text{Math.Pow}(x, 4) - 6.2 * \text{Math.Pow}(x, 3) + 2 * x * x + 5 * x - 1$
Entre Xmin = -4 y Xmax = 1
Valor X: -2,643584396394198
Valor Y: 27,011875380368906

Ilustración 295: El mayor valor de Y en un rango dado

De genotipos y fenotipos

En el anterior problema, los individuos fueron números reales de doble precisión en C#, la mutación fue modificar los valores a nivel decimal. ¿Y si se quiere mayor control en esa modificación para poder aplicar el operador cruce?

En los seres vivos, las instrucciones para su desarrollo y funcionamiento están en el ADN. Esa información genética es el genotipo. La interpretación de ese genotipo es el fenotipo. Un ejemplo grosso modo:

Genotipo	Fenotipo
ACCTGAACCTGGCCAATGGCCTAACCTG	Forma del pico de un ave

En algoritmos evolutivos se hace una analogía similar: el genotipo es una cadena de números binarios, esta es interpretada, generando un fenotipo (por ejemplo, un valor real) el cuál se evalúa. Si el fenotipo de ese individuo le permite ser el "ganador", entonces es al genotipo de ese individuo "ganador" al que se le hacen las operaciones de mutación.

Del ejemplo anterior, está la siguiente función:

$$Y = -1.78 * \text{seno}(X)^2 + 6.40 * \text{coseno}(X)^2 + 3.07 * \text{seno}(X)^3$$

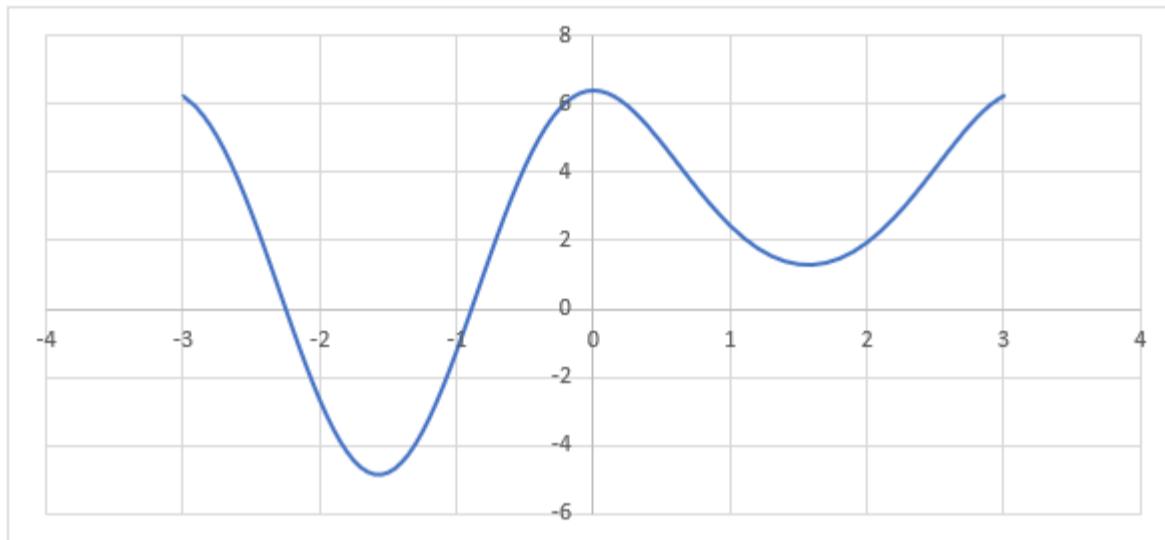


Ilustración 296: Gráfico de una ecuación

Se pide encontrar el valor de "x" entre -3 y 2, donde se obtenga el mínimo valor de "y". A ojo, viendo la gráfica es posible afirmar que el valor aproximado para x es -1.6, pero eso no es preciso. Se requiere precisión y este sería el proceso:

Paso 1

Se define que tan largo será el genotipo. Es una cadena de números binarios, entre más cifras, más preciso serán los resultados, pero tomará más tiempo en procesamiento. Una cadena puede ser: 011101101

Para el ejemplo, el valor mínimo -3 será representado por "000000000" y el valor máximo 2 será representado por "111111111". ¿Cuántas combinaciones hay? Es $2^{\text{total_cifras}}$, luego en este caso concreto sería $2^9 = 512$. ¿Qué significa eso? Que se va a dividir en 512 partes iguales el rango entre -3 y 2, y es en una de esas partes, donde estará el valor de "x" que se obtiene el mínimo valor de "y".

Nota aclaratoria: El rango mínimo sin importar su valor será representado como "000000000" y el rango máximo sin importar su valor será representado como "111111111". ¡No se debe hacer la conversión valor binario a valor decimal aquí!

Paso 2

Interpretando el genotipo. Si -3 será "000000000" y 2 será "111111111", entonces ¿Qué sería, por ejemplo, "011011101"? ¿A qué valor x correspondería? Así se calcula:

$$x = \text{RangoMinimo} + \text{valorbinarioadecimal} * \frac{\text{RangoMáximo} - \text{RangoMínimo}}{2^{\text{TotalCifras}} - 1}$$

Probando con "000000000" (0 en decimal)

$$x = -3 + 0 * \frac{2 - -3}{2^9 - 1} = -3 + 0 * \frac{5}{512 - 1} = -3 + 0 * \frac{5}{511} = -3$$

Probando con "111111111" (511 en decimal)

$$x = -3 + 511 * \frac{2 - -3}{2^9 - 1} = -3 + 511 * \frac{5}{511} = -3 + 5 = 2$$

Probando con "011011101" (221 en decimal)

$$x = -3 + 221 * \frac{2 - -3}{2^9 - 1} = -3 + 221 * \frac{5}{511} = -0,837573385518591$$

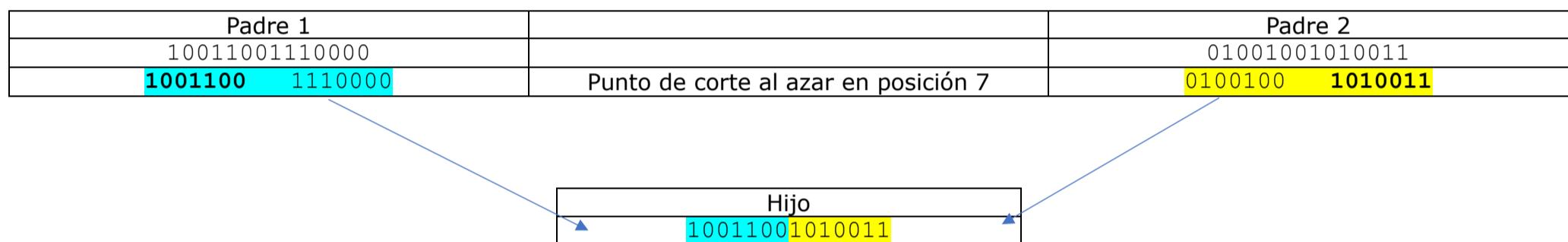
Paso 3

La representación binaria de un individuo da como ventaja la facilidad de implementar el operador mutación, porque es ir a una determinada posición de la cadena binaria y si esa tenía un "0" se cambia a "1" y viceversa. Por otro lado, la totalidad del individuo puede cambiarse porque cualquier posición de la cadena binaria está al alcance de ese operador. Por ejemplo:

Individuo	10011001110000
Posición al azar 9 que es	10011001 1 10000
Cambiando valor de esa posición	10011001 0 10000

Paso 4

Otra ventaja de la representación binaria es la facilidad de implementar el operador cruce, porque es simplemente dividir la cadena de los padres, combinar esos pedazos para dar origen al nuevo individuo. Por ejemplo:



Paso 5

¿Cómo el lenguaje de programación elegido para implementar el algoritmo evolutivo puede operar con valores binarios? ¿Tendrá operadores de fácil uso? ¿El desempeño será bueno? C# permite operaciones a nivel de bit, la cadena binaria estaría almacenada en una variable de tipo entero.

Paso 6

Definir una población. Los algoritmos evolutivos inician con una población inicial de individuos generados al azar. Siguiendo el enfoque anterior, la solución a esto es una lista de cadenas.

Genotipo: Operador mutación

El algoritmo sería el siguiente:

```
Algoritmo BuscaXparaMinimoValorY
Inicio
    Generar población de N individuos al azar (cadenas en binario)
    Inicio ciclo
        Seleccionar al azar dos individuos de esa población: A y B
        Evaluar valorY generado por el individuo A
        Evaluar valorY generado por el individuo B
        Si valorY de A es mayor que valorY de B entonces
            Eliminar individuo B
            Duplicar individuo A
            Modificar un bit al azar del nuevo duplicado
        de lo contrario
            Eliminar individuo A
            Duplicar individuo B
            Modificar un bit al azar del nuevo duplicado
        Fin Si
    Fin ciclo
    Buscar individuo que genere mayor Y de la población
    Imprimir individuo
Fin
```

J/005.cs

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Buscar el mayor valor de una ecuación modificando números binarios
            Poblacion pobl = new();

            int TotalIndividuos = 100;
            int TotalCiclos = 90000;
            int NumeroBits = 20;
            double ValorXminimo = -4;
            double ValorXmaximo = 1;
            pobl.Proceso(TotalIndividuos, NumeroBits, TotalCiclos, ValorXminimo, ValorXmaximo);
        }
    }

    //Cómo es el individuo
    internal class Individuo {
        public int Genotipo;

        //Al nacer, tendrá un valor double entre 0 y 1
        public Individuo(Random Azar, int NumeroBits) {
            Genotipo = Azar.Next((int) Math.Pow(2, NumeroBits));
        }

        //Cambia el valor en algun bit
        public void Mutar(Random Azar, int NumeroBits) {
            int Mascara = 1 << Azar.Next(NumeroBits);
            Genotipo ^= Mascara;
        }
    }

    //La población
    internal class Poblacion {
        public List<Individuo> Individuos = new List<Individuo>();
        private Random Azar = new Random();

        public void Proceso(int TotalIndividuos, int NumeroBits, int TotalCiclos, double Xmin, double Xmax) {
            //Genera la población
            Individuos.Clear();
            for (int Contador = 1; Contador <= TotalIndividuos; Contador++)
                Individuos.Add(new Individuo(Azar, NumeroBits));

            //El divisor
            double Divide = Math.Pow(2, NumeroBits) - 1;

            //El proceso evolutivo
            for (int Contador = 1; Contador <= TotalCiclos; Contador++) {
                //Seleccionar al azar dos individuos de esa población: A y B
                int PosA = Azar.Next(Individuos.Count);
```

```

int PosB;
do {
    PosB = Azar.Next(Individuos.Count);
} while (PosB == PosA);

double ValorXa = Xmin + Individuos[PosA].Genotipo * (Xmax - Xmin) / Divide;
double PuntajeA = Ecuacion(ValorXa); //Evaluar adaptación de A

double ValorXb = Xmin + Individuos[PosB].Genotipo * (Xmax - Xmin) / Divide;
double PuntajeB = Ecuacion(ValorXb); //Evaluar adaptación de B

//Si adaptación de A es mejor que adaptación de B entonces
if (PuntajeA > PuntajeB) {
    //Eliminar individuo B y duplicar individuo A
    Individuos[PosB].Genotipo = Individuos[PosA].Genotipo;
    //Modificar levemente al azar el nuevo duplicado
    Individuos[PosB].Muta(Azar, NumeroBits);
}
else {
    //Eliminar individuo A y duplicar individuo B
    Individuos[PosA].Genotipo = Individuos[PosB].Genotipo;
    //Modificar levemente al azar el nuevo duplicado
    Individuos[PosA].Muta(Azar, NumeroBits);
}

}

//Buscar individuo con mejor adaptación de la población
double MejorPuntaje = double.MinValue;
int MejorIndivid = 0;
for (int indiv = 0; indiv < Individuos.Count; indiv++) {
    double ValorX = Xmin + Individuos[indiv].Genotipo * (Xmax - Xmin) / Divide;
    double Puntaje = Ecuacion(ValorX);
    if (Puntaje > MejorPuntaje) {
        MejorPuntaje = Puntaje;
        MejorIndivid = indiv;
    }
}

//Imprime el mejor individuo
double MejorValorX = Xmin + Individuos[MejorIndivid].Genotipo * (Xmax - Xmin) / Divide;

Console.WriteLine("Búsqueda del mayor valor Y de una ecuación");
Console.WriteLine("Y = 0.1 * Math.Pow(x, 6) + 0.6 * Math.Pow(x, 5) - 0.9 * Math.Pow(x, 4) -
6.2 * Math.Pow(x, 3) + 2 * x * x + 5 * x - 1");
Console.WriteLine("Entre Xmin = " + Xmin + " y Xmax = " + Xmax);
Console.WriteLine("Número de bits: " + NumeroBits);
Console.WriteLine("Valor X: " + MejorValorX);
Console.WriteLine("Valor Y: " + Ecuacion(MejorValorX));
}

public double Ecuacion(double x) {
    return 0.1 * Math.Pow(x, 6) + 0.6 * Math.Pow(x, 5) - 0.9 * Math.Pow(x, 4) - 6.2 * Math.Pow(x,
3) + 2 * x * x + 5 * x - 1;
}
}
}

```

Consola de depuración de Mi... X + ▾

Búsqueda del mayor valor Y de una ecuación
 $Y = 0.1 * Math.Pow(x, 6) + 0.6 * Math.Pow(x, 5) - 0.9 * Math.Pow(x, 4) - 6.2 * Math.Pow(x, 3) + 2 * x * x + 5 * x - 1$
Entre Xmin = -4 y Xmax = 1
Número de bits: 20
Valor X: -2,643582957823713
Valor Y: 27,01187538036423

Ilustración 297: Búsqueda del mayor valor usando genotipo de cadenas de bits

Genotipo: Operador cruce

El algoritmo sería el siguiente:

```
Algoritmo BuscaXparaMinimoValorY
Inicio
    Generar población de N individuos al azar (cadenas en binario)
    Inicio ciclo
        Seleccionar al azar dos individuos de esa población: A y B
        Generar Hijo cruzando los genes de A y B
        Evaluar valorY generado por el individuo A
        Evaluar valorY generado por el individuo B
        Evaluar valorY generado por el Hijo
        Si valorY de Hijo es mayor que valorY de A entonces Hijo reemplaza a A
        Si valorY de Hijo es mayor que valorY de B entonces Hijo reemplaza a B
    Fin ciclo
    Buscar individuo que genere mayor Y de la población
    Imprimir individuo
Fin
```

J/006.cs

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Buscar el mayor valor de una ecuación modificando números binarios
            Poblacion pobl = new();

            int TotalIndividuos = 1000;
            int TotalCiclos = 90000;
            int NumeroBits = 20;
            double ValorXminimo = -4;
            double ValorXmaximo = 1;
            pobl.Proceso(TotalIndividuos, NumeroBits, TotalCiclos, ValorXminimo, ValorXmaximo);
        }
    }

    //Cómo es el individuo
    internal class Individuo {
        public int Genotipo;

        //Al nacer, tendrá un valor dependiendo del número de bits
        public Individuo(Random Azar, int NumeroBits) {
            Genotipo = Azar.Next((int) Math.Pow(2, NumeroBits));
        }

        //Operador cruce.
        public Individuo(Random Azar, int GeneticoA, int GeneticoB) {
            //En que posición corta el genotipo de cada parente
            int Posicion = Azar.Next(sizeof(int) * 8);

            //Extrae las partes de cada progenitor
            int Mascara = (1 << Posicion) - 1;
            int ParteA = GeneticoA >> Posicion;
            int ParteB = GeneticoB & Mascara;

            //Une las partes las inicial de A y la final de B
            Genotipo = (ParteA << Posicion) | ParteB;
        }
    }

    //La población
    internal class Poblacion {
        public List<Individuo> Individuos = new List<Individuo>();
        private Random Azar = new Random();

        public void Proceso(int TotalIndividuos, int NumeroBits, int TotalCiclos, double Xmin, double Xmax) {
            //Genera la población
            Individuos.Clear();
            for (int Contador = 1; Contador <= TotalIndividuos; Contador++)
                Individuos.Add(new Individuo(Azar, NumeroBits));

            //El divisor
            double Divide = Math.Pow(2, NumeroBits) - 1;

            //El proceso evolutivo
            for (int Contador = 1; Contador <= TotalCiclos; Contador++) {
                //Seleccionar al azar dos individuos de esa población: A y B
```

```

int PosA = Azar.Next(Individuos.Count);
int PosB;
do {
    PosB = Azar.Next(Individuos.Count);
} while (PosB == PosA);

//Generan un hijo que nace del cruce
Individuo Hijo = new Individuo(Azar, Individuos[PosA].Genotipo,
Individuos[PosB].Genotipo);

double ValorXa = Xmin + Individuos[PosA].Genotipo * (Xmax - Xmin) / Divide;
double PuntajeA = Ecuacion(ValorXa); //Evaluar adaptación de A

double ValorXb = Xmin + Individuos[PosB].Genotipo * (Xmax - Xmin) / Divide;
double PuntajeB = Ecuacion(ValorXb); //Evaluar adaptación de B

double ValorXHijo = Xmin + Hijo.Genotipo * (Xmax - Xmin) / Divide;
double PuntajeHijo = Ecuacion(ValorXHijo); //Evaluar adaptación de Hijo

if (PuntajeHijo > PuntajeA) Individuos[PosA].Genotipo = Hijo.Genotipo;
if (PuntajeHijo > PuntajeB) Individuos[PosB].Genotipo = Hijo.Genotipo;
}

//Buscar individuo con mejor adaptación de la población
double MejorPuntaje = double.MinValue;
int MejorIndivid = 0;
for (int indiv = 0; indiv < Individuos.Count; indiv++) {
    double ValorX = Xmin + Individuos[indiv].Genotipo * (Xmax - Xmin) / Divide;
    double Puntaje = Ecuacion(ValorX);
    if (Puntaje > MejorPuntaje) {
        MejorPuntaje = Puntaje;
        MejorIndivid = indiv;
    }
}

//Imprime el mejor individuo
double MejorValorX = Xmin + Individuos[MejorIndivid].Genotipo * (Xmax - Xmin) / Divide;

Console.WriteLine("Búsqueda del mayor valor Y de una ecuación. Operador cruce.");
Console.WriteLine("Y = 0.1 * Math.Pow(x, 6) + 0.6 * Math.Pow(x, 5) - 0.9 * Math.Pow(x, 4) - 6.2 * Math.Pow(x, 3) + 2 * x * x + 5 * x - 1");
Console.WriteLine("Entre Xmin = " + Xmin + " y Xmax = " + Xmax);
Console.WriteLine("Número de bits: " + NumeroBits);
Console.WriteLine("Valor X: " + MejorValorX);
Console.WriteLine("Valor Y: " + Ecuacion(MejorValorX));
}

public double Ecuacion(double x) {
    return 0.1 * Math.Pow(x, 6) + 0.6 * Math.Pow(x, 5) - 0.9 * Math.Pow(x, 4) - 6.2 * Math.Pow(x, 3) + 2 * x * x + 5 * x - 1;
}
}
}

```

```

Búsqueda del mayor valor Y de una ecuación. Operador cruce.
Y = 0.1 * Math.Pow(x, 6) + 0.6 * Math.Pow(x, 5) - 0.9 * Math.Pow(x, 4) - 6.2 * Math.Pow(x, 3) + 2 * x * x + 5 * x - 1
Entre Xmin = -4 y Xmax = 1
Número de bits: 20
Valor X: -2,643582957823713
Valor Y: 27,01187538036423

```

Ilustración 298: Genotipo: Operador cruce

Genotipo: Operador cruce y mutación

El algoritmo sería el siguiente:

```
Algoritmo BuscaXparaMinimoValorY
Inicio
    Generar población de N individuos al azar (cadenas en binario)
    Inicio ciclo
        Seleccionar al azar dos individuos de esa población: A y B
        Generar Hijo cruzando los genes de A y B
        Mutar Hijo
        Evaluar valorY generado por el individuo A
        Evaluar valorY generado por el individuo B
        Evaluar valorY generado por el Hijo
        Si valorY de Hijo es mayor que valorY de A entonces Hijo reemplaza a A
        Si valorY de Hijo es mayor que valorY de B entonces Hijo reemplaza a B
    Fin ciclo
    Buscar individuo que genere mayor Y de la población
    Imprimir individuo
Fin
```

J/007.cs

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Buscar el mayor valor de una ecuación. Operador cruce y mutación
            Poblacion pobl = new();

            int TotalIndividuos = 1000;
            int TotalCiclos = 90000;
            int NumeroBits = 20;
            double ValorXminimo = -4;
            double ValorXmaximo = 1;
            pobl.Proceso(TotalIndividuos, NumeroBits, TotalCiclos, ValorXminimo, ValorXmaximo);
        }
    }

    //Cómo es el individuo
    internal class Individuo {
        public int Genotipo;

        //Al nacer, tendrá un valor dependiendo del número de bits
        public Individuo(Random Azar, int NumeroBits) {
            Genotipo = Azar.Next((int) Math.Pow(2, NumeroBits));
        }

        //Operador cruce.
        public Individuo(Random Azar, int GeneticoA, int GeneticoB) {
            //En que posición corta el genotipo de cada parente
            int Posicion = Azar.Next(sizeof(int) * 8);

            //Extrae las partes de cada progenitor
            int Mascara = (1 << Posicion) - 1;
            int ParteA = GeneticoA >> Posicion;
            int ParteB = GeneticoB & Mascara;

            //Une las partes las inicial de A y la final de B
            Genotipo = (ParteA << Posicion) | ParteB;
        }

        //Mutación: Cambia el valor en algun bit
        public void Mutar(Random Azar, int NumeroBits) {
            int Mascara = 1 << Azar.Next(NumeroBits);
            Genotipo ^= Mascara;
        }
    }

    //La población
    internal class Poblacion {
        public List<Individuo> Individuos = new List<Individuo>();
        private Random Azar = new Random();

        public void Proceso(int TotalIndividuos, int NumeroBits, int TotalCiclos, double Xmin, double Xmax) {
            //Genera la población
            Individuos.Clear();
        }
    }
}
```

```

for (int Contador = 1; Contador <= TotalIndividuos; Contador++)
    Individuos.Add(new Individuo(Azar, NumeroBits));

//El divisor
double Divide = Math.Pow(2, NumeroBits) - 1;

//El proceso evolutivo
for (int Contador = 1; Contador <= TotalCiclos; Contador++) {
    //Seleccionar al azar dos individuos de esa población: A y B
    int PosA = Azar.Next(Individuos.Count);
    int PosB;
    do {
        PosB = Azar.Next(Individuos.Count);
    } while (PosB == PosA);

    //Generan un hijo que nace del cruce
    Individuo Hijo = new Individuo(Azar, Individuos[PosA].Genotipo,
Individuos[PosB].Genotipo);

    //Además muta al Hijo
    Hijo.Muta(Azar, NumeroBits);

    double ValorXA = Xmin + Individuos[PosA].Genotipo * (Xmax - Xmin) / Divide;
    double PuntajeA = Ecuacion(ValorXA); //Evaluar adaptación de A

    double ValorXB = Xmin + Individuos[PosB].Genotipo * (Xmax - Xmin) / Divide;
    double PuntajeB = Ecuacion(ValorXB); //Evaluar adaptación de B

    double ValorXHijo = Xmin + Hijo.Genotipo * (Xmax - Xmin) / Divide;
    double PuntajeHijo = Ecuacion(ValorXHijo); //Evaluar adaptación de Hijo

    if (PuntajeHijo > PuntajeA) Individuos[PosA].Genotipo = Hijo.Genotipo;
    if (PuntajeHijo > PuntajeB) Individuos[PosB].Genotipo = Hijo.Genotipo;
}

//Buscar individuo con mejor adaptación de la población
double MejorPuntaje = double.MinValue;
int MejorIndivid = 0;
for (int indiv = 0; indiv < Individuos.Count; indiv++) {
    double ValorX = Xmin + Individuos[indiv].Genotipo * (Xmax - Xmin) / Divide;
    double Puntaje = Ecuacion(ValorX);
    if (Puntaje > MejorPuntaje) {
        MejorPuntaje = Puntaje;
        MejorIndivid = indiv;
    }
}

//Imprime el mejor individuo
double MejorValorX = Xmin + Individuos[MejorIndivid].Genotipo * (Xmax - Xmin) / Divide;

Console.WriteLine("Búsqueda del mayor valor Y de una ecuación. Operador cruce y mutación.");
Console.WriteLine("Y = 0.1 * Math.Pow(x, 6) + 0.6 * Math.Pow(x, 5) - 0.9 * Math.Pow(x, 4) - 6.2 * Math.Pow(x, 3) + 2 * x * x + 5 * x - 1");
Console.WriteLine("Entre Xmin = " + Xmin + " y Xmax = " + Xmax);
Console.WriteLine("Número de bits: " + NumeroBits);
Console.WriteLine("Valor X: " + MejorValorX);
Console.WriteLine("Valor Y: " + Ecuacion(MejorValorX));
}

public double Ecuacion(double x) {
    return 0.1 * Math.Pow(x, 6) + 0.6 * Math.Pow(x, 5) - 0.9 * Math.Pow(x, 4) - 6.2 * Math.Pow(x, 3) + 2 * x * x + 5 * x - 1;
}
}
}

```

Consola de depuración de Mi... X + ▾

Búsqueda del mayor valor Y de una ecuación. Operador cruce y mutación.
Y = 0.1 * Math.Pow(x, 6) + 0.6 * Math.Pow(x, 5) - 0.9 * Math.Pow(x, 4) - 6.2 * Math.Pow(x, 3) + 2 * x * x + 5 * x - 1
Entre Xmin = -4 y Xmax = 1
Número de bits: 20
Valor X: -2,643582957823713
Valor Y: 27,01187538036423

Ilustración 299: Operador cruce y mutación

Máximos y mínimos locales

En el siguiente gráfico se puede apreciar visualmente un problema con los algoritmos evolutivos. Se busca obtener el valor de X con el que se obtiene el mayor valor de Y. Los círculos rellenos representan diferentes individuos generados al azar dentro de la población.

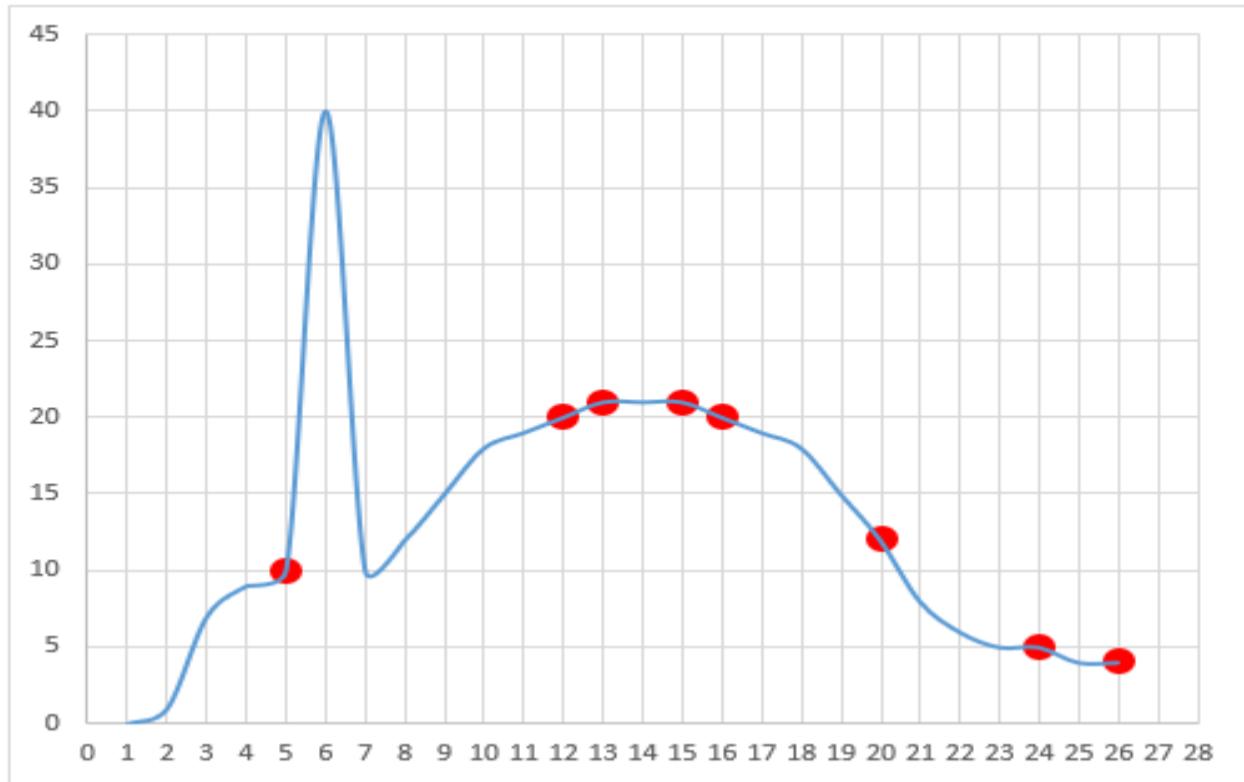


Ilustración 300: Gráfico matemático

Visualmente, el máximo "Y" se encuentra entre los valores de $5 \leq X \leq 7$. Hay muchos individuos entre $11 \leq X \leq 17$ que al competir contra el único individuo en la posición $X=5$, ganarían y se reproducirían eliminando de la población al individuo que en verdad estaba cerca de la solución global. El algoritmo evolutivo dará más puntaje a los individuos entre $11 \leq X \leq 17$, y se deduzca erróneamente que el máximo está entre $11 \leq X \leq 17$. Eso se le conoce como máximo local y una vez que el algoritmo evolutivo privilegie a los individuos de ese máximo local, se borrará cualquier esperanza de que se llegue al máximo global real.

¿Cómo solucionarlo? La variedad de los individuos dentro de la población debe mantenerse, pero no es fácil porque el mismo algoritmo premia a los ganadores con reproducirse y a los perdedores los castiga con la eliminación, así al perdedor le faltase muy poco para dar con la respuesta correcta.

El mismo problema se tendría con un mínimo local y un mínimo global.

Técnicas para evitar caer en máximos y mínimos locales:

1. Ejecutar varias veces el algoritmo evolutivo.
2. Tener varias poblaciones separadas.
3. Cada cierto número de ciclos generar aleatoriamente algunos individuos que reemplacen a algunos de la población existente.
4. Hacer uso del operador cruce que asemeja a la reproducción sexual, la cual es una técnica utilizada en la naturaleza para mantener la variabilidad.

Variando el algoritmo evolutivo

Los algoritmos evolutivos vistos en los ejemplos han sido así: se genera una población de N individuos, se seleccionan dos individuos al azar, se evalúan y comparan sus puntajes, el individuo perdedor es eliminado de la población dejando una vacante, el individuo ganador, en cambio, se premia clonándolo y ese clon se modifica en alguna parte al azar (operador mutación) para posteriormente ocupar la vacante dejada por el individuo eliminado.

Cuando se hace uso del operador cruce, entonces se toman dos individuos al azar, de ellos se genera un hijo al cruzar sus genotipos. Si el hijo resultante, es mejor que alguno de los padres, entonces el o los padres son reemplazados por el hijo en la población.

Tamaño de la población

En los dos casos mencionados anteriormente, la población mantiene una cantidad constante de individuos, pero esto no necesariamente debe ser así. Es posible que un algoritmo evolutivo varíe el tamaño de la población si así lo requiere. Cabe recordar que una población muy grande hará lenta la búsqueda de una solución, pero en una población pequeña se corre el riesgo que se caiga en un mínimo o máximo local, porque se pierde la variedad cuando un individuo es exitoso y su descendencia cubra la población entera.

Selección de individuos

La selección de individuos que compiten entre sí (operador mutación) o para reproducirse (operador cruce) se ha mantenido igual en los diferentes ejemplos. No es necesario limitarse a escoger dos individuos al azar, se puede escoger un número mayor y variar la escogencia del ganador y del perdedor, por ejemplo:

Paso 1: Seleccionar al azar tres individuos de la población

Paso 2: Poner a competir esos tres individuos y seleccionar el mejor

Paso 3: El mejor, se clona y se modifica una parte del clon al azar

Paso 4: Seleccionar al azar tres individuos de la población

Paso 5: Poner a competir esos tres individuos y seleccionar el peor

Paso 6: El peor individuo es reemplazado por el clon mutado generado en el paso 3

Otra forma de seleccionar individuos y que opere el algoritmo evolutivo es así:

Paso 1: Evalúe cada individuo de la población frente al problema y determine su puntaje.

Paso 2: Ordene la población del mejor individuo al peor individuo.

Paso 3: Seleccione un individuo al azar de las primeras posiciones, clóñelo, modifíquelo en alguna parte al azar y evalúelo.

Paso 4: Tome un individuo al azar de las últimas posiciones y compare su puntaje con el clon mutado generado en el paso 3. Si el clon mutado es mejor, entonces elimine el individuo y esa vacante es ocupada por el clon mutado. Vuelva al paso 2.

Representación de los individuos

En los ejemplos, el genotipo es de tamaño fijo. La razón es que, dependiendo del problema, el cubrir el rango de soluciones se use un genoma de tamaño fijo. Pero no necesariamente es así, se puede implementar que varíe dinámicamente el tamaño del genoma.

Paralelizar el algoritmo

Los algoritmos evolutivos son candidatos excelentes para ser paralelizados aprovechando que en la actualidad los equipos de cómputo vienen con varios procesadores.

Diversos hilos de ejecución podrían seleccionar individuos de la población y ponerlos a competir entre sí, el único cuidado a tener es que los hilos no seleccionen los mismos individuos. También se podría tener varias poblaciones separadas entre sí y cada hilo aplicaría el algoritmo evolutivo en cada una.

Buscar el mayor valor en una ecuación de múltiples variables

Si hay una ecuación del tipo:

$$Y = F(a, b, c, d, e)$$

Donde a, b, c, d, e son variables independientes. ¿Cómo encontrar el mayor valor de Y si nos dan un rango en el que oscilan esas variables independientes? Este es un problema sencillo de solucionar, pero difícil de llevarlo a la práctica, porque se tendrían los siguientes ciclos anidados:

```
for (a = Minimo; a <= Maximo; a += 0.001)
    for (b = Minimo; b <= Maximo; b += 0.001)
        for (c = Minimo; c <= Maximo; c += 0.001)
            for (d = Minimo; d <= Maximo; d += 0.001)
                for (e = Minimo; e <= Maximo; e += 0.001)
```

Al terminar de ejecutar se obtendría el mayor valor de Y, pero haciendo cuentas, suponiendo que cada ciclo itera unas 1000 veces, se tiene que el total de iteraciones es $1000^5 = 1.000.000.000.000$ de veces, una cantidad enorme que puede consumir mucho tiempo. ¿Y si requiere mayor precisión? ¿Y si son más variables? En esos casos hay un gran problema.

Los algoritmos evolutivos ofrecen una solución muy buena (no perfecta) para encontrar el mayor valor, el algoritmo es el siguiente:

```
Algoritmo Evolutivo
Inicio
    Generar población de N individuos (cada uno con valores a, b, c, d, e al azar entre Mínimo y Máximo)
    Inicio ciclo
        Seleccionar al azar dos individuos de esa población: A y B
        Evaluar adaptación de A
        Evaluar adaptación de B
        Si adaptación de A es mejor que adaptación de B entonces
            Eliminar individuo B
            Duplicar individuo A
            Cambiar al azar alguna variable (a, b, c, d, e) del nuevo duplicado
        de lo contrario
            Eliminar individuo A
            Duplicar individuo B
            Cambiar al azar alguna variable (a, b, c, d, e) del nuevo duplicado
        Fin Si
    Fin ciclo
    Buscar individuo con mejor adaptación de la población
    Imprimir individuo
Fin
```

J/008.cs

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Buscar el mayor valor de una ecuación modificando números de tipo double
            Poblacion pobl = new Poblacion();

            int TotalPoblacion = 100;
            int TotalCiclos = 90000;
            double ValorMinimo = -10;
            double ValorMaximo = 10;
            pobl.Proceso(TotalPoblacion, TotalCiclos, ValorMinimo, ValorMaximo);
        }
    }

    //Cómo es el individuo
    internal class Individuo {
        public double valA, valB, valC, valD, valE;

        //Al nacer, tendrá un valor double entre 0 y 1
        public Individuo(Random Azar, double ValorMinimo, double ValorMaximo) {
            valA = Azar.NextDouble() * (ValorMaximo - ValorMinimo) + ValorMinimo;
            valB = Azar.NextDouble() * (ValorMaximo - ValorMinimo) + ValorMinimo;
            valC = Azar.NextDouble() * (ValorMaximo - ValorMinimo) + ValorMinimo;
            valD = Azar.NextDouble() * (ValorMaximo - ValorMinimo) + ValorMinimo;
            valE = Azar.NextDouble() * (ValorMaximo - ValorMinimo) + ValorMinimo;
        }
    }
}
```

```

//Cambia el valor de una variable
public void Muta(Random Azar, double ValorMinimo, double ValorMaximo) {
    switch (Azar.Next(5)) {
        case 0: valA = Azar.NextDouble() * (ValorMaximo - ValorMinimo) + ValorMinimo; break;
        case 1: valB = Azar.NextDouble() * (ValorMaximo - ValorMinimo) + ValorMinimo; break;
        case 2: valC = Azar.NextDouble() * (ValorMaximo - ValorMinimo) + ValorMinimo; break;
        case 3: valD = Azar.NextDouble() * (ValorMaximo - ValorMinimo) + ValorMinimo; break;
        case 4: valE = Azar.NextDouble() * (ValorMaximo - ValorMinimo) + ValorMinimo; break;
    }
}

//La población
internal class Poblacion {
    public List<Individuo> Individuos = new List<Individuo>();
    private Random Azar = new Random();

    public void Proceso(int TotalIndividuos, int TotalCiclos, double Minimo, double Maximo) {
        //Genera la población
        Individuos.Clear();
        for (int Contador = 1; Contador <= TotalIndividuos; Contador++)
            Individuos.Add(new Individuo(Azar, Minimo, Maximo));

        //El proceso evolutivo
        for (int Contador = 1; Contador <= TotalCiclos; Contador++) {
            //Seleccionar al azar dos individuos de esa población: A y B
            int PosA = Azar.Next(Individuos.Count);
            int PosB;
            do {
                PosB = Azar.Next(Individuos.Count);
            } while (PosB == PosA);

            double PuntajeA = Ecuacion(Individuos[PosA].valA, Individuos[PosA].valB,
Individuos[PosA].valC, Individuos[PosA].valD, Individuos[PosA].valE); //Evaluar adaptación de A
            double PuntajeB = Ecuacion(Individuos[PosB].valA, Individuos[PosB].valB,
Individuos[PosB].valC, Individuos[PosB].valD, Individuos[PosB].valE); //Evaluar adaptación de B

            //Si adaptación de A es mejor que adaptación de B entonces
            if (PuntajeA > PuntajeB) {
                //Eliminar individuo B y duplicar individuo A
                Individuos[PosB].valA = Individuos[PosA].valA;
                Individuos[PosB].valB = Individuos[PosA].valB;
                Individuos[PosB].valC = Individuos[PosA].valC;
                Individuos[PosB].valD = Individuos[PosA].valD;
                Individuos[PosB].valE = Individuos[PosA].valE;

                //Modificar levemente al azar el nuevo duplicado
                Individuos[PosB].Muta(Azar, Minimo, Maximo);
            }
            else {
                //Eliminar individuo A y duplicar individuo B
                Individuos[PosA].valA = Individuos[PosB].valA;
                Individuos[PosA].valB = Individuos[PosB].valB;
                Individuos[PosA].valC = Individuos[PosB].valC;
                Individuos[PosA].valD = Individuos[PosB].valD;
                Individuos[PosA].valE = Individuos[PosB].valE;

                //Modificar levemente al azar el nuevo duplicado
                Individuos[PosA].Muta(Azar, Minimo, Maximo);
            }
        }

        //Buscar individuo con mejor adaptación de la población
        double MejorPuntaje = double.MinValue;
        int MejorIndivid = 0;
        for (int indiv = 0; indiv < Individuos.Count; indiv++) {
            double Puntaje = Ecuacion(Individuos[indiv].valA, Individuos[indiv].valB,
Individuos[indiv].valC, Individuos[indiv].valD, Individuos[indiv].valE);
            if (Puntaje > MejorPuntaje) {
                MejorPuntaje = Puntaje;
                MejorIndivid = indiv;
            }
        }

        //Imprime el mejor individuo
        Console.WriteLine("Búsqueda del mayor valor Y de una ecuación de múltiples variables");
        Console.WriteLine("Entre Mínimo = " + Minimo + " y Máximo = " + Maximo);
        Console.WriteLine("Variable A: " + Individuos[MejorIndivid].valA);
        Console.WriteLine("Variable B: " + Individuos[MejorIndivid].valB);
        Console.WriteLine("Variable C: " + Individuos[MejorIndivid].valC);
    }
}

```

```

        Console.WriteLine("Variable D: " + Individuos[MejorIndivid].valD);
        Console.WriteLine("Variable E: " + Individuos[MejorIndivid].valE);
        Console.WriteLine("Valor Y: " + MejorPuntaje);
    }

    public double Ecuacion(double a, double b, double c, double d, double e) {
        return 0.3 * Math.Sin(a * c - d) +
            1.7 * Math.Sin(e * b + c) +
            2.8 * Math.Cos(3.1 * b - 4.4 * a) -
            3.1 * Math.Sin(a * d - e * c) +
            0.7 * Math.Cos(a + b * c - d);
    }
}

```

Consola de depuración de Mi... X + ▾

Búsqueda del mayor valor Y de una ecuación de múltiples variables
Entre Mínimo = -10 y Máximo = 10
Variable A: 2,8148394832897683
Variable B: -0,058897996064839475
Variable C: 8,004225647496664
Variable D: 2,3006977935944306
Variable E: 2,5729139849288067
Valor Y: 8,593369281905042

Ilustración 301: Buscar el mayor valor de Y con múltiples variables independientes

Parte 11: Redes Neuronales

Iniciando

Las redes neuronales son como una caja negra en la cual hay unas entradas, la caja en sí y unas salidas.

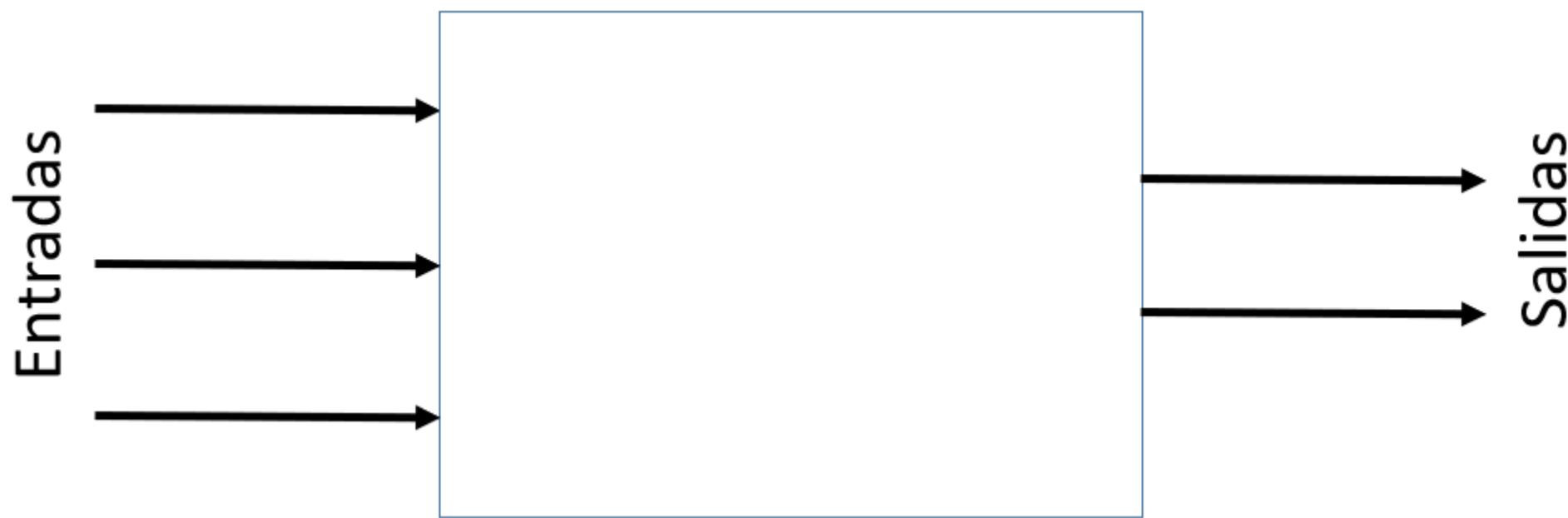


Ilustración 302: Caja negra, entradas y salidas

Lo particular es que hay unos pesos que dependiendo de su valor (más arriba o más abajo) afectan el valor de las salidas.

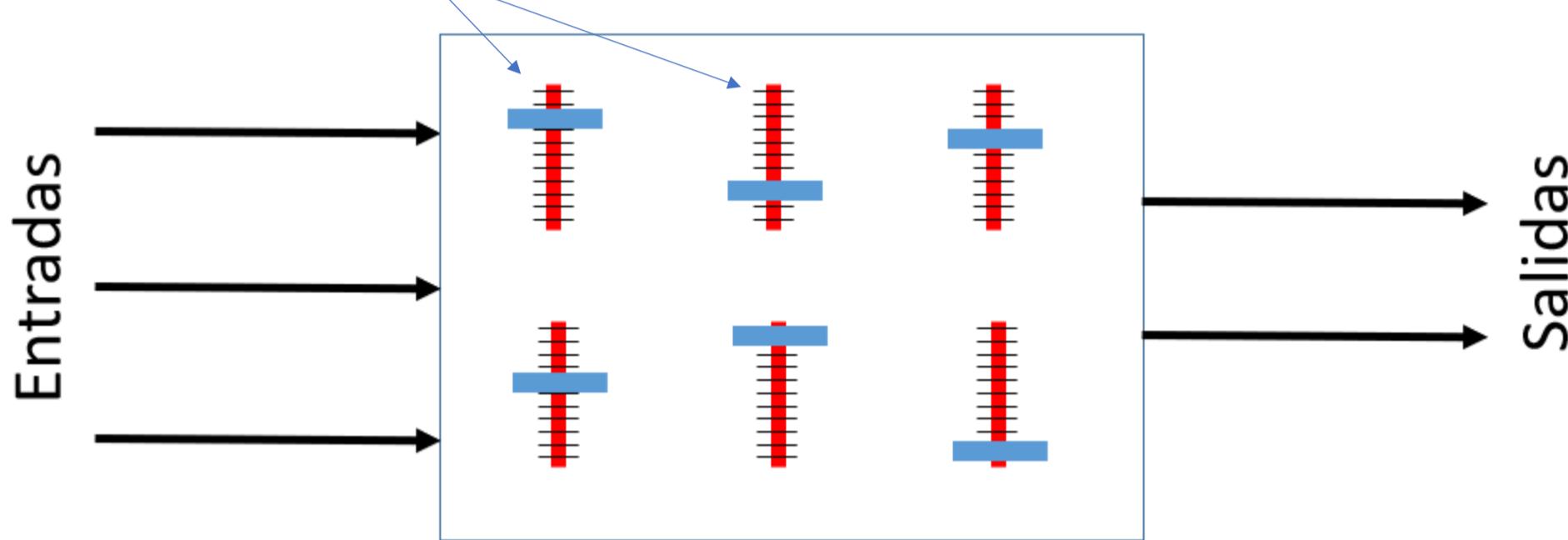


Ilustración 303: Pesos al interior de la caja

En el ejemplo, están las siguientes entradas y salidas:

	Entrada 1	Entrada 2	Entrada 3	Entrada 4	Salida deseada 1	Salida deseada 2
Ejemplo A	1	4	7	10	25	58
Ejemplo B	2	5	8	11	36	64
Ejemplo C	3	6	9	12	47	70

Significa que, si entran los números 1, 4, 7, 10, (ejemplo A), deberían salir 25 y 58. Luego hay que ajustar pesos (moviéndolos arriba o abajo) hasta obtener esa salida deseada.

Luego se prueban esos pesos con el nuevo conjunto de datos (ejemplo B). Se ingresa 2, 5, 8, 11 y debería salir 36 y 64. ¿Qué pasaría si eso no sucede? Que hay que cambiar los pesos con nuevos valores y probar desde el inicio (con el ejemplo A). ¿Cuándo termina? Cuando los valores de los pesos encontrados funcionen para los tres ejemplos (A, B y C).

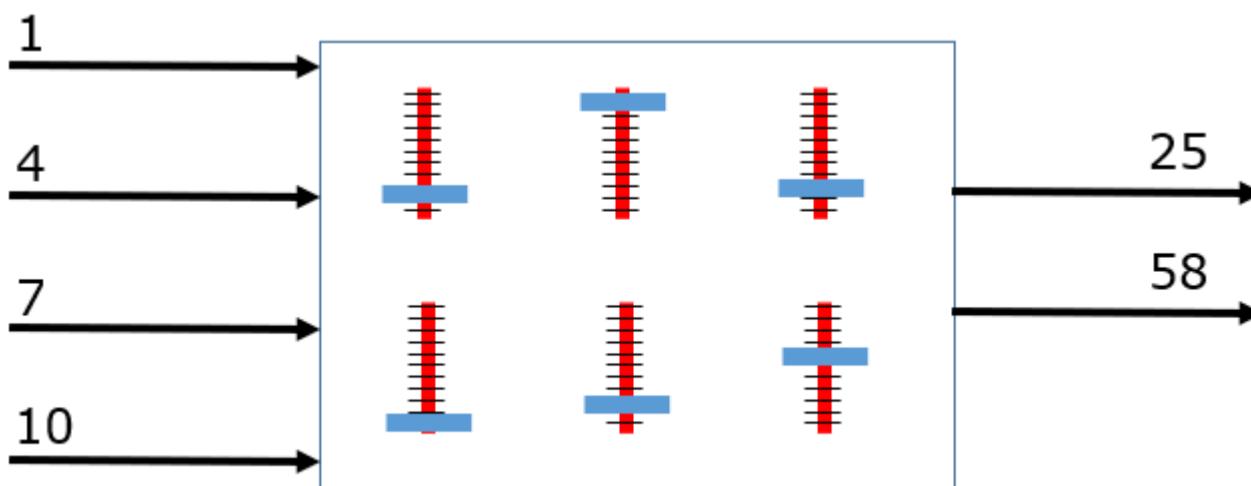


Ilustración 304: Esos pesos ya operan con el ejemplo A

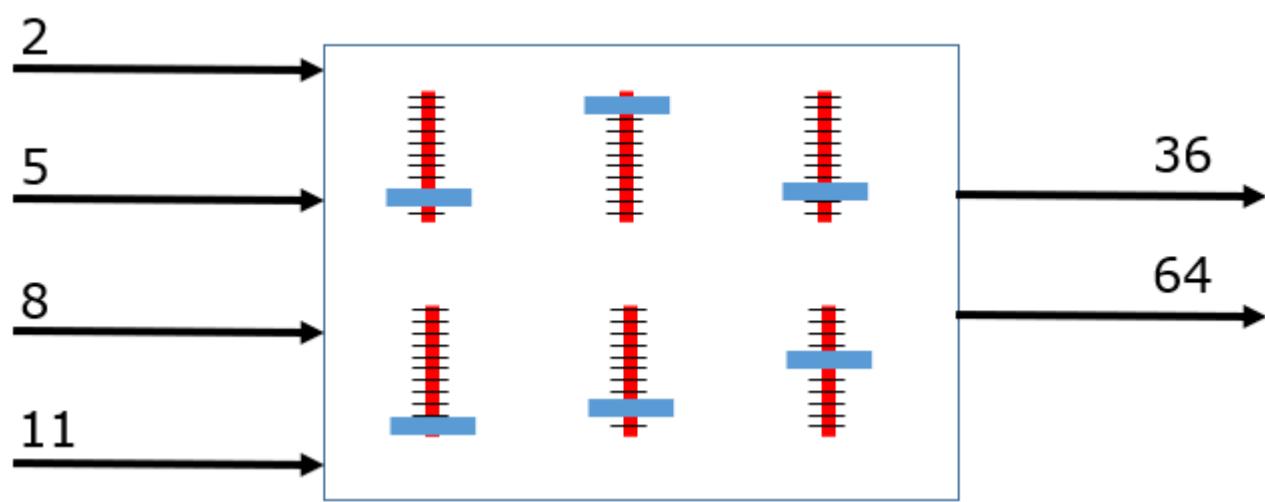


Ilustración 305: Los mismos pesos funcionan para el ejemplo B

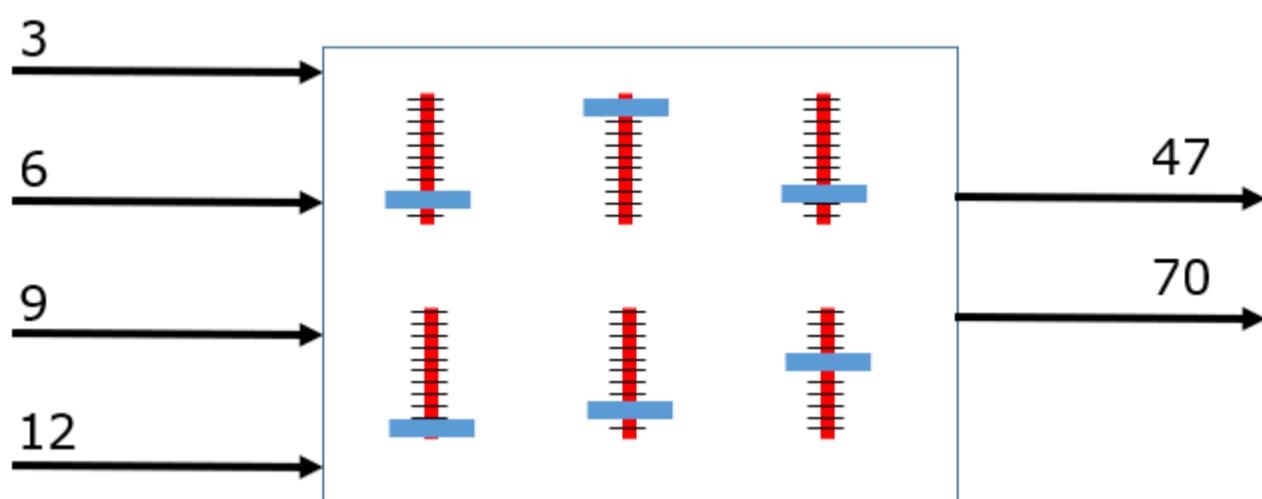


Ilustración 306: Y esos mismos pesos funcionan para el ejemplo C

¿Cómo es el proceso? Al iniciar, esos pesos tienen valores al azar y poco a poco se van ajustando. Existen técnicas matemáticas que colaboran en encontrar esos pesos rápidamente porque de lo contrario, sería un ajuste al azar continuamente hasta que por suerte se encuentren los valores correctos.

Perceptrón simple

Se inicia con una neurona. Esta es su representación:

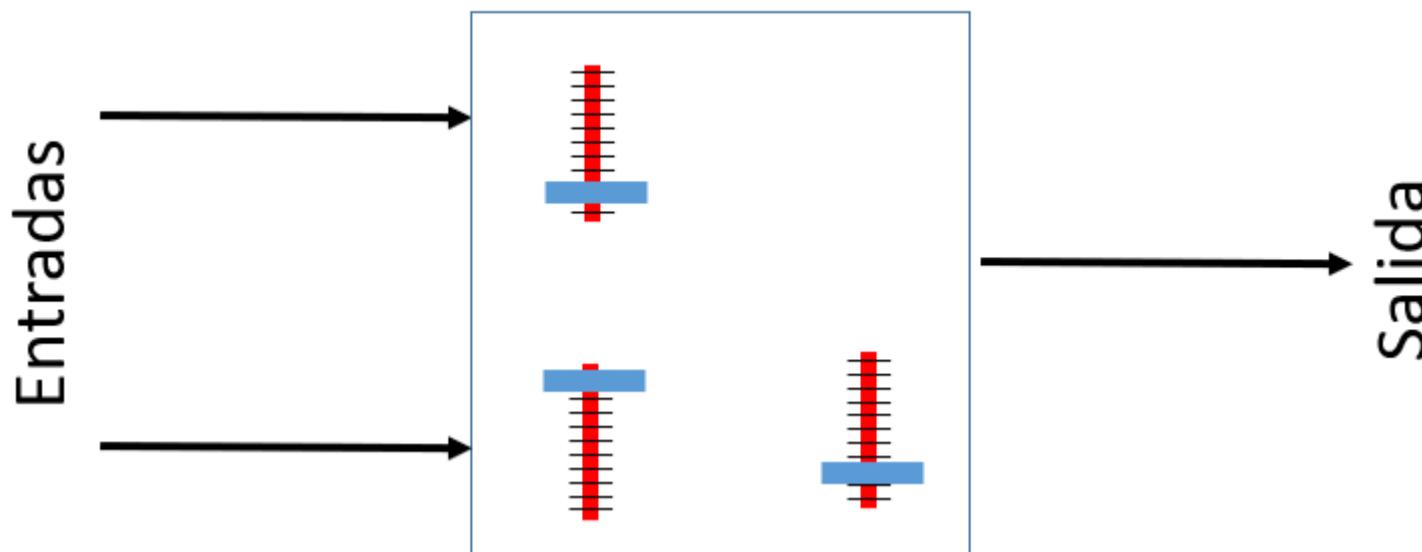


Ilustración 307: Perceptrón simple

Dos entradas, una salida y tres pesos. Se demostrará que esta neurona puede "aprender" como opera la tabla del AND:

A	B	Resultado (A AND B)
Verdadero	Verdadero	Verdadero
Verdadero	Falso	Falso
Falso	Verdadero	Falso
Falso	Falso	Falso

Esa neurona se le conoce con el nombre de Perceptrón Simple.

Paso 1: Hacerlo cuantitativo (1 es verdadero, 0 es falso)

A	B	Resultado (A AND B)
1	1	1
1	0	0
0	1	0
0	0	0

La razón de este cambio es que se requieren valores cuantitativos para ser usados dentro de fórmulas matemáticas.

Paso 2: Diseñando la neurona:

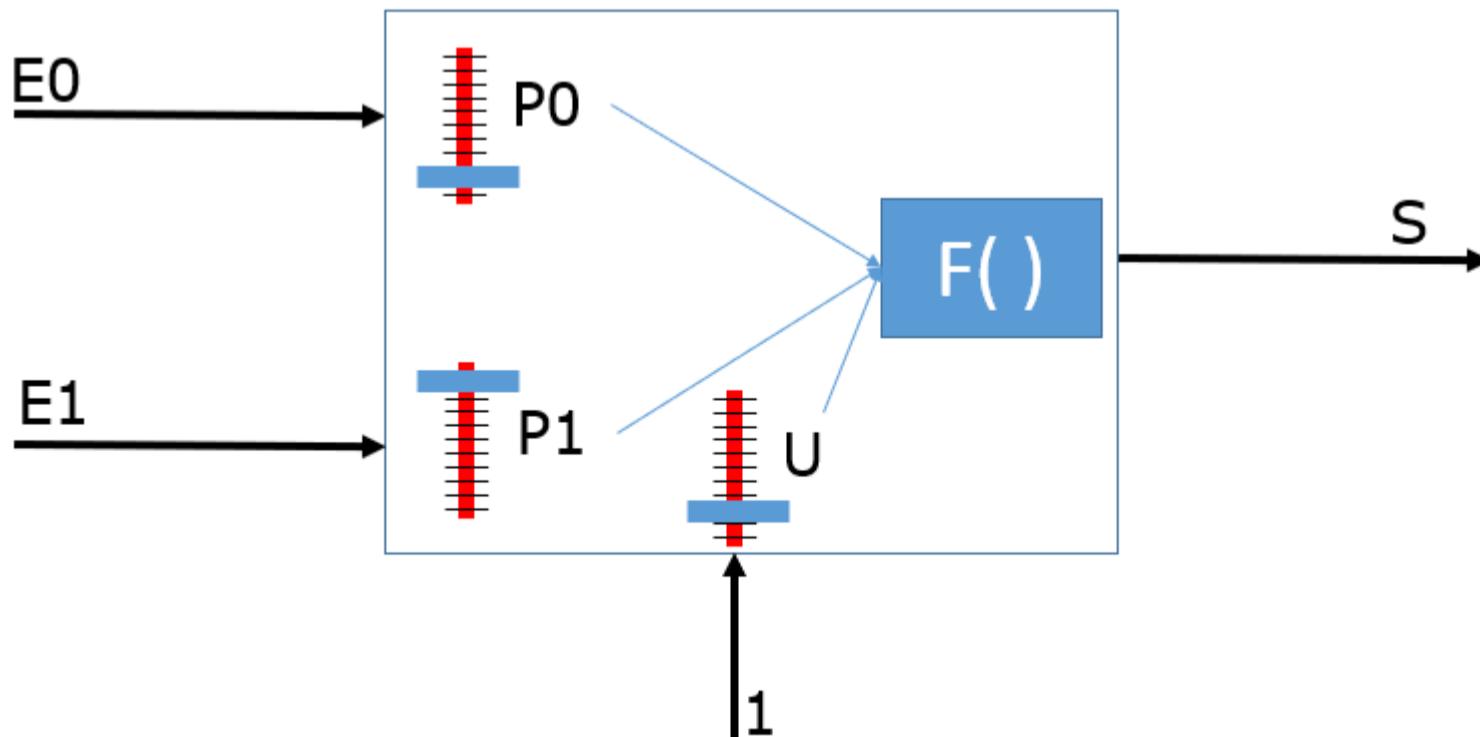


Ilustración 308: Funcionamiento del perceptrón simple

Un peso por cada entrada y se le adiciona una entrada interna que se llama umbral y tiene el valor de 1 con su propio peso.

E0 y E1 son las entradas

P0, P1 son los pesos de las entradas

U es el peso del umbral

S es la salida

F() es la función que le da el valor a S

Paso 3: Haciendo los cálculos:

La salida S se calcula con la siguiente fórmula matemática

$$S = F(E0 * P0 + E1 * P1 + 1 * U)$$

Se inicia con la primera regla de la tabla AND (verdadero y verdadero, da verdadero), en este caso se ingresa 1 y 1, la salida debería ser 1.

E0 = 1 (verdadero)

E1 = 1 (verdadero)

P0 = 0.6172 (un valor al azar)

P1 = 0.4501 (un valor real al azar)

U = 0.3789 (un valor real al azar)

Entonces la salida sería:

$$S = F(E0 * P0 + E1 * P1 + 1 * U)$$

$$S = F(1 * 0.6172 + 1 * 0.4501 + 1 * 0.3789)$$

$$S = F(1.4462)$$

¿Y que es F()? una función que podría ser matemática o un algoritmo. Así:

```
Función F(Valor)
Inicio
    Si Valor > 0.5 entonces
        retorne 1
    de lo contrario
        retorne 0
    fin si
Fin
```

Continuando con el ejemplo:

$$S = F(1.4462)$$

$$S = 1$$

Y ese es el valor esperado. Los pesos funcionan para esas entradas.

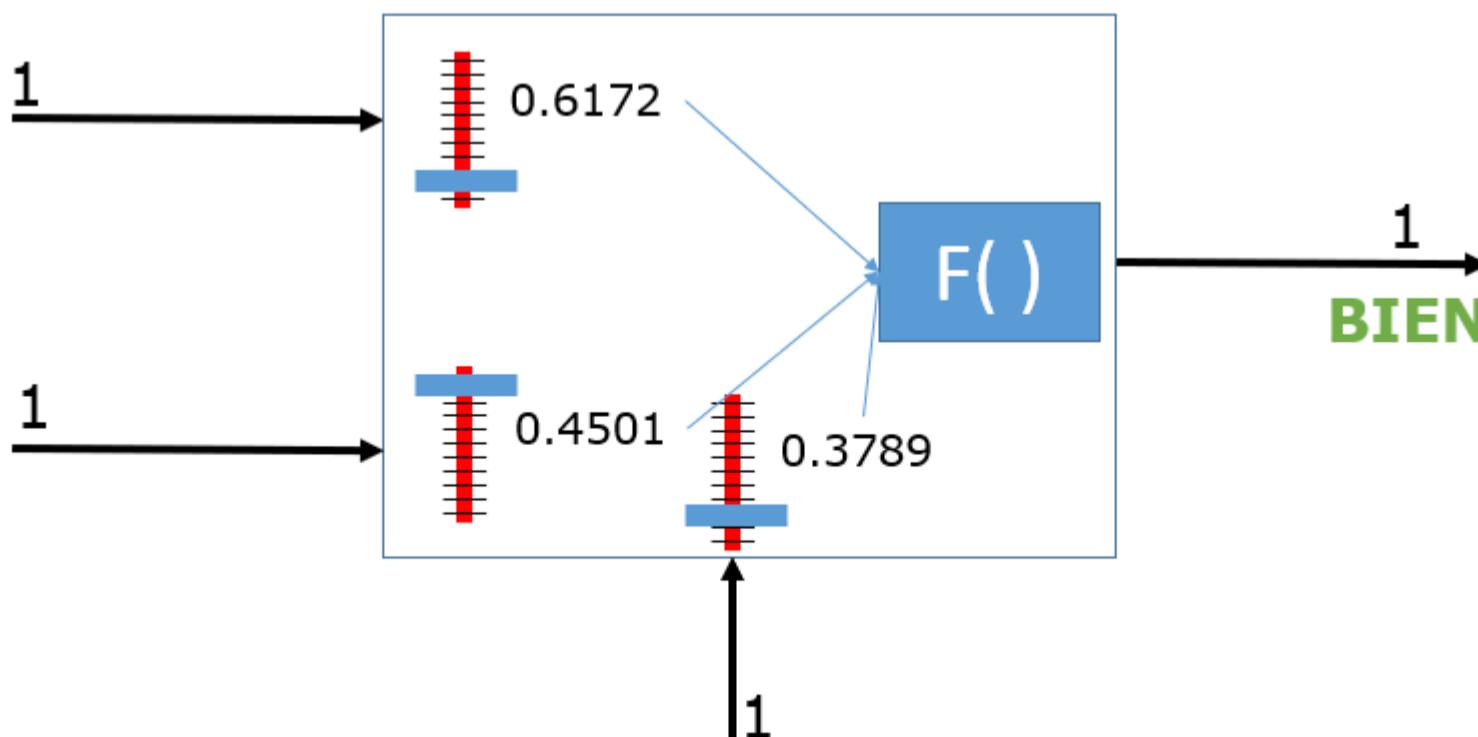


Ilustración 309: Los pesos funcionan para esa regla

¿Funcionarán esos pesos para las otras reglas de la tabla del AND? Se prueba entonces Verdadero y Falso, debería dar Falso

E0 = 1 (verdadero)

E1 = 0 (falso)

S = F(E0 * P0 + E1 * P1 + 1 * U)

S = F(1 * 0.6172 + 0 * 0.4501 + 1 * 0.3789)

S = F(0.9961)

S = 1

No, no funcionó, debería haber dado cero.

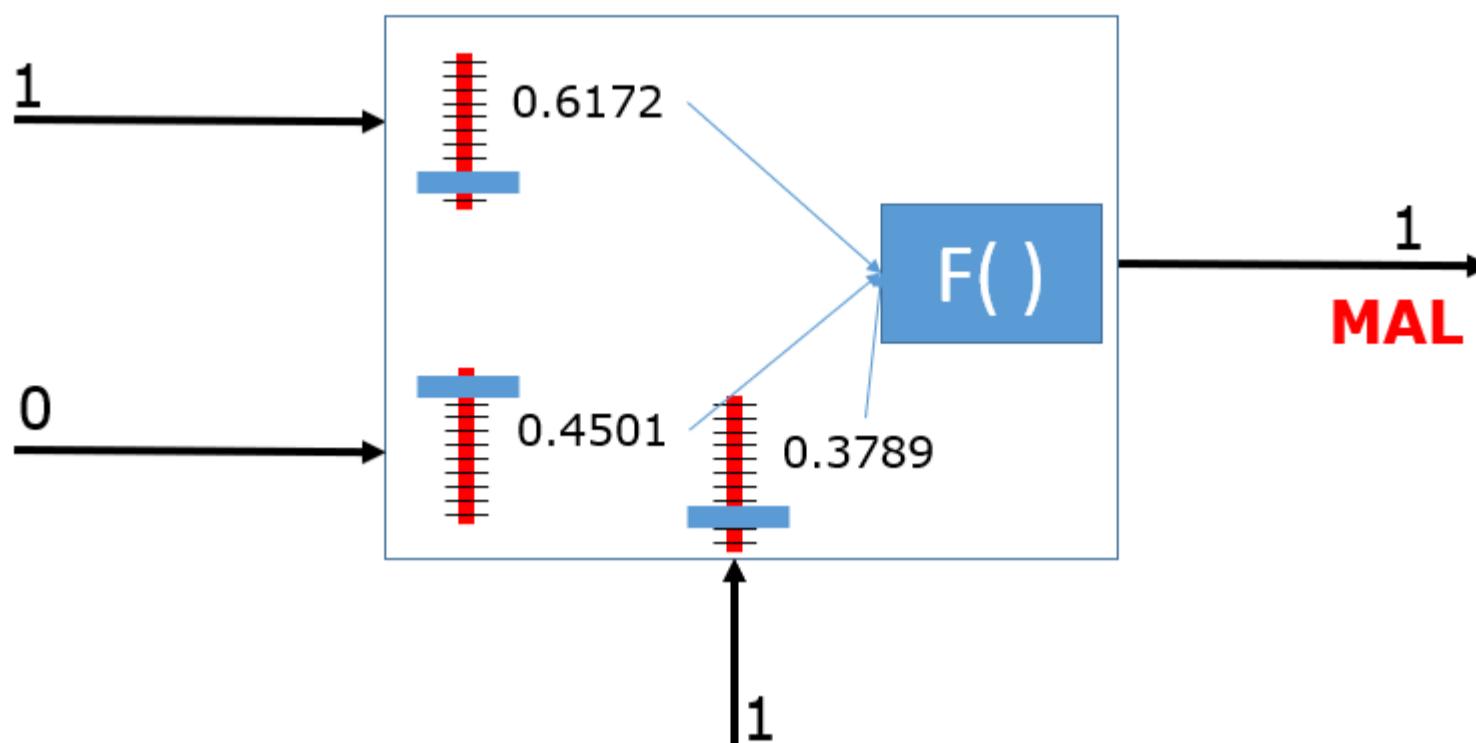


Ilustración 310: Esos pesos fallan con la segunda regla

¿Qué sigue? Habrá que utilizar otros valores para los pesos. Una forma es darle otros valores al azar. Ejecutar de nuevo el proceso, probar con todas las reglas hasta que finalmente de las salidas esperadas.

K/001.cs

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Único generador de números aleatorios
            Random Azar = new();

            //Tabla AND
            int[][] Entradas = new int[][][] {
                new int[] {1, 1},
                new int[] {1, 0},
                new int[] {0, 1},
                new int[] {0, 0}
            };
            int[] Salidas = new int[] { 1, 0, 0, 0 };

            //Los pesos
            double P0, P1, U;

            //Mantiene el proceso activo
            bool Proceso;

            //Número de iteraciones
            int Iteracion = 0;

            //Hasta que aprenda la tabla AND
            do {
                Iteracion++;

                //Pesos al azar
                P0 = Azar.NextDouble();
                P1 = Azar.NextDouble();
                U = Azar.NextDouble();

                //Prueba la tabla AND
                Proceso = false;
                for (int Contador = 0; Contador < Entradas.GetLength(0); Contador++) {

                    //Calcula el valor de entrada a la función
                    double Operacion = Entradas[Contador][0] * P0 + Entradas[Contador][1] * P1 + U;

                    //Función de activación
                    if (Operacion > 0.5)
                        Salidas[Contador] = 1;
                    else
                        Salidas[Contador] = 0;

                    if (Salidas[Contador] == Entradas[Contador][2])
                        Proceso = true;
                }

                if (Proceso)
                    break;
            }
        }
    }
}
```

```

        int SalidaEntera = Operacion > 0.5 ? 1 : 0;

        //Si la salida no coincide con lo esperado continua con la búsqueda de pesos.
        if (SalidaEntera != Salidas[Contador]) {
            Proceso = true;
            break;
        }
    } while (Proceso);

    //Muestra aprendizaje perceptrón simple
    for (int Contador = 0; Contador < Entradas.GetLength(0); Contador++) {
        double Operacion = Entradas[Contador][0] * P0 + Entradas[Contador][1] * P1 + U;

        //Función de activación
        int SalidaEntera = Operacion > 0.5 ? 1 : 0;

        Console.WriteLine("Entradas: " + Entradas[Contador][0] + " y " + Entradas[Contador][1] + " = " +
        Salidas[Contador] + " perceptron: " + SalidaEntera);
    }

    Console.WriteLine("Pesos encontrados P0= " + P0 + " P1= " + P1 + " U= " + U);
    Console.WriteLine("Total Iteraciones: " + Iteracion);
}
}
}

```

Consola de depuración de Mi... X + ▾

Entradas: 1 y 1 = 1 perceptron: 1
 Entradas: 1 y 0 = 0 perceptron: 0
 Entradas: 0 y 1 = 0 perceptron: 0
 Entradas: 0 y 0 = 0 perceptron: 0
 Pesos encontrados P0= 0,1864582462399923 P1= 0,32470435638036177 U= 0,1552258204430117
 Total Iteraciones: 13

Ilustración 311: Dar con los pesos con sólo azar

Consola de depuración de Mi... X + ▾

Entradas: 1 y 1 = 1 perceptron: 1
 Entradas: 1 y 0 = 0 perceptron: 0
 Entradas: 0 y 1 = 0 perceptron: 0
 Entradas: 0 y 0 = 0 perceptron: 0
 Pesos encontrados P0= 0,3709107516082192 P1= 0,22883767769211982 U= 0,04995841495690845
 Total Iteraciones: 106

Ilustración 312: Dar con los pesos con sólo azar

Consola de depuración de Mi... X + ▾

Entradas: 1 y 1 = 1 perceptron: 1
 Entradas: 1 y 0 = 0 perceptron: 0
 Entradas: 0 y 1 = 0 perceptron: 0
 Entradas: 0 y 0 = 0 perceptron: 0
 Pesos encontrados P0= 0,2729037576898937 P1= 0,3652514916457045 U= 0,12023829796275443
 Total Iteraciones: 65

Ilustración 313: Dar con los pesos con sólo azar

Los valores de los pesos no es una respuesta única, pueden ser distintos y son números reales (en C# se implementaron de tipo double). También se observa que en una ejecución requirió 106 iteraciones. El cambio de pesos sucede en estas líneas:

```

P0 = azar.NextDouble();
P1 = azar.NextDouble();

```

```
U = azar.NextDouble();
```

En caso de que no funcionasen los pesos, el programa simplemente los cambiaba al azar en un valor que oscila entre 0 y 1. Eso puede ser muy ineficiente y riesgoso porque limita los valores a estar entre 0 y 1 ¿Y si los pesos requieren valores mucho más altos o bajos?

Afortunadamente, hay un método matemático que minimiza el uso del azar y puede dar con valores de los pesos en cualquier rango. ¿Cómo funciona? Al principio los pesos tienen un valor al azar, pero de allí en adelante el cálculo de esos pesos se basa en comparar la salida esperada con la salida obtenida, si difieren, ese error sirve para ir cuadrando poco a poco los pesos.

Fórmula de Frank Rosenblatt

Los pesos se cambian haciendo uso de una fórmula matemática:

```
Error = SalidaEsperada - SalidaReal  
Si Error != cero entonces  
    P0 = P0 + tasaAprende * Error * E0  
    P1 = P1 + tasaAprende * Error * E1  
    U = U + tasaAprende * Error * 1  
Fin Si
```

tasaAprende es un valor constante de tipo real y de valor entre 0 y 1 (sin tomar el 0, ni el 1). A continuación, el código en C#

K/002.cs

```
namespace Ejemplo {  
    internal class Program {  
        static void Main() {  
            //Único generador de números aleatorios  
            Random Azar = new();  
  
            //Tabla AND  
            int[][] Entradas = new int[][][] {  
                new int[] {1, 1},  
                new int[] {1, 0},  
                new int[] {0, 1},  
                new int[] {0, 0}  
            };  
            int[] Salidas = new int[] { 1, 0, 0, 0 };  
  
            //Los pesos  
            double P0, P1, U;  
  
            //Mantiene el proceso activo  
            bool Proceso;  
  
            //Número de iteraciones  
            int Iteracion = 0;  
  
            //Tasa de aprendizaje  
            double TasaAprende = 0.3;  
  
            //Pesos inician al azar  
            P0 = Azar.NextDouble();  
            P1 = Azar.NextDouble();  
            U = Azar.NextDouble();  
  
            //Hasta que aprenda la tabla AND  
            do {  
                Iteracion++;  
  
                //Prueba la tabla AND  
                Proceso = false;  
                for (int Contador = 0; Contador < Entradas.GetLength(0); Contador++) {  
  
                    //Calcula el valor de entrada a la función  
                    double Operacion = Entradas[Contador][0] * P0 + Entradas[Contador][1] * P1 + U;  
  
                    //Función de activación  
                    int SalidaEntera = Operacion > 0.5 ? 1 : 0;  
  
                    //El error  
                    int Error = Salidas[Contador] - SalidaEntera;  
  
                    //Si hay error, cambia los pesos con la Tasa de Aprendizaje  
                    if (Error != 0) {  
                        P0 += TasaAprende * Error * Entradas[Contador][0];  
                        P1 += TasaAprende * Error * Entradas[Contador][1];  
                        U += TasaAprende * Error * 1;  
                        Proceso = true;  
                    }  
                }  
            } while (!Proceso);  
        }  
    }  
}
```

```

        }
    } while (Proceso);

    //Muestra aprendizaje perceptrón simple
    for (int Contador = 0; Contador < Entradas.GetLength(0); Contador++) {
        double Operacion = Entradas[Contador][0] * P0 + Entradas[Contador][1] * P1 + U;

        //Función de activación
        int SalidaEntera = Operacion > 0.5 ? 1 : 0;

        Console.WriteLine("Entradas: " + Entradas[Contador][0] + " y " + Entradas[Contador][1]
+ " = " +
            Salidas[Contador] + " perceptron: " + SalidaEntera);
    }

    Console.WriteLine("Pesos encontrados P0= " + P0 + " P1= " + P1 + " U= " + U);
    Console.WriteLine("Total Iteraciones: " + Iteracion);
}
}
}

```

```

Entradas: 1 y 1 = 1 perceptron: 1
Entradas: 1 y 0 = 0 perceptron: 0
Entradas: 0 y 1 = 0 perceptron: 0
Entradas: 0 y 0 = 0 perceptron: 0
Pesos encontrados P0= 0,5218730868070858 P1= 0,6514357002751983 U= -0,4271222277132198
Total Iteraciones: 7

```

Ilustración 314: Encontrando los pesos más rápido

```

Entradas: 1 y 1 = 1 perceptron: 1
Entradas: 1 y 0 = 0 perceptron: 0
Entradas: 0 y 1 = 0 perceptron: 0
Entradas: 0 y 0 = 0 perceptron: 0
Pesos encontrados P0= 0,42514198541349996 P1= 0,6067804235862224 U= -0,3399274573373506
Total Iteraciones: 7

```

Ilustración 315: Encontrando los pesos más rápido

```

Entradas: 1 y 1 = 1 perceptron: 1
Entradas: 1 y 0 = 0 perceptron: 0
Entradas: 0 y 1 = 0 perceptron: 0
Entradas: 0 y 0 = 0 perceptron: 0
Pesos encontrados P0= 0,5705308605404625 P1= 0,5039279097470348 U= -0,5078508611506496
Total Iteraciones: 3

```

Ilustración 316: Encontrando los pesos más rápido

Como se puede observar, se necesitan menos iteraciones en promedio usando la fórmula para hallar los pesos.

Perceptrón simple: Aprendiendo la tabla del OR

El ejemplo anterior el perceptrón simple aprendía la tabla AND, ¿y con la OR?

A	B	Resultado (A OR B)
Verdadero	Verdadero	Verdadero
Verdadero	Falso	Verdadero
Falso	Verdadero	Verdadero
Falso	Falso	Falso

Paso 1: Volver cuantitativa esa tabla (1 es verdadero, 0 es falso)

A	B	Resultado (A OR B)
1	1	1
1	0	1
0	1	1
0	0	0

Es sólo cambiar estas líneas del programa:

```
//Tabla OR
int[][] Entradas = new int[][] {
    new int[] {1, 1},
    new int[] {1, 0},
    new int[] {0, 1},
    new int[] {0, 0}
};
int[] Salidas = new int[] { 1, 1, 1, 0 };
```

Límites del Perceptrón Simple

El perceptrón simple tiene un límite: que sólo sirve cuando la solución se puede separar con **una** recta. Se explica a continuación:

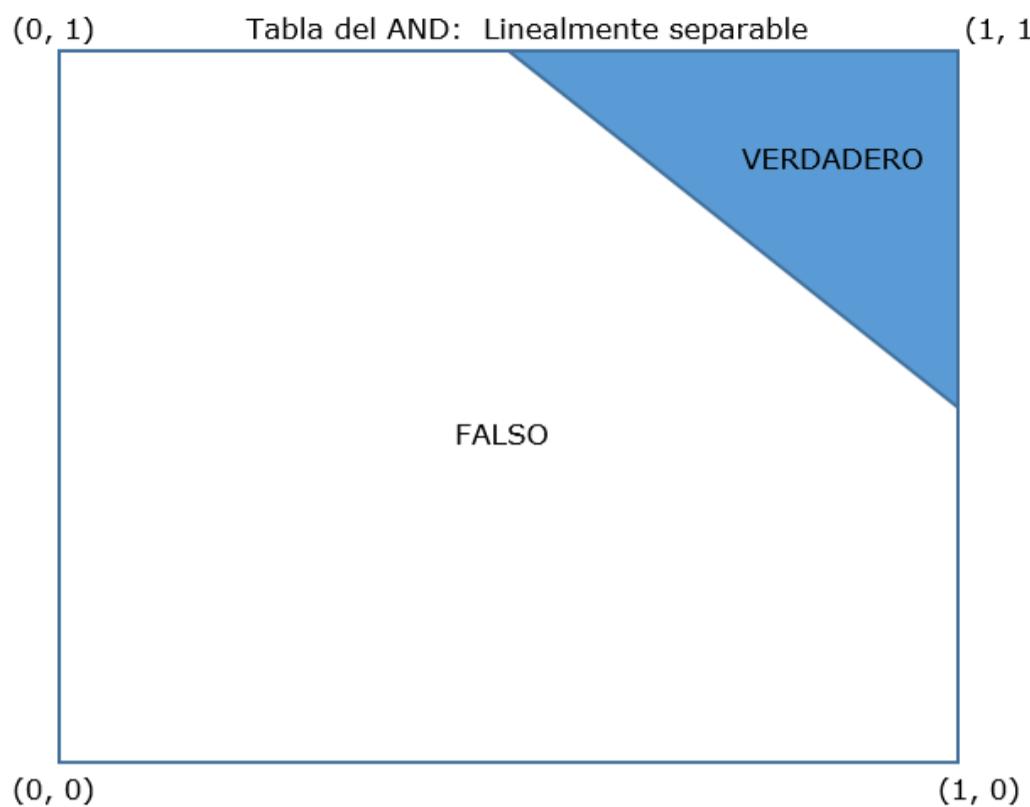


Ilustración 317: Tabla del AND

En cambio, si se quiere abordar un problema que requiera dos separaciones, no lo podría hacer el perceptrón simple. El ejemplo clásico es la tabla XOR:

A	B	Resultado (A XOR B)
Verdadero	Verdadero	Falso
Verdadero	Falso	Verdadero
Falso	Verdadero	Verdadero
Falso	Falso	Falso

Volviendo cuantitativa esa tabla:

A	B	Resultado (A XOR B)
1	1	0
1	0	1
0	1	1
0	0	0

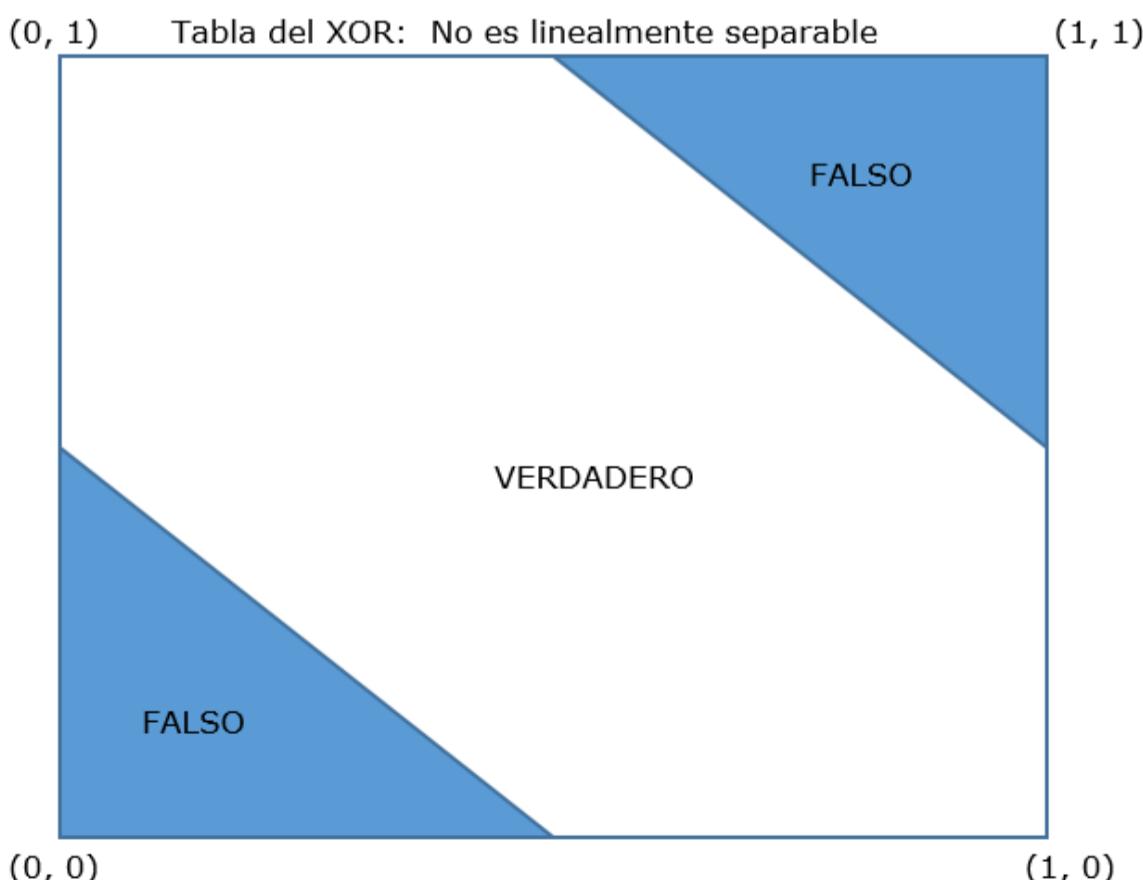


Ilustración 318: Tabla del XOR

Para solucionar ese problema es necesario usar más neuronas puestas en varias capas. Por ejemplo:

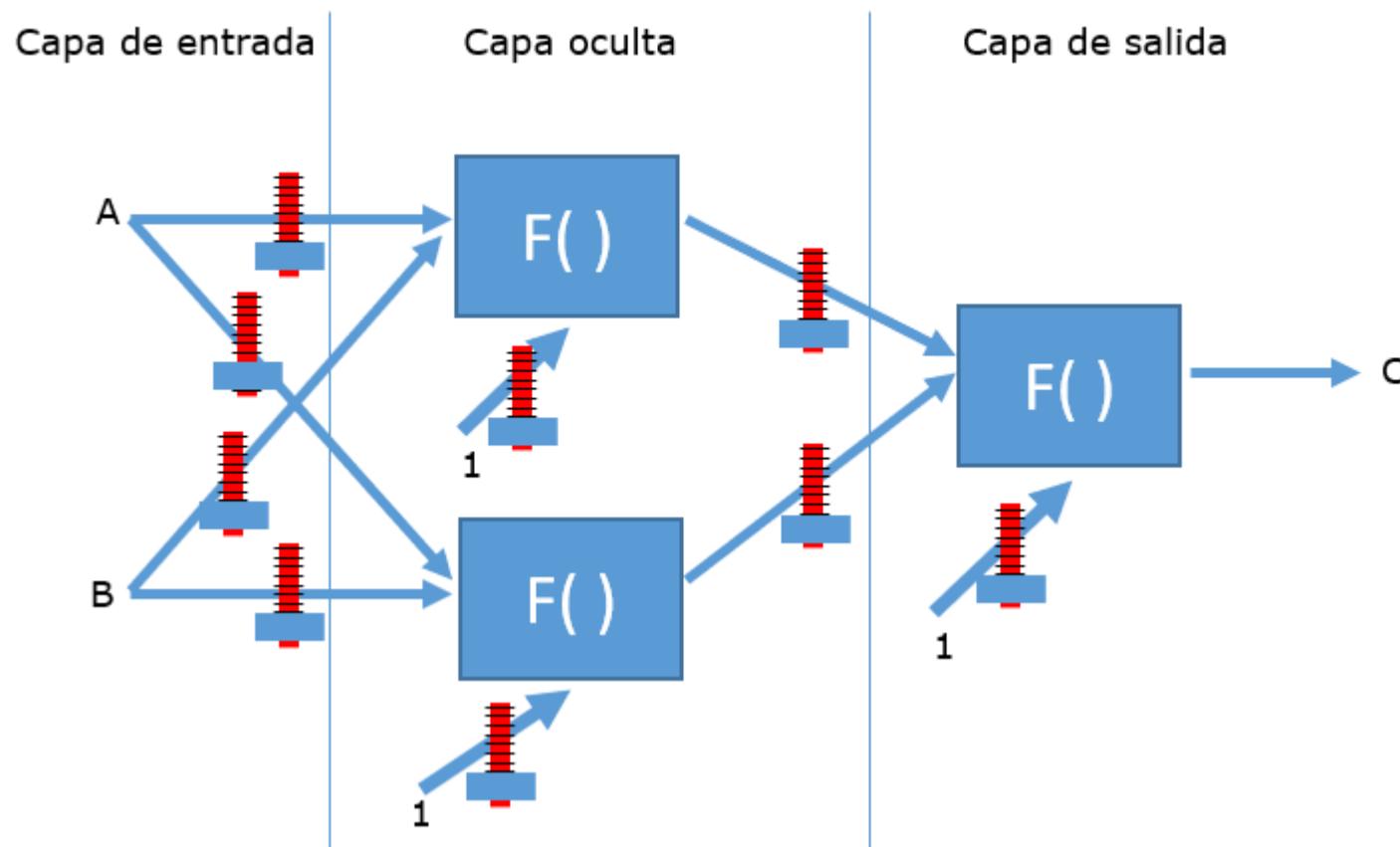


Ilustración 319: Red neuronal con 3 neuronas

O así

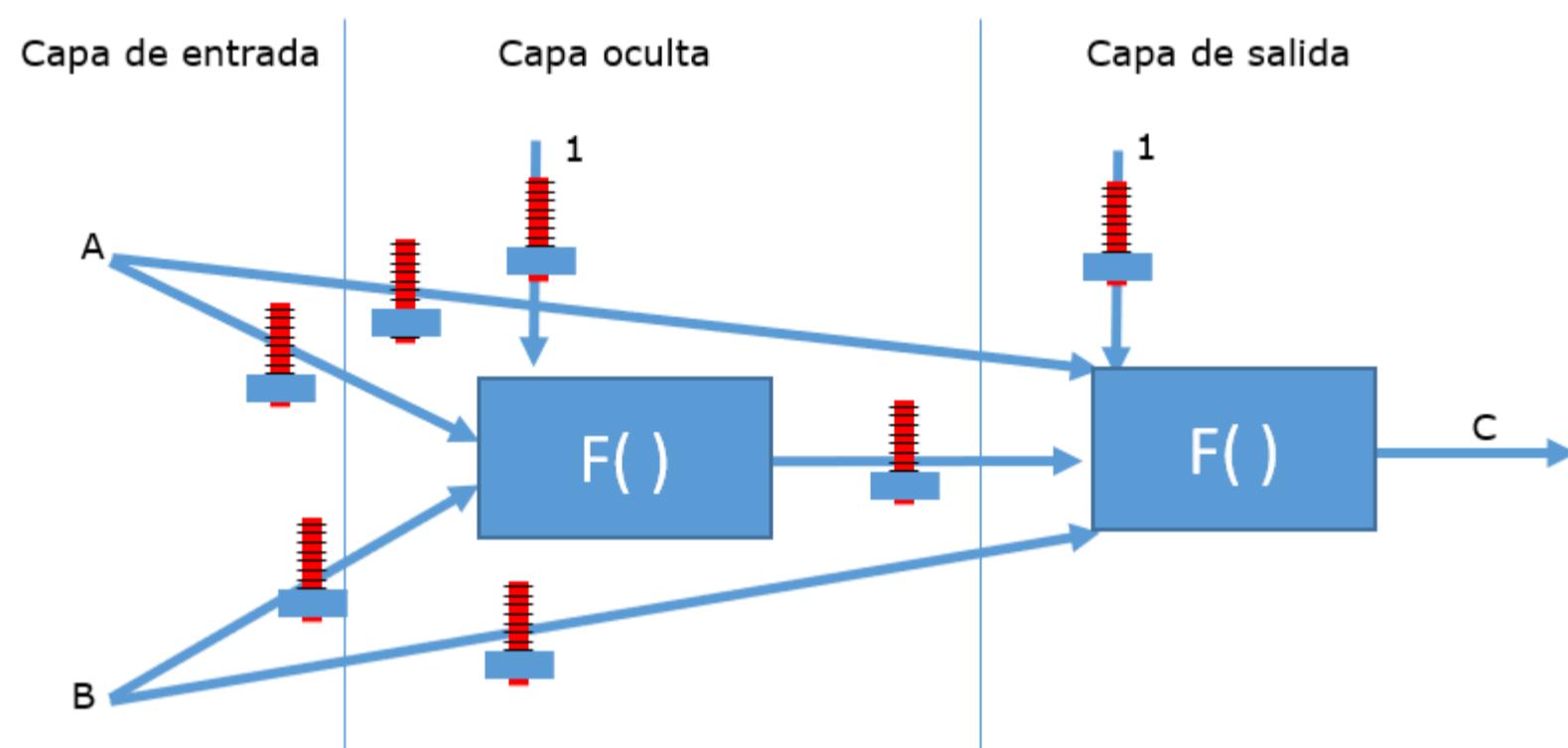


Ilustración 320: Red neuronal con otro tipo de conexiones

Muchos más pesos, luego el reto es cómo dar con cada peso para que se cumplan las salidas. Hay entonces un modelo matemático para lograr esto.

Encontrando el mínimo en una ecuación

A continuación, se explica la matemática que ayudará a deducir los pesos en una red neuronal.

Para dar con el mínimo de una ecuación se hace uso de las derivadas. Por ejemplo, dada la ecuación:

$$y = 5 * x^2 - 7 * x - 13$$

Tabla de datos y gráfico

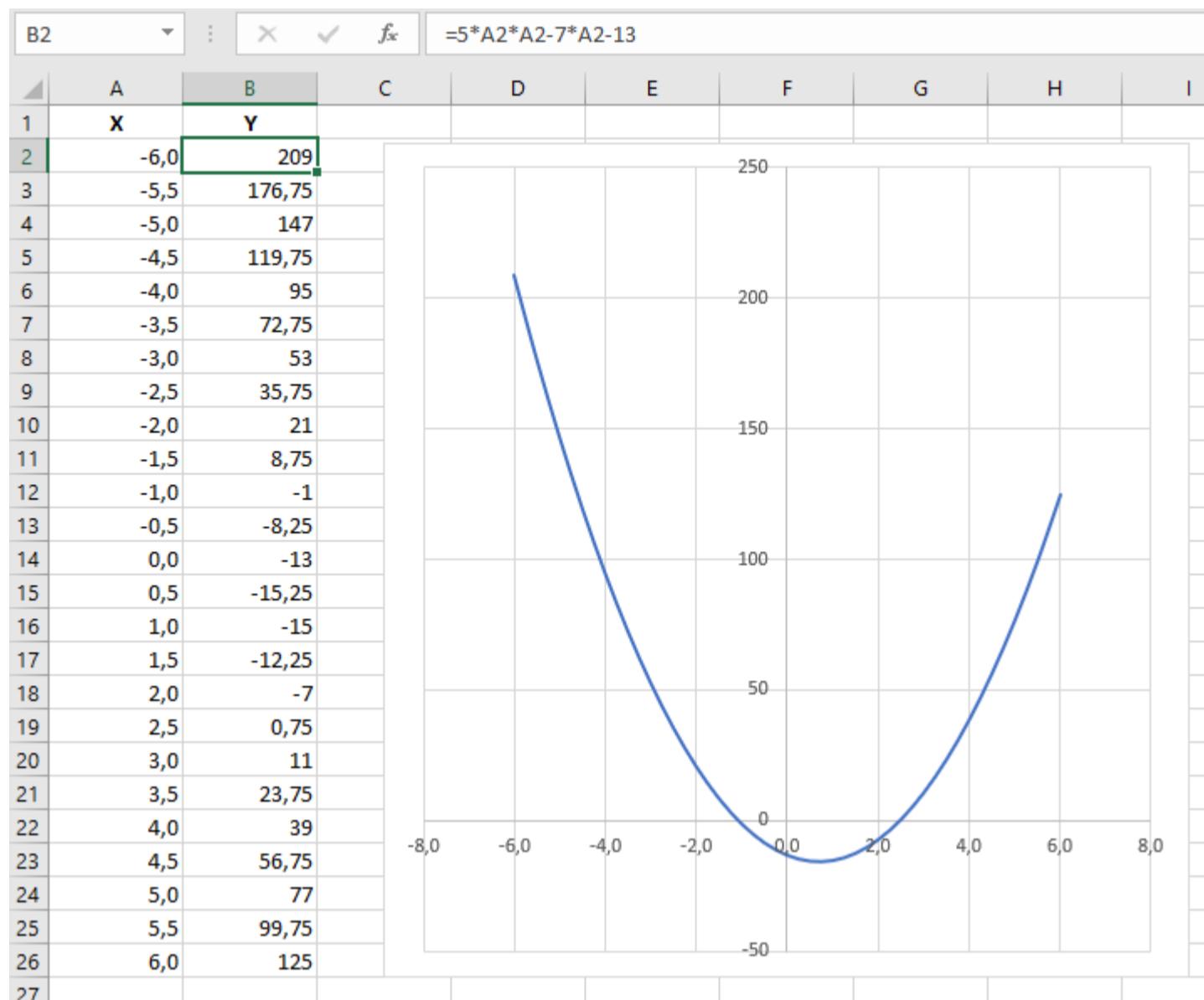


Ilustración 321: Tabla y gráfico de la ecuación hechos con Microsoft Excel

Si se quiere dar con el valor de X para que Y sea el mínimo valor, el primer paso es derivar

$$y' = 10 * x - 7$$



derivate $5*x^2-7*x-13$

Extended Keyboard Upload Examples Random

Derivative:

$$\frac{d}{dx}(5x^2 - 7x - 13) = 10x - 7$$

Step-by-step solution

Ilustración 322: Derivada usando en WolframAlpha

Luego esa derivada se iguala a cero

$$0 = 10 * x - 7$$

Se resuelve el valor de x

$$x = 7/10$$

$$x = 0.7$$

Se deduce el valor de x con el que se obtiene el mínimo valor de y

$$y = 5 * x^2 - 7 * x - 13$$

$$y = 5 * 0.7^2 - 7 * 0.7 - 13$$

$$y = -15.45$$

En este caso fue fácil dar con la derivada, porque fue un polinomio de grado 2, el problema sucede cuando la ecuación es compleja, derivarla se torna un desafío y despejar X sea muy complicado.

Otra forma de dar con el mínimo es iniciar con algún punto X al azar, por ejemplo, $X = 1.0$

Valor de X	$y = 5 * x^2 - 7 * x - 13$
1.0	-15

Luego un desplazamiento tanto a la izquierda como a la derecha de 0.5 en 0.5, es decir, $x=0.5$ y $x=1.5$

Valor de X	$y = 5 * x^2 - 7 * x - 13$
0.5	-15.25
1.0	-15
1.5	-12,25

Se obtiene un nuevo valor de X más prometedor que es 0.5, luego se repite el procedimiento, izquierda y derecha, es decir, $x=0.0$ y $x=1.0$

Valor de X	$y = 5 * x^2 - 7 * x - 13$
0.0	-13
0.5	-15.25
1.0	-15

El valor de 0.5 se mantiene como el mejor, luego se hace izquierda y derecha a un paso menor de 0.25

Valor de X	$y = 5 * x^2 - 7 * x - 13$
0.25	-14,4375
0.50	-15.25
0.75	-15.4375

El valor de $x=0.75$ es el que muestra mejor comportamiento, luego se hace izquierda y derecha a un paso de 0.25

Valor de X	$y = 5 * x^2 - 7 * x - 13$
0.50	-15.25
0.75	-15.4375
1.00	-15

Sigue $x=0.75$ como mejor valor, luego se prueba a izquierda y derecha, pero en una variación menor de 0.125

Valor de X	$y = 5 * x^2 - 7 * x - 13$
0.625	-15.421875
0.75	-15.4375
0.875	-15.296875

Sigue $x=0.75$ como mejor valor, luego se prueba a izquierda y derecha, pero en una variación menor de 0.0625

Valor de X	$y = 5 * x^2 - 7 * x - 13$
0.6875	-15.4492188
0.75	-15.4375
0.8125	-15.3867188

Ahora es $x=0.6875$ como mejor valor, luego se prueba a izquierda y derecha en una variación de 0.0625

Valor de X	$y = 5 * x^2 - 7 * x - 13$
0.625	-15.421875
0.6875	-15.4492188
0.75	-15.4375

Sigue $x=0.6875$ como mejor valor, luego se prueba a izquierda y derecha, pero en una variación menor de 0.03125

Valor de X	$y = 5 * x^2 - 7 * x - 13$
0.65625	-15.4404297
0.6875	-15.4492188
0.71875	-15.4482422

Sigue $x=0.6875$ como mejor valor, luego se prueba a izquierda y derecha, pero en una variación menor de 0.015625

Valor de X	$y = 5 * x^2 - 7 * x - 13$
0.671875	-15.4460449
0.6875	-15.4492188
0.703125	-15.4499512

Ahora es $x=0.703125$ como mejor valor. Este método poco a poco se aproxima a $x=0.7$ que es el resultado que se dedujo con las derivadas. Esta es su implementación en C#:

K/003.cs

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            double X = 1; //valor inicial
            double Yini = Ecuacion(X);
            double Variacion = 1;

            while (Math.Abs(Variacion) > 0.00001) {
                double Ysigue = Ecuacion(X + Variacion);

                //Si en vez de disminuir Y, lo que hace es aumentar, cambia de dirección a un paso menor
                if (Ysigue > Yini) {
                    Variacion *= -1;
                    Variacion /= 10;
                }
                else { //Está disminuyendo Y
                    Yini = Ysigue;
                    X += Variacion;
                    Console.WriteLine("X: " + X.ToString() + " Y:" + Yini.ToString());
                }
            }
            Console.WriteLine("Respuesta: " + X.ToString());
        }

        //Ecuación a analizar
        static double Ecuacion(double X) {
            return 5 * X * X - 7 * X - 13;
        }
    }
}
```

```
X: 0,9 Y:-15,25
X: 0,8 Y:-15,4
X: 0,7000000000000001 Y:-15,45
Respuesta: 0,7000000000000001
```

Ilustración 323: Buscando el mínimo

Descenso del gradiente

Anteriormente se mostró, con las aproximaciones, como buscar el mínimo valor de Y modificando el valor de X, ya sea yendo por la izquierda (disminuyendo) o por la derecha (aumentando). Matemáticamente para saber en qué dirección ir, es con esta expresión:

$$\Delta x = -y'$$

¿Qué significa? Que X debe modificarse en contra de la derivada de la ecuación.

¿Por qué? La derivada muestra la tangente que pasa por el punto que se seleccionó al azar al inicio. Esa tangente es una línea recta y como toda línea recta tiene una pendiente. Si la pendiente es positiva entonces X se debe ir hacia la izquierda (el valor de X debe disminuir), en cambio, si la pendiente es negativa entonces X debe ir hacia la derecha (el valor de X debe aumentar). Con esa indicación ya se sabe por dónde ir para dar con el valor de X que obtiene el mínimo Y .

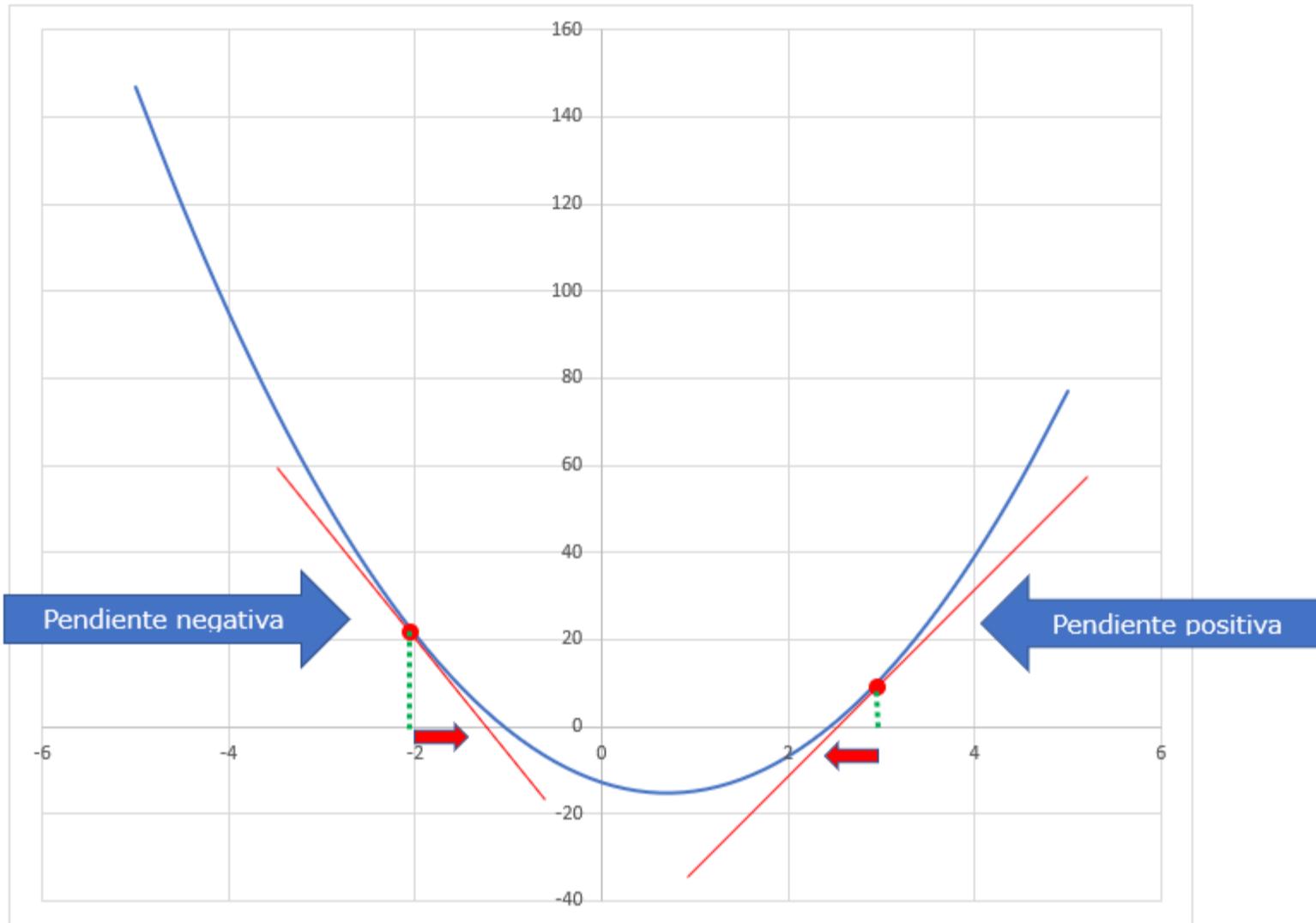


Ilustración 324: Pendientes

Para dar con el nuevo valor de X esta sería la expresión:

$$x_{nuevo} = x_{anterior} + \Delta x$$

Reemplazando

$$x_{nuevo} = x_{anterior} - y'$$

EJEMPLO

Con la ecuación anterior

$$y = 5 * x^2 - 7 * x - 13$$

$$y' = 10 * x - 7$$

$$x_{nuevo} = x_{anterior} - y'$$

$$x_{nuevo} = x_{anterior} - (10 * x - 7)$$

X inicia en 0.4 por ejemplo, luego

$$x_{nuevo} = 0.4 - (10 * 0.4 - 7)$$

$$x_{nuevo} = -3.4$$

Ahora hay un nuevo valor para X que es -2. En la siguiente tabla se muestra como progresa X

Xanterior	Xnuevo	Y
0.4	3.4	21
3.4	-23.6	2937
-23.6	219.4	239133
219.4	-1967.6	19371009
-1967.6	17715.4	1569052965
17715.4	-159431.6	1.27093E+11
-159431.6	1434891.4	1.02946E+13
1434891.4	-12914015.6	8.33859E+14
-12914015.6	116226147.4	6.75426E+16
116226147.4	-1046035320	5.47095E+18
-1046035320	9414317883	4.43147E+20

El valor de X se dispara, se vuelve extremo hacia la izquierda o derecha. Se debe arreglar agregando una constante a la ecuación:

$$x_{nuevo} = x_{anterior} - \alpha * y'$$

Se agrega entonces un α que es una constante muy pequeña, por ejemplo $\alpha=0.05$ y esto es lo que sucede

$$x_{nuevo} = x_{anterior} - 0.05 * (10 * x_{anterior} - 7)$$

Xanterior	Xnuevo	Y
0.4	0.55	-15.3375
0.55	0.625	-15.421875
0.625	0.6625	-15.4429688
0.6625	0.68125	-15.4482422
0.68125	0.690625	-15.4495605
0.690625	0.6953125	-15.4498901
0.6953125	0.69765625	-15.4499725
0.69765625	0.698828125	-15.4499931
0.698828125	0.699414063	-15.4499983
0.699414063	0.699707031	-15.4499996
0.699707031	0.699853516	-15.4499999

Tiene más sentido y se acerca a $X=0.7$ que es la respuesta correcta.

Este método se le conoce como el descenso del gradiente que se expresa así:

$$x_{nuevo} = x_{anterior} - \alpha * f'(x_{anterior})$$

En formato matemático

$$x_{n+1} = x_n - \alpha * f'(x_n)$$

Mínimos locales y globales

La siguiente curva es generada por el siguiente polinomio

$$y = 0.1 * x^6 + 0.6 * x^5 - 0.7 * x^4 - 6 * x^3 + 2 * x^2 + 2 * x + 1$$

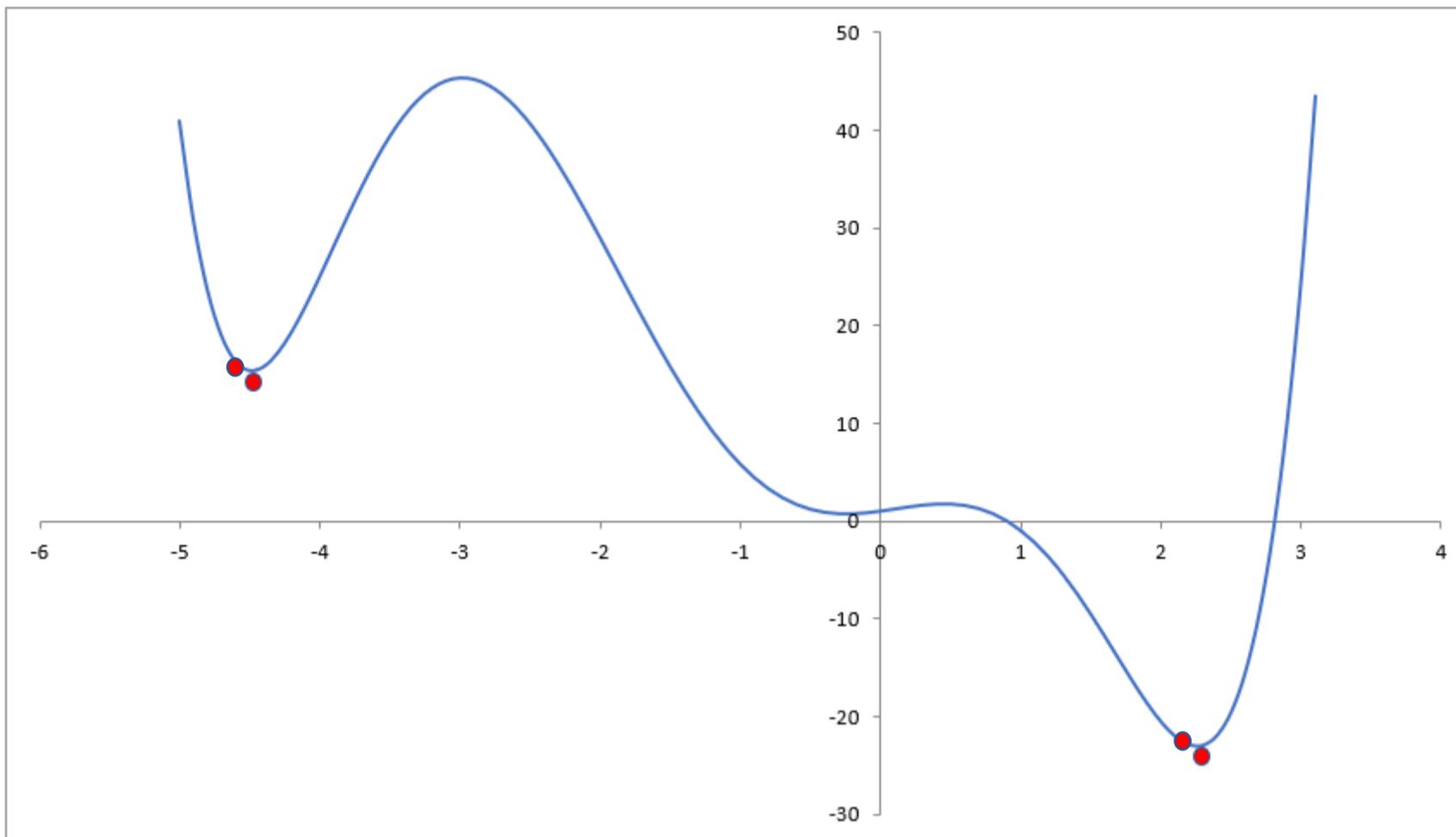


Ilustración 325: Gráfico de un polinomio de quinto grado

Se aprecian dos puntos donde claramente la curva desciende y vuelve a ascender (se han marcado con puntos en rojo), por supuesto, el segundo a la derecha es el mínimo real, pero ¿Qué pasaría si se hubiese hecho una búsqueda iniciando en $x=-4$? La respuesta es que el algoritmo se hubiese decantado por el mínimo de la izquierda:

$$x_{nuevo} = x_{anterior} - \alpha * y'$$

$$x_{nuevo} = x_{anterior} - 0.01 * (0.6 * x^5 + 3 * x^4 - 2.8 * x^3 - 18 * x^2 + 4 * x + 2)$$

Xanterior	Xnuevo	Y
-4	-4.308	25
-4.308	-4.4838	17.0485
-4.4838	-4.4815	15.3935
-4.4815	-4.4822	15.3933
-4.4822	-4.482	15.3933
-4.482	-4.482	15.3933

Este problema se le conoce como caer en mínimo local y también lo sufren los algoritmos evolutivos. Así que se deben probar otros valores de X para iniciar, si fuese X=2 se observa que si acierta con el mínimo real:

Xanterior	Xnuevo	Y
2	2.172	-20.6
2.172	2.24349	-22.777
2.24349	2.25489	-23.08
2.25489	2.25559	-23.087
2.25559	2.25562	-23.087
2.25562	2.25563	-23.087
2.25563	2.25563	-23.087

Fue fácil darse cuenta donde está el mínimo real viendo la gráfica, pero el problema estará vigente cuando no sea fácil generar el gráfico o peor aún, cuando no sea una sola variable independiente $f(x)$ sino varias, como funciones del tipo $f(a,b,c,d,e)$

Búsqueda de mínimos y redes neuronales

En la figura, hay dos entradas: A y B, y una salida: C, todo eso son constantes porque son los datos de entrenamiento, no hay control sobre estos. Lo que, si se puede variar, son los pesos. Si se quiere saber que tanto se debe ajustar cada peso, el procedimiento matemático de obtener mínimos, se enfoca solamente en esos pesos.

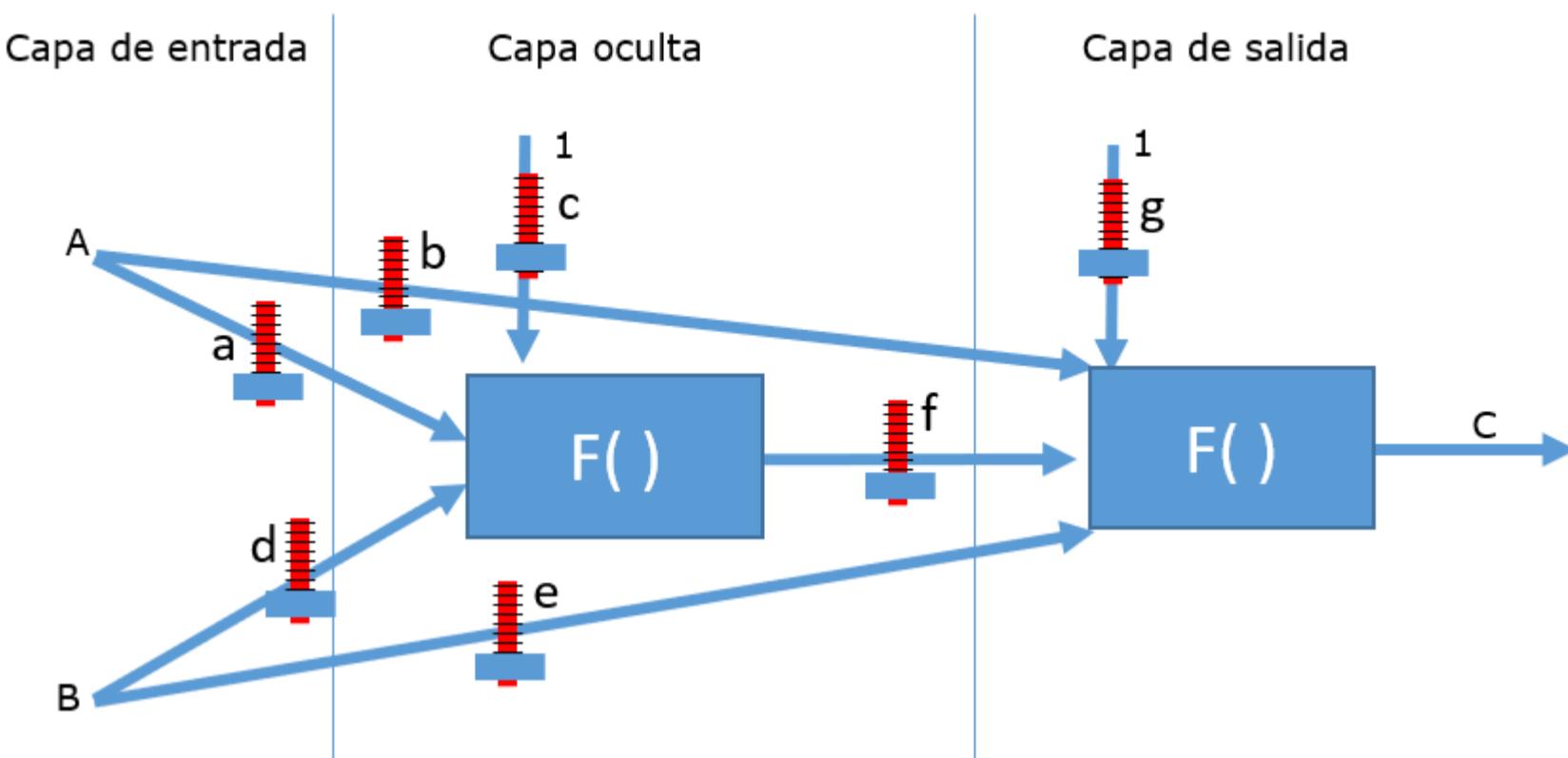


Ilustración 326: Red neuronal con varias conexiones distintas

En la figura se aprecian 7 pesos: a,b,c,d,e,f,g. ¿Cómo obtener un mínimo? En ese caso se utilizan derivadas parciales, es decir, se deriva por 'a' dejando el resto como constantes, luego por 'b' dejando el resto constantes y así sucesivamente. Esos mínimos servirán para ir ajustando los pesos.

Perceptrón Multicapa

Es un tipo de red neuronal en donde hay varias capas:

1. Capa de entrada
2. Capas ocultas
3. Capa de salida

En la siguiente figura se muestra un ejemplo de perceptrón multicapa, los círculos representan las neuronas. Tiene dos capas ocultas. Las capas ocultas donde cada una tiene 3 neuronas y la capa de salida con 2 neuronas.

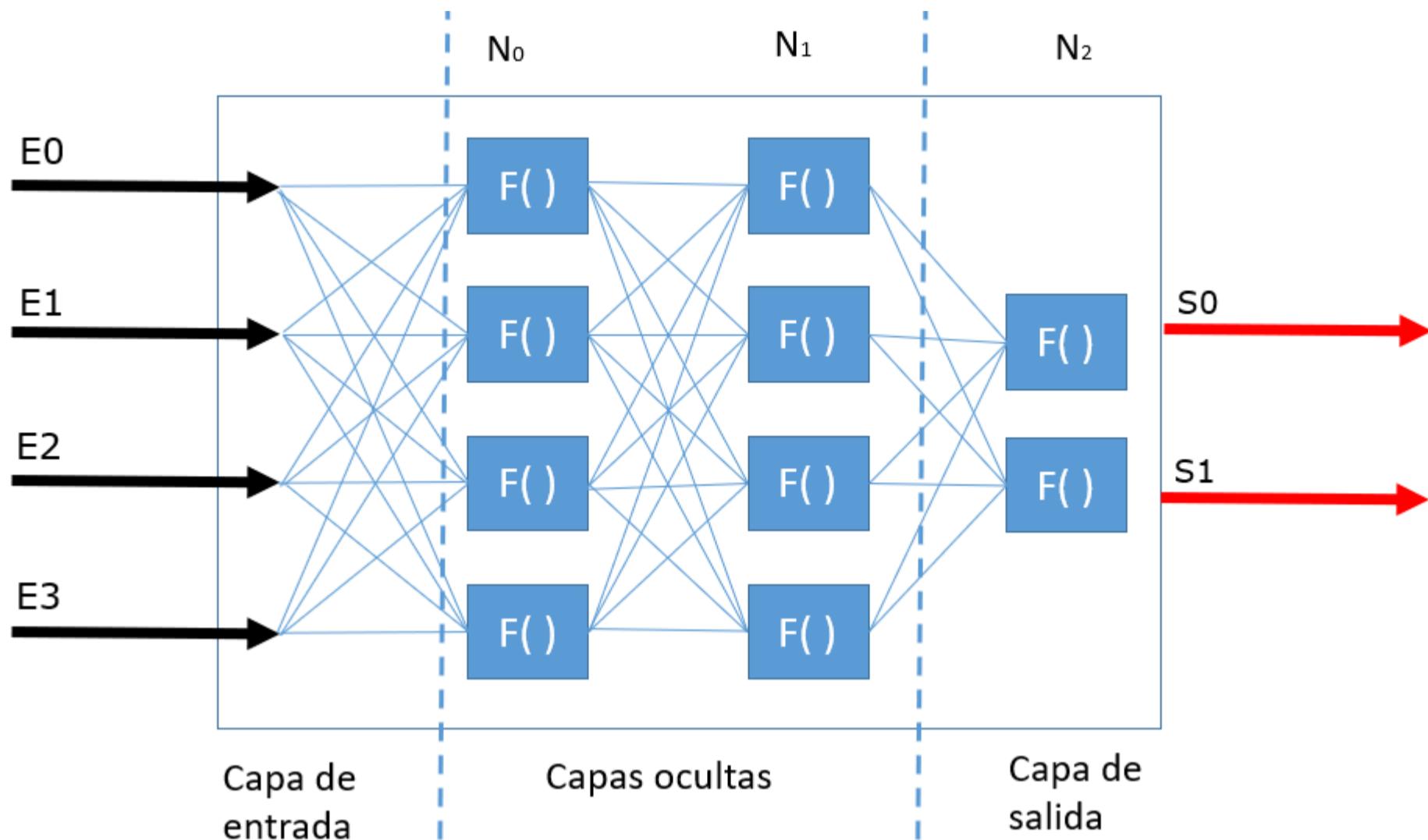


Ilustración 327: Perceptrón multicapa

Las capas se denotarán con la letra 'N', luego

$N_0=4$ (capa 0, que es oculta, tiene 4 neuronas)

$N_1=4$ (capa 1, que es oculta, tiene 4 neuronas)

$N_2=2$ (capa 2, que es la de salida, tiene 2 neuronas)

La capa de entrada no hace ningún proceso, sólo recibe los datos de entrada.

En el perceptrón multicapa, las neuronas de la capa 0 se conectan con las neuronas de la capa 1, las neuronas de la capa 1 con las neuronas de la capa 2. No está permitido conectar neuronas de la capa 0 con las neuronas de la capa 2 por ejemplo, ese salto podrá suceder en otro tipo de redes neuronales, pero no en el perceptrón multicapa.

Las neuronas

De nuevo se muestra un esquema de cómo es una neurona con dos entradas externas y su salida.

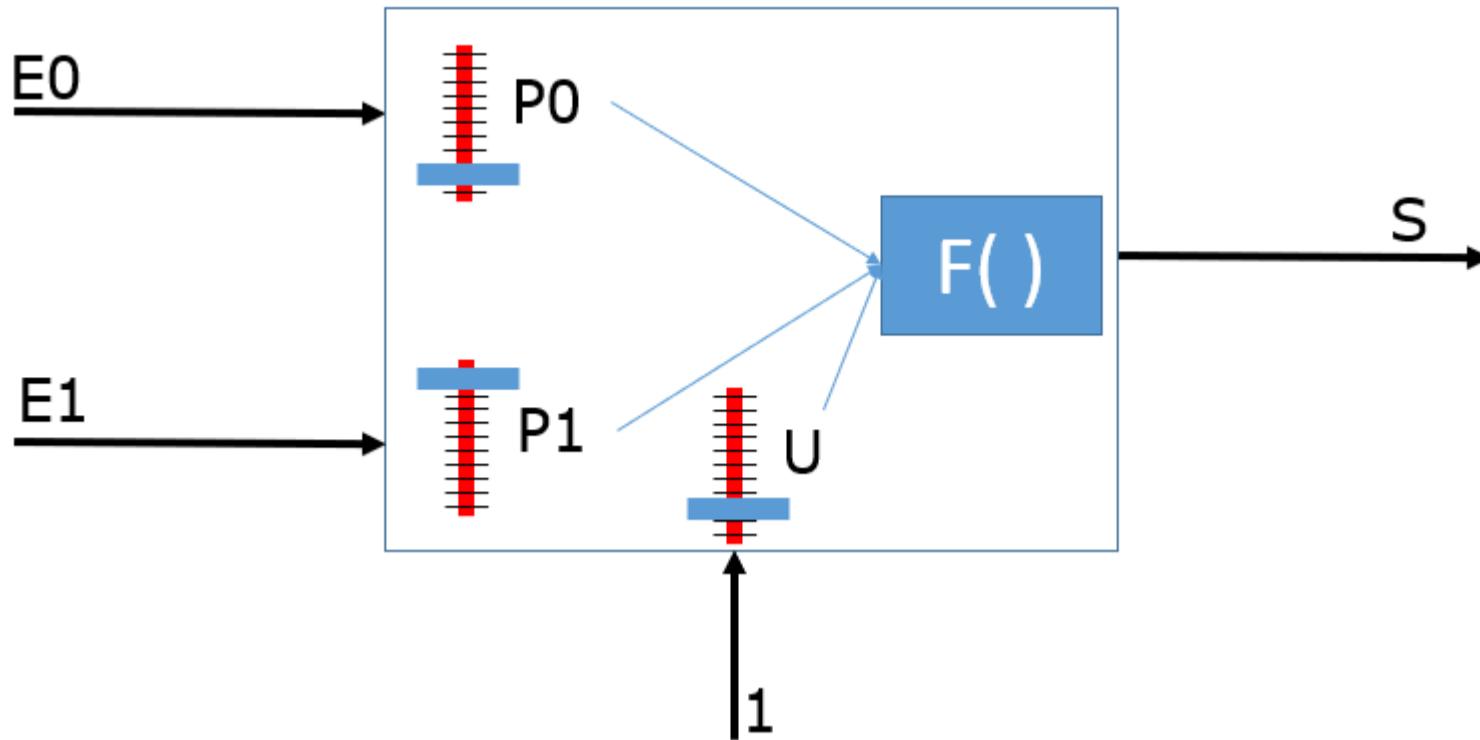


Ilustración 328: Esquema de una neurona

Mostrado como una clase en C#, esta sería su implementación:

K/004.cs

```
namespace Ejemplo {
    class Neurona {
        public double CalculaSalida(double E0, double E1) {
            double S = 0;

            //Se hace una operación aquí

            return S;
        }
    }

    class Program {
        static void Main() {
            Neurona objA = new Neurona();
            Neurona objB = new Neurona();
        }
    }
}
```

En cada entrada hay un peso P_0 y P_1 . Para la entrada interna, que siempre es 1, el peso se llama U

K/005.cs

```
namespace Ejemplo {
    class Neurona {
        //Pesos para cada entrada  $P_0$  y  $P_1$ ; y el peso de la entrada interna  $U$ 
        private double P0;
        private double P1;
        private double U;

        public double CalculaSalida(double E0, double E1) {
            double S = 0;

            //Se hace una operación aquí

            return S;
        }
    }

    class Program {
        static void Main() {
            Neurona objA = new Neurona();
            Neurona objB = new Neurona();
        }
    }
}
```

Los pesos se inicializan con un valor al azar y un buen sitio es hacerlo en el constructor. En el ejemplo se hace uso de la clase Random y luego NextDouble() que retorna un número real al azar entre 0 y 1.

K/006.cs

```
namespace Ejemplo {
    class Neurona {
        //Pesos para cada entrada P0 y P1; y el peso de la entrada interna U
        private double P0;
        private double P1;
        private double U;

        public Neurona(Random Azar) { //Constructor
            P0 = Azar.NextDouble();
            P1 = Azar.NextDouble();
            U = Azar.NextDouble();
        }

        public double CalculaSalida(double E0, double E1) {
            double S = 0;

            //Se hace una operación aquí

            return S;
        }
    }

    class Program {
        static void Main() {
            Random Azar = new();
            Neurona objA = new Neurona(Azar);
            Neurona objB = new Neurona(Azar);
        }
    }
}
```

Pesos y como nombrarlos

En el gráfico se dibujan algunos pesos y como se podrá dilucidar, el número de estos pesos crece rápidamente a medida que se agregan capas y neuronas.

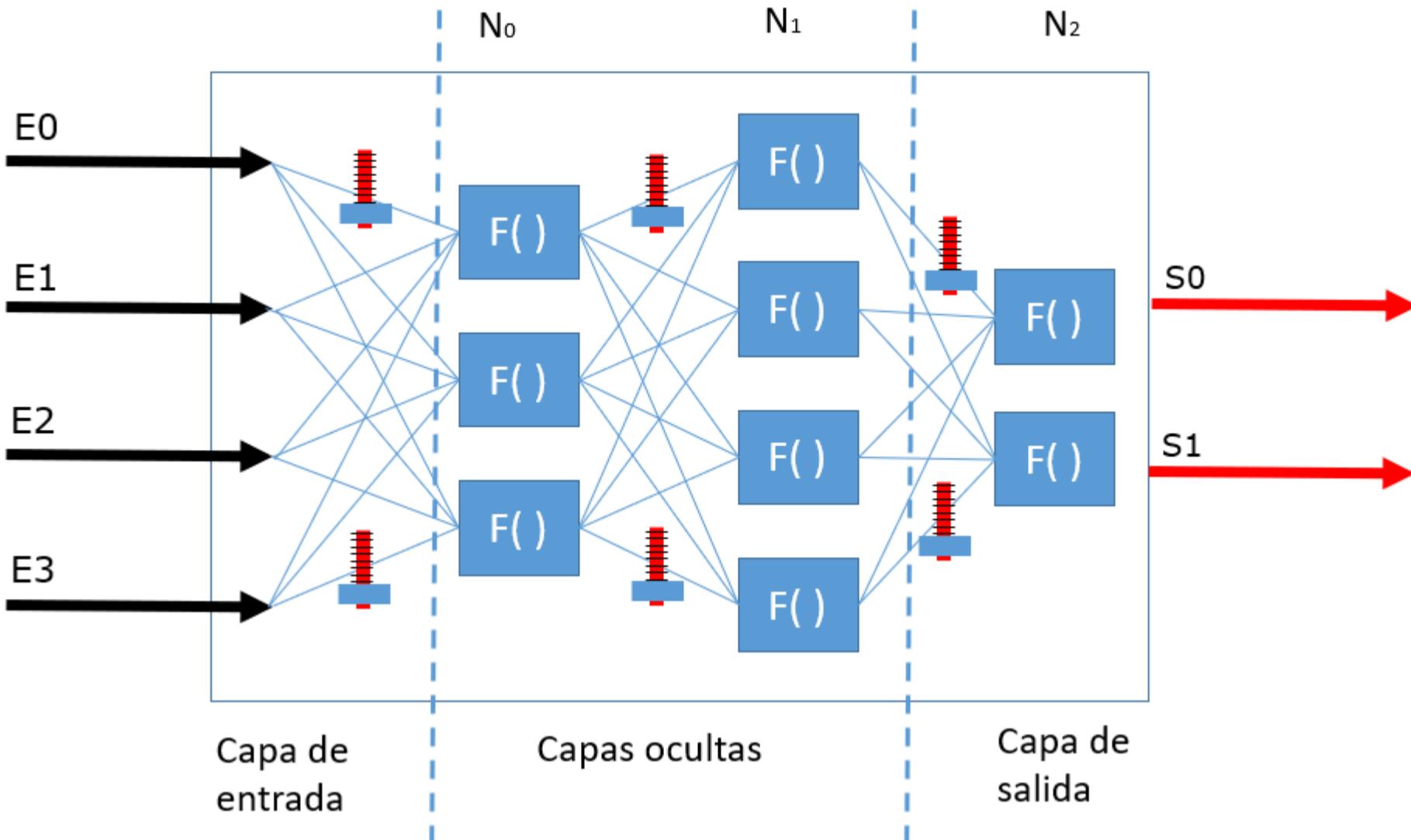


Ilustración 329: Esquema de un perceptrón multicapa

Un ejemplo: Capa 0 tiene 3 neuronas, capa 1 tiene 4 neuronas, luego el total de conexiones entre Capa 0 y Capa 1 son $3 \times 4 = 12$ conexiones, luego son 12 pesos. Si se usaran más neuronas por capa, habría tantos pesos que nombrarlos con una sola letra no sería conveniente. Por tal motivo, hay otra forma de nombrarlos y es el siguiente:

$w_{\text{neurona inicial}, \text{neurona final}}$ (*capa a donde llega la conexión*)

W es la letra inicial de la palabra peso en inglés: Weight.

(Capa de donde sale la conexión) Las capas se enumeran desde 0 que sería en este caso la primera capa oculta. ¿Cómo nombrar los pesos iniciales? Esos pesos de la capa de entrada tendrán como "capa de donde sale la conexión" la letra E.

Neurona inicial, de donde parte la conexión. Se enumeran desde 0 de arriba abajo en la capa.

Neurona final, a donde llega la conexión. Se enumeran desde 0 de arriba abajo en la capa.

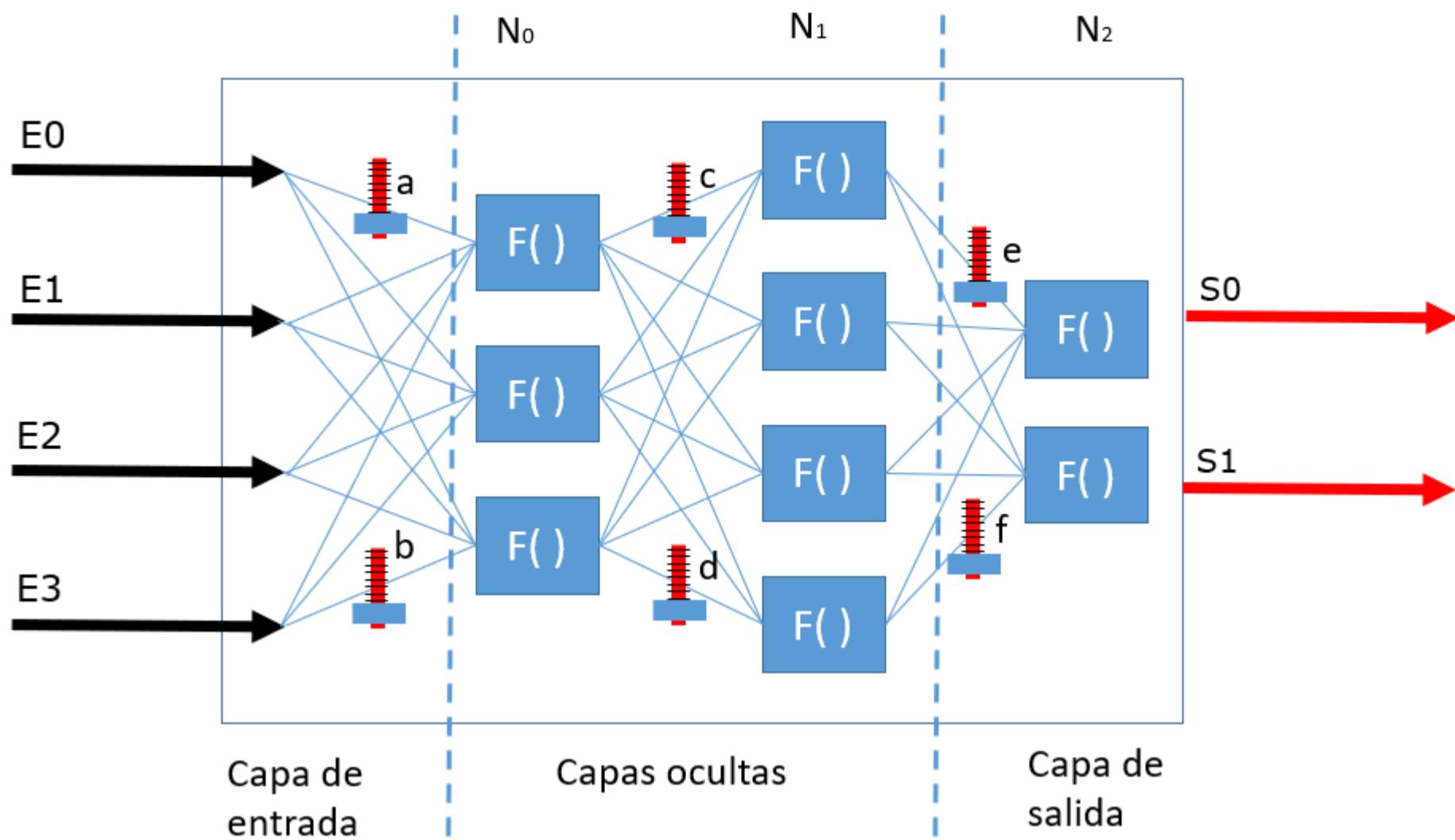


Ilustración 330: Nombrando los pesos con una letra

Para nombrar el peso mostrado con la letra 'a' sería entonces:

$w_{\text{neurona inicial}, \text{neurona final}}^{(\text{capa a donde llega la conexión})}$

$$w_{0,0}^{(0)}$$

En esta tabla se muestra como se nombrarían los pesos que se han puesto en la gráfica:

Peso	Se nombra
a	$w_{0,0}^{(0)}$
b	$w_{3,2}^{(0)}$
c	$w_{0,0}^{(1)}$
d	$w_{2,3}^{(1)}$
e	$w_{0,0}^{(2)}$
f	$w_{3,1}^{(2)}$

La función de activación de la neurona

Anteriormente se había mostrado que la función de activación era esta:

```
Función F(valor)
Inicio
    Si valor > 0.5 entonces
        retorno 1
    de lo contrario
        retorno 0
    fin si
Fin
```

En otros problemas, por lo general, esa función es la sigmoide que tiene la siguiente ecuación:

$$y = \frac{1}{1 + e^{-x}}$$

Esta sería una tabla de valores generados con esa función

x	y
-10	4.5E-05
-9	0.00012
-8	0.00034
-7	0.00091
-6	0.00247
-5	0.00669
-4	0.01799
-3	0.04743
-2	0.1192
-1	0.26894
0	0.5
1	0.73106
2	0.8808
3	0.95257
4	0.98201
5	0.99331
6	0.99753
7	0.99909
8	0.99966
9	0.99988
10	0.99995

Y esta sería su gráfica

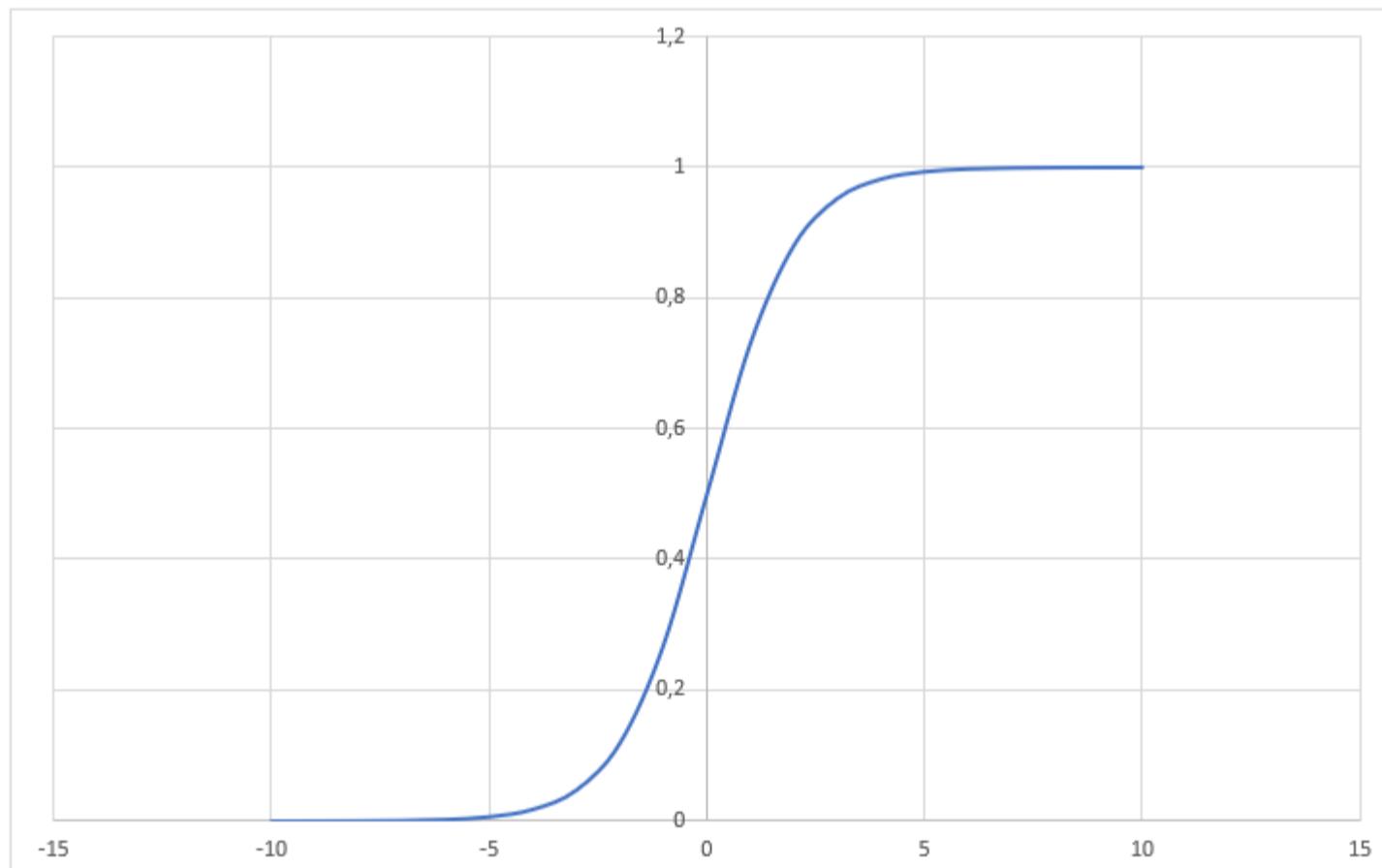


Ilustración 331: Gráfico de una función sigmoide

Al moverse a la izquierda el valor que toma es 0 y al moverse a la derecha toma el valor de 1. Hay una transición pronunciada de 0 a 1 en el rango [-5 y 5].

¿Qué tiene de especial esta función sigmoide? Su derivada.

Ecuación original:

$$y = \frac{1}{1 + e^{-x}}$$

Derivada no negativa de esa ecuación:

$$\partial y = \frac{e^{-x}}{(1 + e^{-x})^2}$$

Que equivale a esto:

$$\partial y = y * (1 - y)$$

Demostración de la equivalencia:

$$\begin{aligned}\partial y &= \frac{1}{1 + e^{-x}} * \left[1 - \frac{1}{1 + e^{-x}} \right] \\ \partial y &= \frac{1}{1 + e^{-x}} - \frac{1}{1 + e^{-x}} * \frac{1}{1 + e^{-x}} \\ \partial y &= \frac{1}{1 + e^{-x}} - \frac{1}{(1 + e^{-x})^2} \\ \partial y &= \frac{1 + e^{-x} - 1}{(1 + e^{-x})^2} \\ \partial y &= \frac{e^{-x}}{(1 + e^{-x})^2}\end{aligned}$$

Nota: Si hace uso de WolframAlpha y deriva la sigmoidea, sucede esto:

derive $y=1/(1+e^{-x})$

Extended Keyboard Upload

Input interpretation:

differentiate	$y = \frac{1}{1 + e^{-x}}$	with respect to	x
---------------	----------------------------	-----------------	-----

Result:

$$y'(x) = \frac{e^x}{(e^x + 1)^2}$$

Ilustración 332: Derivada de la función sigmoide

A primera vista parece diferente a la derivada mostrada anteriormente, pero si se compara

$$e^x/(e^x+1)^2 = e^{-x}/(1+e^{-x})^2$$

Σ Extended Keyboard

Upload

Input:

$$\frac{e^x}{(e^x + 1)^2} = \frac{e^{-x}}{(1 + e^{-x})^2}$$

Alternate form:

True

Ilustración 333: Comparativa de derivadas

```

namespace Ejemplo {
    class Neurona {
        //Pesos para cada entrada P0 y P1; y el peso de la entrada interna U
        private double P0;
        private double P1;
        private double U;

        public Neurona(Random Azar) { //Constructor
            P0 = Azar.NextDouble();
            P1 = Azar.NextDouble();
            U = Azar.NextDouble();
        }

        //Calcula la salida de la neurona con las dos entradas E0 y E1
        public double CalculaSalida(double E0, double E1) {
            double Valor = E0 * P0 + E1 * P1 + 1 * U;
            return 1 / (1 + Math.Exp(-Valor));
        }
    }

    class Program {
        static void Main() {
            Random Azar = new();
            Neurona objA = new Neurona(Azar);
            Neurona objB = new Neurona(Azar);
        }
    }
}

```

El método calculaSalida implementa el procesamiento de la neurona. Tiene como parámetros las entradas, en este caso, dos entradas E0 y E1. En el interior cada entrada se multiplica con su peso respectivo, se suman, incluyendo la entrada interna (umbral). Una vez con ese valor, se calcula la salida con la sigmoide.

Introducción al algoritmo “Backpropagation” (backward propagation of errors)

En el siguiente ejemplo se observa una conexión entre tres neuronas

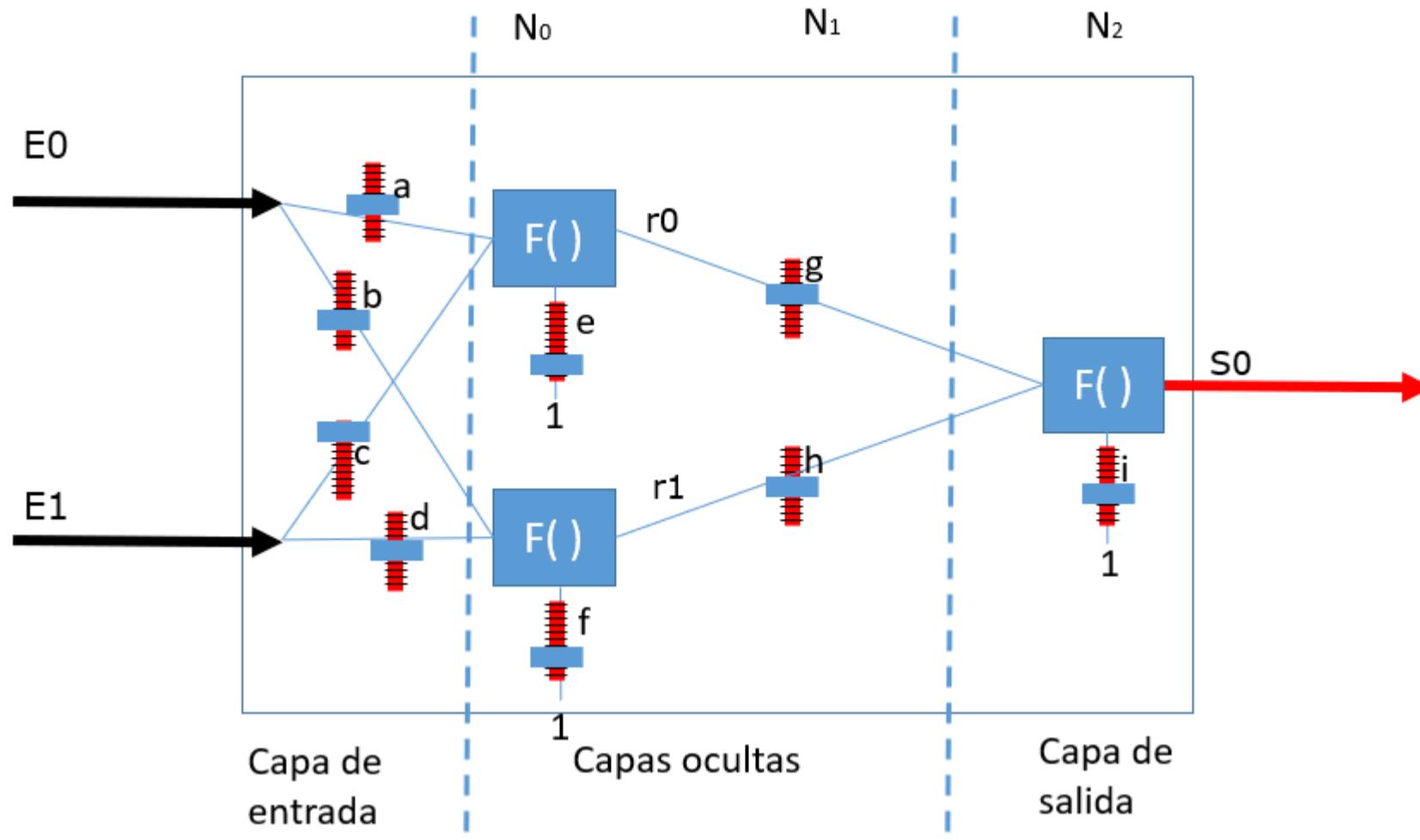


Ilustración 334: Esquema de un perceptrón multicapa

E_0 y E_1 son las entradas externas. Se observa que E_0 entra con el peso ‘ a ’ en la neurona de arriba y con peso ‘ b ’ en la neurona de abajo. Sucede lo mismo con la entrada E_1 .

Lo interesante viene después, porque el resultado de la neurona de arriba ‘ r_0 ’ y el resultado de la neurona de abajo ‘ r_1 ’ se convierten en entradas para la neurona de la derecha y esas entradas a su vez tienen sus propios pesos. Al final el sistema genera una salida S_0 que es la respuesta final de la red neuronal.

Obsérvese que cada neurona tiene la entrada 1 y su peso llamado umbral.

¿Qué importancia tiene eso? Que, si se quiere ajustar S_0 al resultado esperado, entonces se retrocede a las entradas de esa neurona de la derecha ajustando sus pesos respectivos y por supuesto, ese ajuste hace retroceder más aún hasta mirar los pesos de las neuronas de arriba y abajo. Eso se conocerá como el algoritmo “Backpropagation”.

Luego:

$$r_0 = F(E_0 * a + E_1 * c + 1 * e)$$

$$r_1 = F(E_0 * b + E_1 * d + 1 * f)$$

$$S_0 = F(r_0 * g + r_1 * h + 1 * i)$$

Se concluye entonces que:

$$S_0 = F(F(E_0 * a + E_1 * c + 1 * e) * g + F(E_0 * b + E_1 * d + 1 * f) * h + 1 * i)$$

Nombrando las entradas, pesos, umbrales, salidas y capas

Volviendo al perceptrón multicapa. Es momento de ponerle nombres a las diversas partes de la red neuronal:

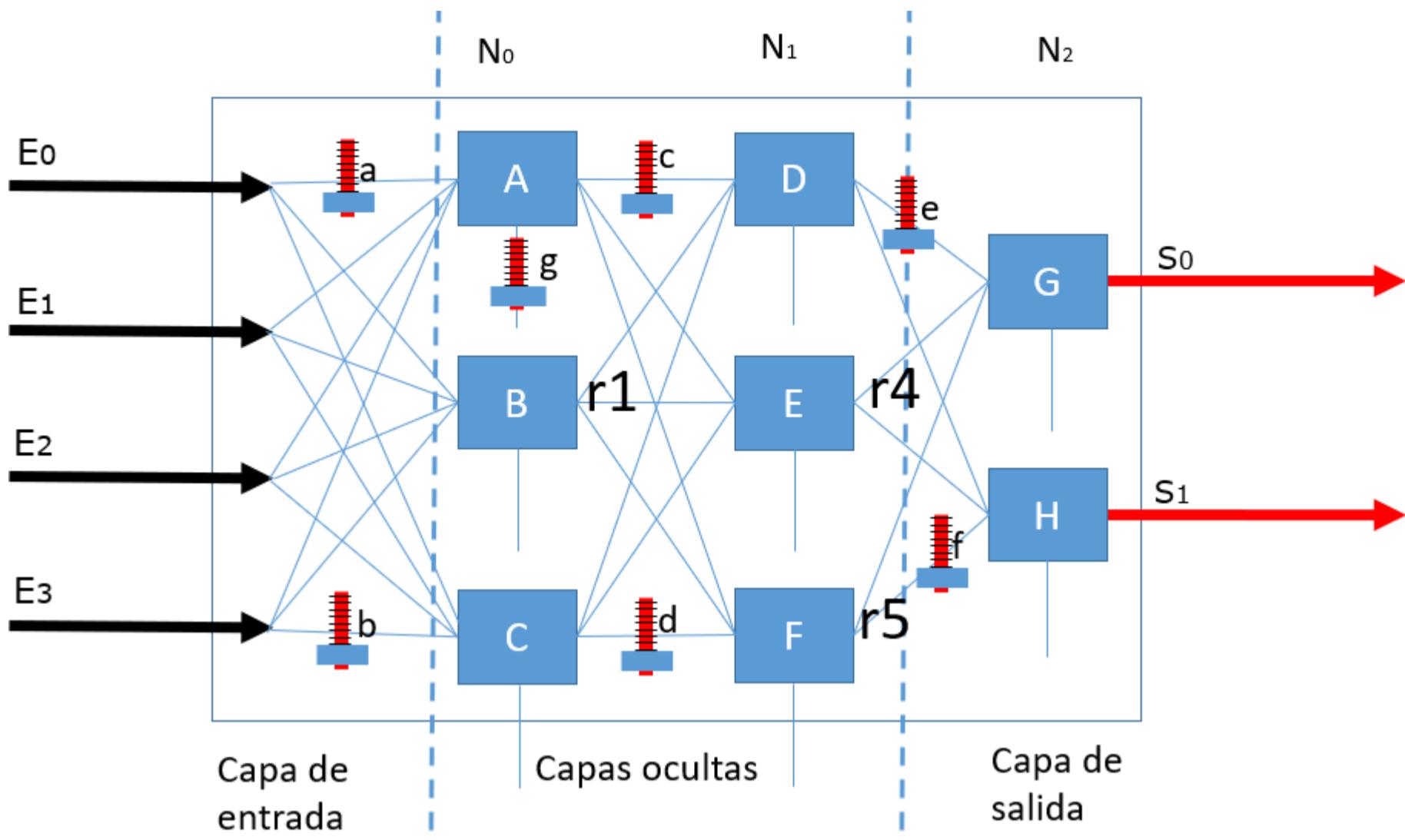


Ilustración 335: Partes de un perceptrón multicapa

Las entradas:

$$E_{\text{número de la entrada}}$$

Los pesos:

$$w_{\text{neurona inicial}, \text{neurona final}}^{\text{(capa a donde llega la conexión)}}$$

Los umbrales:

$$u_{\text{neurona que tiene esa entrada interna}}^{\text{(capa de la neurona que tiene esa entrada interna)}}$$

Las salidas internas de cada neurona:

$$a_{\text{neurona de esa salida}}^{\text{(capa de la neurona de esa salida)}}$$

Las capas:

$$n_{\text{número de la capa}}$$

Entonces, viendo el gráfico:

Como se nombró antes	Nueva nomenclatura
E_0	E_0
E_1	E_1
E_2	E_2
E_3	E_3
a	$w_{0,0}^{(0)}$
b	$w_{3,3}^{(0)}$
c	$w_{0,0}^{(1)}$

d	$w_{2,2}^{(1)}$
e	$w_{0,0}^{(2)}$
f	$w_{2,1}^{(2)}$
g	$u_0^{(0)}$
r1	$a_1^{(0)}$
r4	$a_1^{(1)}$
r5	$a_2^{(1)}$
S ₀	$a_0^{(2)}$
S ₁	$a_1^{(2)}$

Luego:

$$a_0^{(2)} = F \left(a_0^{(1)} * w_{0,0}^{(2)} + a_1^{(1)} * w_{1,0}^{(2)} + a_2^{(1)} * w_{2,0}^{(2)} + u_0^{(2)} \right)$$

$$a_1^{(2)} = F \left(a_0^{(1)} * w_{0,1}^{(2)} + a_1^{(1)} * w_{1,1}^{(2)} + a_2^{(1)} * w_{2,1}^{(2)} + u_1^{(2)} \right)$$

$$a_0^{(1)} = F \left(a_0^{(0)} * w_{0,0}^{(1)} + a_1^{(0)} * w_{1,0}^{(1)} + a_2^{(0)} * w_{2,0}^{(1)} + u_0^{(1)} \right)$$

$$a_1^{(1)} = F \left(a_0^{(0)} * w_{0,1}^{(1)} + a_1^{(0)} * w_{1,1}^{(1)} + a_2^{(0)} * w_{2,1}^{(1)} + u_1^{(1)} \right)$$

$$a_2^{(1)} = F \left(a_0^{(0)} * w_{0,2}^{(1)} + a_1^{(0)} * w_{1,2}^{(1)} + a_2^{(0)} * w_{2,2}^{(1)} + u_2^{(1)} \right)$$

$$a_0^{(0)} = F \left(E_0 * w_{0,0}^{(0)} + E_1 * w_{1,0}^{(0)} + E_2 * w_{2,0}^{(0)} + E_3 * w_{3,0}^{(0)} + u_0^{(0)} \right)$$

$$a_1^{(0)} = F \left(E_0 * w_{0,1}^{(0)} + E_1 * w_{1,1}^{(0)} + E_2 * w_{2,1}^{(0)} + E_3 * w_{3,1}^{(0)} + u_1^{(0)} \right)$$

$$a_2^{(0)} = F \left(E_0 * w_{0,2}^{(0)} + E_1 * w_{1,2}^{(0)} + E_2 * w_{2,2}^{(0)} + E_3 * w_{3,2}^{(0)} + u_2^{(0)} \right)$$

¿Qué hay de E_0 , E_1 , E_2 y E_3 ? Podrían tomarse como salidas de las neuronas de la capa E (Entrada). Cabe recordar que en esa capa no hay procesamiento. Luego:

Como se nombró antes	Nueva nomenclatura
E_0	$a_0^{(E)}$
E_1	$a_1^{(E)}$
E_2	$a_2^{(E)}$
E_3	$a_3^{(E)}$

Luego las tres últimas ecuaciones quedan así:

$$a_0^{(0)} = F \left(a_0^{(E)} * w_{0,0}^{(0)} + a_1^{(E)} * w_{1,0}^{(0)} + a_2^{(E)} * w_{2,0}^{(0)} + a_3^{(E)} * w_{3,0}^{(0)} + u_0^{(0)} \right)$$

$$a_1^{(0)} = F \left(a_0^{(E)} * w_{0,1}^{(0)} + a_1^{(E)} * w_{1,1}^{(0)} + a_2^{(E)} * w_{2,1}^{(0)} + a_2^{(E)} * w_{3,1}^{(0)} + u_1^{(0)} \right)$$

$$a_2^{(0)} = F \left(a_0^{(E)} * w_{0,2}^{(0)} + a_1^{(E)} * w_{1,2}^{(0)} + a_2^{(E)} * w_{2,2}^{(0)} + a_2^{(E)} * w_{3,2}^{(0)} + u_2^{(0)} \right)$$

¡OJO! Las capas tendrían este orden E, 0, 1, 2

Y generalizando se puede decir que:

$$a_i^{(k)} = F \left(a_j^{(k-1)} * w_{j,i}^{(k)} + a_{j+1}^{(k-1)} * w_{j+1,i}^{(k)} + a_{j+2}^{(k-1)} * w_{j+2,i}^{(k)} + a_{j+3}^{(k-1)} * w_{j+3,i}^{(k)} + 1 * u_i^{(k)} \right)$$

Es decir que si $k-1 < 0$ entonces se está refiriendo a la capa E la de entrada.

Se puede simplificar más:

$$a_i^{(k)} = f \left(u_i^{(k)} + \sum_{j=0}^{n_k-1} a_j^{(k-1)} * w_{j,i}^{(k)} \right)$$

Donde k inicia en 0 y termina en el número de la última capa.

Regla de la cadena

Para continuar con el algoritmo de "Backpropagation", cabe recordar esta regla matemática llamada regla de la cadena.

$$\partial\{F[g(x)]\} = \partial F[g(x)] * \partial g(x)$$

Un ejemplo, hay dos funciones:

$$g(x) = 3 * x^2$$

$$F[p] = 7 - p^3$$

Luego:

$$F[g(x)] = 7 - (3 * x^2)^3 = 7 - 27 * x^6$$

Reemplazando "p" con lo que tiene $g(x)$

Derivando

$$\partial\{F[g(x)]\} = -162 * x^5$$

Usando la regla de la cadena

$$\partial\{F[g(x)]\} = \partial F[g(x)] * \partial g(x) = \partial(7 - p^3) * \partial(3 * x^2) =$$

$$(0 - 3 * p^2) * (6 * x) = (0 - 3 * (3 * x^2)^2) * (6 * x) =$$

$$(0 - 3 * (9 * x^4)) * (6 * x) = -162 * x^5$$

La regla de la cadena funciona.

Derivadas parciales

Dada una ecuación que tenga dos o más variables independientes, es posible derivar por una variable considerando las demás constantes, eso es conocido como derivada parcial.

Ejemplo de una ecuación con tres variables independientes:

$$q = a^2 + b^3 + c^4$$

Su derivada parcial con respecto a la variable **a** sería

$$\frac{\partial q}{\partial a}(a, b, c) = 2 * a + 0 + 0$$

$$\frac{\partial q}{\partial a}(a, b, c) = 2 * a$$

Su derivada parcial con respecto a la variable **b** sería

$$\frac{\partial q}{\partial b}(a, b, c) = 0 + 3 * b^2 + 0$$

$$\frac{\partial q}{\partial b}(a, b, c) = 3 * b^2$$

Su derivada parcial con respecto a la variable **c** sería

$$\frac{\partial q}{\partial c}(a, b, c) = 0 + 0 + 4 * c^3$$

$$\frac{\partial q}{\partial c}(a, b, c) = 4 * c^3$$



d/da a^2+b^3+c^4

✉ Extended Keyboard

⬆ Upload

⋮ Examp

Derivative:

St

$$\frac{\partial}{\partial a}(a^2 + b^3 + c^4) = 2a$$

Ilustración 336: Derivada parcial

Las derivadas en el algoritmo de propagación hacia atrás

Observese el siguiente gráfico muy sencillo de un perceptrón multicapa. Hay que recordar que la capa de entrada no hace proceso.

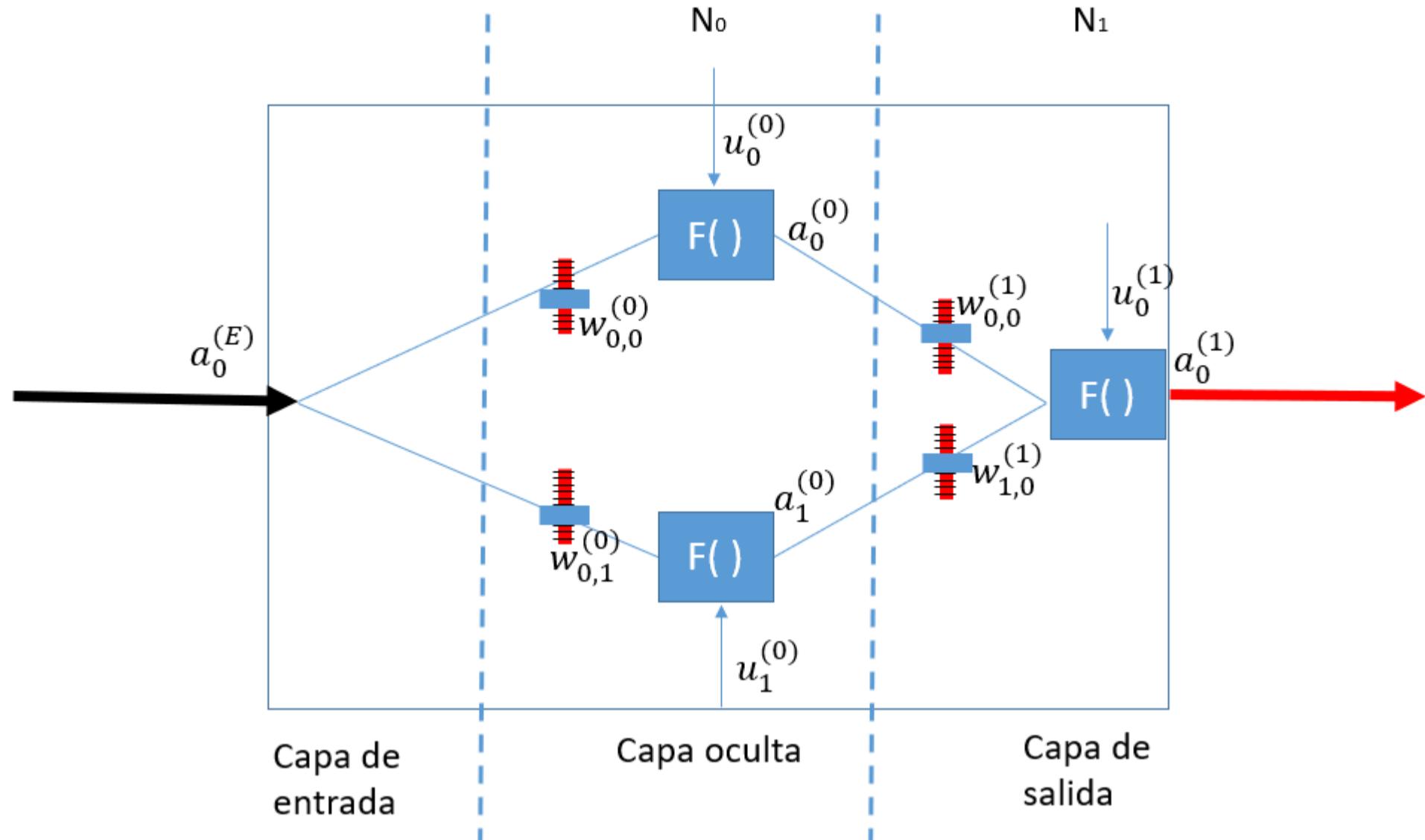


Ilustración 337: Nombramiento de pesos, umbrales y salidas

$$a_0^{(1)} = F(a_0^{(0)} * w_{0,0}^{(1)} + a_1^{(0)} * w_{1,0}^{(1)} + 1 * u_0^{(1)})$$

$$a_0^{(0)} = F(a_0^{(E)} * w_{0,0}^{(0)} + 1 * u_0^{(0)})$$

$$a_1^{(0)} = F(a_0^{(E)} * w_{0,1}^{(0)} + 1 * u_1^{(0)})$$

Luego reemplazando

$$a_0^{(1)} = F(F(a_0^{(E)} * w_{0,0}^{(0)} + 1 * u_0^{(0)}) * w_{0,0}^{(1)} + F(a_0^{(E)} * w_{0,1}^{(0)} + 1 * u_1^{(0)}) * w_{1,0}^{(1)} + 1 * u_0^{(1)})$$

Simplificando

$$a_0^{(1)} = F(F(a_0^{(E)} * w_{0,0}^{(0)} + u_0^{(0)}) * w_{0,0}^{(1)} + F(a_0^{(E)} * w_{0,1}^{(0)} + u_1^{(0)}) * w_{1,0}^{(1)} + u_0^{(1)})$$

Como la función $F[\cdot]$ es una sigmoidea y la derivada de las sigmoideas es así:

$$\partial y = y * (1 - y)$$

Luego:

$$\partial F[r] = F[r] * (1 - F[r])$$

¿Qué sucedería si r es a su vez una función sigmoidea?

$$r = g(k)$$

Tener en cuenta que $g(\cdot)$ es una sigmoidea. Y además de eso, k es una función polinómica.

Recordando la regla de la cadena, esto sucedería:

$$\partial\{F[g(k)]\} = \partial F[g(k)] * \partial g(k)$$

Y como $F(\cdot)$ es sigmoidea, entonces al derivar

$$\partial\{F[g(k)]\} = F(g(k)) * (1 - F(g(k))) * \partial g(k)$$

$g(k)$ es una función sigmoidea y k es una función polinómica, luego la derivada de $g(k)$ aplicando la regla de la cadena sería:

$$\partial g(k) = \partial g(k) * \partial k = g(k) * (1 - g(k)) * \partial k$$

La derivada queda así

$$\partial\{F[g(k)]\} = F(g(k)) * (1 - F(g(k))) * g(k) * (1 - g(k)) * \partial k$$

Simplificando

$$\partial\{F[g(k)]\} = F(r) * (1 - F(r)) * g(k) * (1 - g(k)) * \partial k$$

Esta es la ecuación (la salida total de esa red neuronal) que se va a derivar parcialmente con respecto a un peso en particular

$$a_0^{(1)} = F(F(a_0^{(E)} * w_{0,0}^{(0)} + u_0^{(0)}) * w_{0,0}^{(1)} + F(a_0^{(E)} * w_{0,1}^{(0)} + u_1^{(0)}) * w_{1,0}^{(1)} + u_0^{(1)})$$

En el ejemplo, se deriva con respecto a $w_{0,0}^{(0)}$ (una derivada parcial). En rojo se pone que ecuación interna es derivable con respecto a $w_{0,0}^{(0)}$

$$\frac{\partial a_0^{(1)}}{\partial w_{0,0}^{(0)}} = \partial F \left(\partial \left[F \left(a_0^{(E)} * w_{0,0}^{(0)} + u_0^{(0)} \right) * w_{0,0}^{(0)} \right] + 0 + 0 \right)$$

Para derivar entonces se deriva la F externa (que está en negro y es $F(r)$), luego la F interna (que está en rojo y es $g(k)$ y que la multiplica la constante $w_{0,0}^{(1)}$) y por último el polinomio (que es k) que está en verde porque allí está $w_{0,0}^{(0)}$. Hay tres derivaciones.

Entonces, se sabe que:

$$F(r) = a_0^{(1)}$$

$$g(k) = a_0^{(0)}$$

$$k = a_0^{(E)} * w_{0,0}^{(0)} + u_0^{(0)}$$

$$\partial k = a_0^{(E)}$$

$$\frac{\partial a_0^{(1)}}{\partial w_{0,0}^{(E)}} = \partial \{F[g(k)]\}$$

$$\partial \{F[g(k)]\} = F(r) * (1 - F(r)) * g(k) * (1 - g(k)) * \partial k$$

Entonces:

$$\frac{\partial a_0^{(1)}}{\partial w_{0,0}^{(0)}} = a_0^{(1)} * (1 - a_0^{(1)}) * a_0^{(0)} * (1 - a_0^{(0)}) * a_0^{(E)} * w_{0,0}^{(1)}$$

Esta es la ecuación (la salida total de esa red neuronal) que se va a derivar parcialmente con respecto a un peso en particular

$$a_0^{(1)} = F(F(a_0^{(E)} * w_{0,0}^{(0)} + u_0^{(0)}) * w_{0,0}^{(1)} + F(a_0^{(E)} * w_{0,1}^{(0)} + u_1^{(0)}) * w_{1,0}^{(1)} + u_0^{(1)})$$

En el ejemplo, se deriva con respecto a $w_{0,1}^{(0)}$ (una derivada parcial). En azul se pone que ecuación interna es derivable con respecto a $w_{0,1}^{(0)}$

$$\frac{\partial a_0^{(1)}}{\partial w_{0,0}^{(0)}} = \partial F \left(0 + \partial [F(a_0^{(E)} * w_{0,1}^{(0)} + u_1^{(0)}) * w_{1,0}^{(1)}] + 0 \right)$$

Para derivar entonces se deriva la F externa (que está en negro y es $F(r)$), luego la F interna (que está en azul y es $g(k)$ y que la multiplica la constante $w_{1,0}^{(1)}$) y por último el polinomio (que es k) que está en verde porque allí está $w_{0,0}^{(0)}$. Hay tres derivaciones.

Entonces, se sabe que:

$$F(r) = a_0^{(1)}$$

$$g(k) = a_1^{(0)}$$

$$k = a_0^{(E)} * w_{0,1}^{(0)} + u_1^{(0)}$$

$$\partial k = a_0^{(E)}$$

$$\frac{\partial a_0^{(1)}}{\partial w_{0,1}^{(0)}} = \partial \{F[g(k)]\}$$

$$\partial \{F[g(k)]\} = F(r) * (1 - F(r)) * g(k) * (1 - g(k)) * \partial k$$

Entonces:

$$\frac{\partial a_0^{(1)}}{\partial w_{0,1}^{(0)}} = a_0^{(1)} * (1 - a_0^{(1)}) * a_1^{(0)} * (1 - a_1^{(0)}) * a_0^{(E)} * w_{1,0}^{(1)}$$

Generalizando

$$\frac{\partial a_0^{(1)}}{\partial w_{0,j}^{(0)}} = a_0^{(1)} * (1 - a_0^{(1)}) * a_j^{(0)} * (1 - a_j^{(0)}) * a_0^{(E)} * w_{j,0}^{(1)}$$

Donde j puede ser 0 o 1. Esa sería la generalización para los pesos $w_{0,0}^{(0)}$ y $w_{0,1}^{(0)}$

Para el peso $w_{0,0}^{(1)}$ este sería el tratamiento:

$$a_0^{(1)} = F(F(a_0^{(E)} * w_{0,0}^{(0)} + u_0^{(0)}) * w_{0,0}^{(1)} + F(a_0^{(E)} * w_{0,1}^{(0)} + u_1^{(0)}) * w_{1,0}^{(1)} + u_0^{(1)})$$

En el ejemplo, se deriva con respecto a $w_{0,0}^{(1)}$ (una derivada parcial). Lo que está en rojo es lo que "sobrevive" de esa derivada parcial (se comporta como una constante porque se deriva parcialmente por $w_{0,0}^{(1)}$), quedando así:

$$\frac{\partial a_0^{(1)}}{\partial w_{0,0}^{(1)}} = \partial F \left(F(a_0^{(E)} * w_{0,0}^{(0)} + u_0^{(0)}) + 0 + 0 \right)$$

Entonces

$$\frac{\partial a_0^{(1)}}{\partial w_{0,0}^{(1)}} = \underbrace{a_0^{(1)} * (1 - a_0^{(1)})}_{\text{Se comporta como una}} * F(a_0^{(E)} * w_{0,0}^{(0)} + u_0^{(0)})$$

Y como

$$a_0^{(0)} = F(a_0^{(E)} * w_{0,0}^{(0)} + u_0^{(0)})$$

Entonces

$$\frac{\partial a_0^{(1)}}{\partial w_{0,0}^{(1)}} = a_0^{(1)} * (1 - a_0^{(1)}) * a_0^{(0)}$$

Para el peso $w_{1,0}^{(1)}$ este sería el tratamiento:

$$a_0^{(1)} = F(F(a_0^{(E)} * w_{0,0}^{(0)} + u_0^{(0)}) * w_{0,0}^{(1)} + F(a_0^{(E)} * w_{0,1}^{(0)} + u_1^{(0)}) * w_{1,0}^{(1)} + u_0^{(1)})$$

En el ejemplo, se deriva con respecto a $w_{1,0}^{(1)}$ (una derivada parcial). Lo que está en azul es lo que "sobrevive" de esa derivada parcial (se comporta como una constante porque se deriva parcialmente por $w_{1,0}^{(1)}$), quedando así:

$$\frac{\partial a_0^{(1)}}{\partial w_{1,0}^{(1)}} = \partial F \left(0 + F(a_0^{(E)} * w_{0,1}^{(0)} + u_1^{(0)}) + 0 \right)$$

Entonces

$$\frac{\partial a_0^{(1)}}{\partial w_{1,0}^{(1)}} = a_0^{(1)} * (1 - a_0^{(1)}) * F(a_0^{(E)} * w_{0,1}^{(0)} + u_1^{(0)})$$

Se comporta como una

Y como

$$a_1^{(0)} = F(a_0^{(E)} * w_{0,1}^{(0)} + u_1^{(0)})$$

Entonces

$$\frac{\partial a_0^{(1)}}{\partial w_{1,0}^{(1)}} = a_0^{(1)} * (1 - a_0^{(1)}) * a_1^{(0)}$$

Generalizando

$$\frac{\partial a_0^{(1)}}{\partial w_{j,0}^{(1)}} = a_0^{(1)} * (1 - a_0^{(1)}) * a_j^{(0)}$$

Donde $j=0$ o 1

Para el umbral $u_0^{(0)}$ este sería el tratamiento:

$$a_0^{(1)} = F(a_0^{(E)} * w_{0,0}^{(0)} + u_0^{(0)}) * w_{0,0}^{(1)} + F(a_0^{(E)} * w_{0,1}^{(0)} + u_1^{(0)}) * w_{1,0}^{(1)} + u_0^{(1)}$$

En el ejemplo, se deriva con respecto a $u_0^{(0)}$ (una derivada parcial). En rojo se pone que ecuación interna es derivable con respecto a $u_0^{(0)}$

$$\frac{\partial a_0^{(1)}}{\partial u_0^{(0)}} = \partial F \left(\partial F(a_0^{(E)} * w_{0,0}^{(0)} + u_0^{(0)}) * w_{0,0}^{(1)} + 0 + 0 \right)$$

Para derivar entonces se deriva la F externa (que está en negro y es $F(r)$), luego la F interna (que está en rojo y es $g(k)$ y que la multiplica la constante $w_{0,0}^{(1)}$) y por último el polinomio (que es k) que está en verde porque allí está $u_0^{(0)}$. Hay tres derivaciones.

Entonces, se sabe que:

$$F(r) = a_0^{(1)}$$

$$g(k) = a_0^{(0)}$$

$$k = a_0^{(E)} * w_{0,0}^{(0)} + u_0^{(0)}$$

$$\partial k = 1$$

$$\frac{\partial a_0^{(1)}}{\partial u_0^{(0)}} = \partial \{F[g(k)]\}$$

$$\partial \{F[g(k)]\} = F(r) * (1 - F(r)) * g(k) * (1 - g(k)) * \partial k$$

Entonces:

$$\frac{\partial a_0^{(1)}}{\partial u_0^{(0)}} = a_0^{(1)} * (1 - a_0^{(1)}) * a_0^{(0)} * (1 - a_0^{(0)}) * 1 * w_{0,0}^{(1)}$$

Para el umbral $u_1^{(0)}$ este sería el tratamiento:

$$a_0^{(1)} = F(a_0^{(E)} * w_{0,0}^{(0)} + u_0^{(0)}) * w_{0,0}^{(1)} + F(a_0^{(E)} * w_{0,1}^{(0)} + u_1^{(0)}) * w_{1,0}^{(1)} + u_0^{(1)}$$

En el ejemplo, se deriva con respecto a $u_1^{(0)}$ (una derivada parcial). En azul se pone que ecuación interna es derivable con respecto a $u_1^{(0)}$

$$\frac{\partial a_0^{(1)}}{\partial u_1^{(0)}} = \partial F \left(0 + F(a_0^{(E)} * w_{0,1}^{(0)} + u_1^{(0)}) * w_{1,0}^{(1)} + 0 \right)$$

Para derivar entonces se deriva la F externa (que está en negro y es $F(r)$), luego la F interna (que está en azul y es $g(k)$ y que la multiplica la constante $w_{1,0}^{(1)}$) y por último el polinomio (que es k) que está en verde porque allí está $u_1^{(0)}$. Hay tres derivaciones.

Entonces, se sabe que:

$$F(r) = a_0^{(1)}$$

$$g(k) = a_1^{(0)}$$

$$k = a_0^{(E)} * w_{0,1}^{(0)} + u_1^{(0)}$$

$$\partial k = 1$$

$$\frac{\partial a_0^{(1)}}{\partial u_1^{(0)}} = \partial \{F[g(k)]\}$$

$$\partial \{F[g(k)]\} = F(r) * (1 - F(r)) * g(k) * (1 - g(k)) * \partial k$$

Entonces:

$$\frac{\partial a_0^{(1)}}{\partial u_1^{(0)}} = a_0^{(1)} * (1 - a_0^{(1)}) * a_1^{(0)} * (1 - a_1^{(0)}) * 1 * w_{1,0}^{(1)}$$

Con un ejemplo más complejo en el que la capa oculta tiene dos capas de neuronas y cada capa tiene dos neuronas.

Luego $N_0 = 2$, $N_1 = 2$, $N_2 = 1$

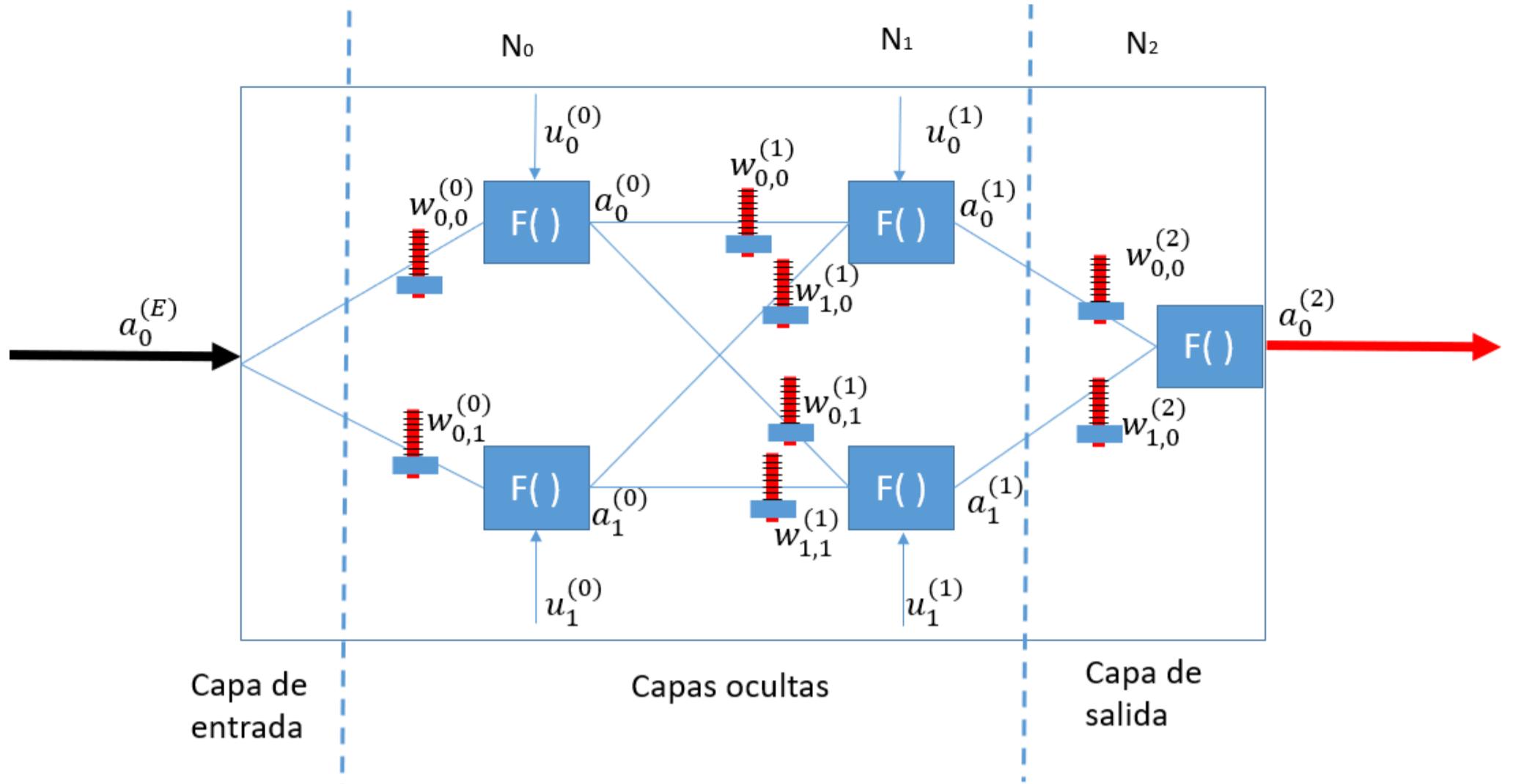


Ilustración 338: Esquema de un perceptrón multicapa

Se buscan los caminos para $w_{0,0}^{(0)}$, entonces hay dos marcados en rojo

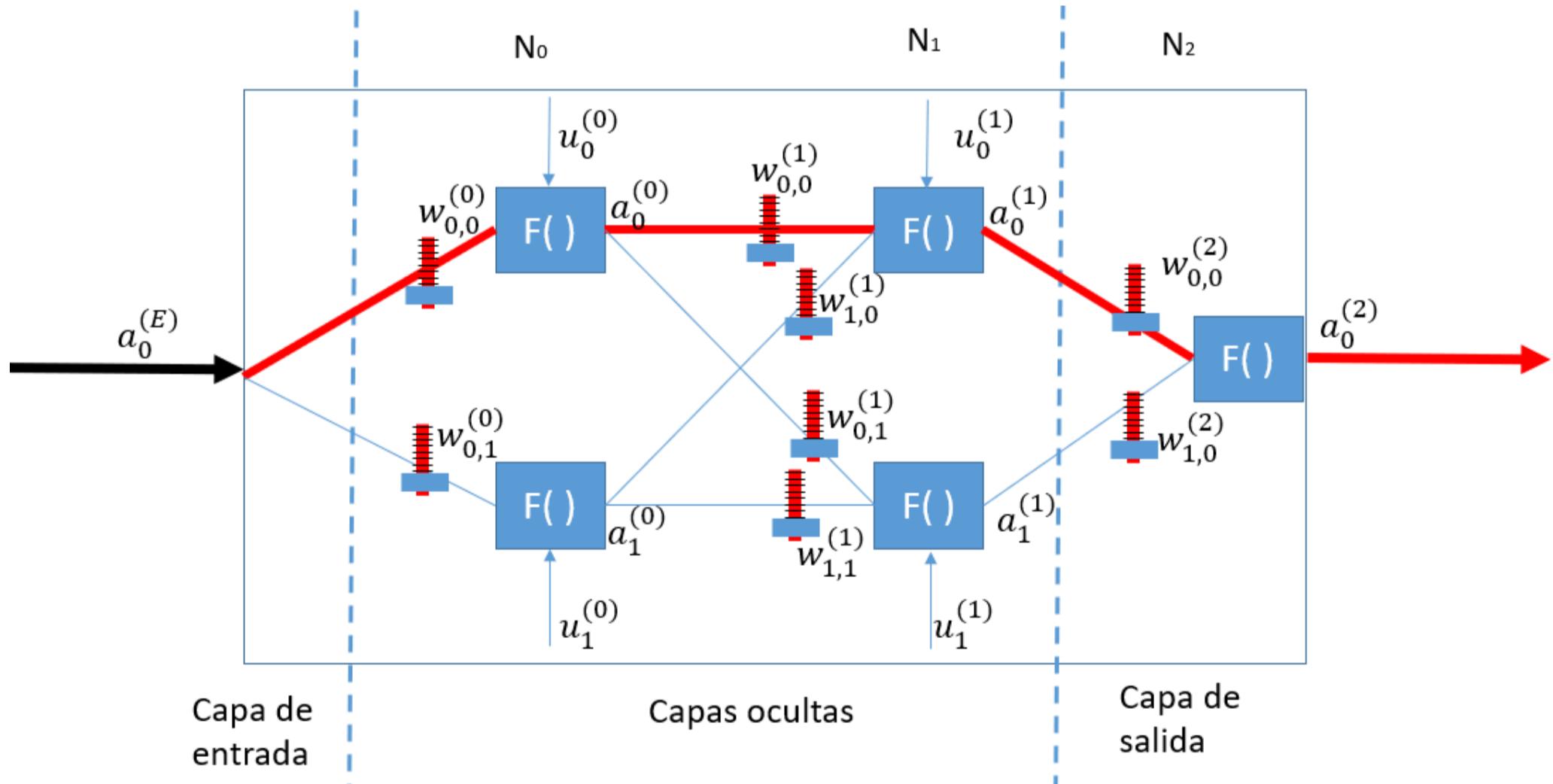


Ilustración 339: Primer camino para ese peso

Y

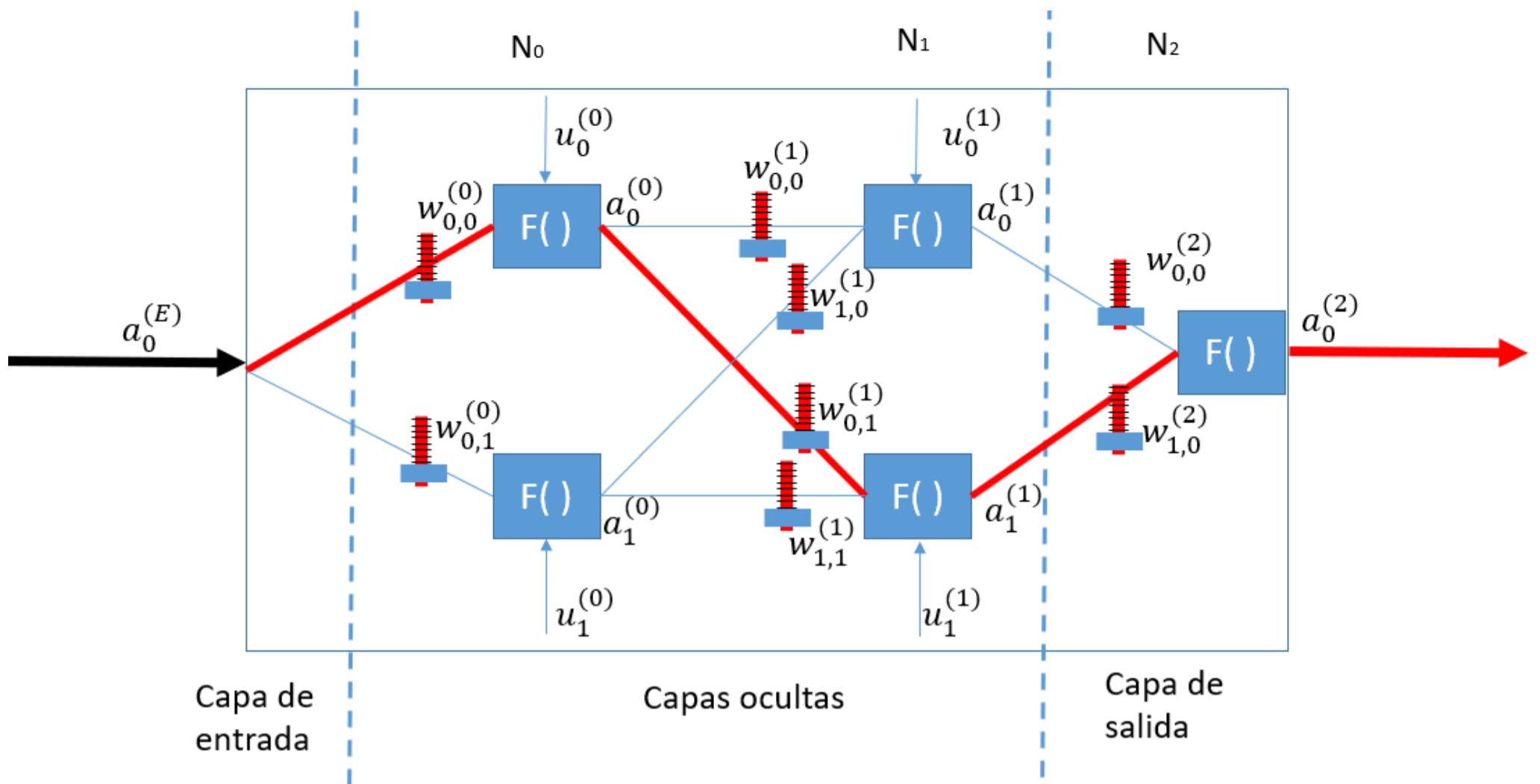


Ilustración 340: Segundo camino para ese peso

Luego la siguiente expresión para la derivada parcial con respecto a $w_{0,0}^{(0)}$ es seguir los dos caminos, sumando ambos

$$\frac{\partial a_0^{(2)}}{\partial w_{0,0}^{(0)}} = a_0^{(2)} * (1 - a_0^{(2)}) * w_{0,0}^{(2)} * a_0^{(1)} * (1 - a_0^{(1)}) * w_{0,0}^{(1)} * a_0^{(0)} * (1 - a_0^{(0)}) * a_0^{(E)} + \\ a_0^{(2)} * (1 - a_0^{(2)}) * w_{1,0}^{(2)} * a_1^{(1)} * (1 - a_1^{(1)}) * w_{0,1}^{(1)} * a_0^{(0)} * (1 - a_0^{(0)}) * a_0^{(E)}$$

Recomendado ir de la entrada a la salida para ver cómo se incrementa el nivel de las capas

$$\frac{\partial a_0^{(2)}}{\partial w_{0,0}^{(0)}} = a_0^{(E)} * a_0^{(0)} * (1 - a_0^{(0)}) * w_{0,0}^{(1)} * a_0^{(1)} * (1 - a_0^{(1)}) * w_{0,0}^{(2)} * a_0^{(2)} * (1 - a_0^{(2)}) + \\ a_0^{(E)} * a_0^{(0)} * (1 - a_0^{(0)}) * w_{0,1}^{(1)} * a_1^{(1)} * (1 - a_1^{(1)}) * w_{1,0}^{(2)} * a_0^{(2)} * (1 - a_0^{(2)})$$

Y así poder generalizar

$$\frac{\partial a_0^{(2)}}{\partial w_{0,0}^{(0)}} = a_0^{(E)} * a_0^{(0)} * (1 - a_0^{(0)}) * \left[\sum_{j=0}^{N_2-1} w_{0,j}^{(1)} * a_j^{(1)} * (1 - a_j^{(1)}) * w_{j,0}^{(2)} \right] * a_0^{(2)} * (1 - a_0^{(2)})$$

La ventaja es que, si las capas ocultas tienen más neuronas, sería cambiar el límite máximo en la sumatoria.

Renombrando la entrada y salida del perceptrón así:

$$a_0^{(E)} = x_0$$

$$a_0^{(2)} = y_0$$

Entonces

$$\frac{\partial y_0}{\partial w_{0,0}^{(0)}} = x_0 * a_0^{(0)} * (1 - a_0^{(0)}) * \left[\sum_{j=0}^{N_2-1} w_{0,j}^{(1)} * a_j^{(1)} * (1 - a_j^{(1)}) * w_{j,0}^{(2)} \right] * y_0 * (1 - y_0)$$

Con un perceptrón con más entradas y salidas, $N_0 = 4$, $N_1 = 4$, $N_2 = 2$, y si se desea dar con $\frac{\partial y_0}{\partial w_{0,0}^{(0)}}$, hay que considerar los diferentes caminos:

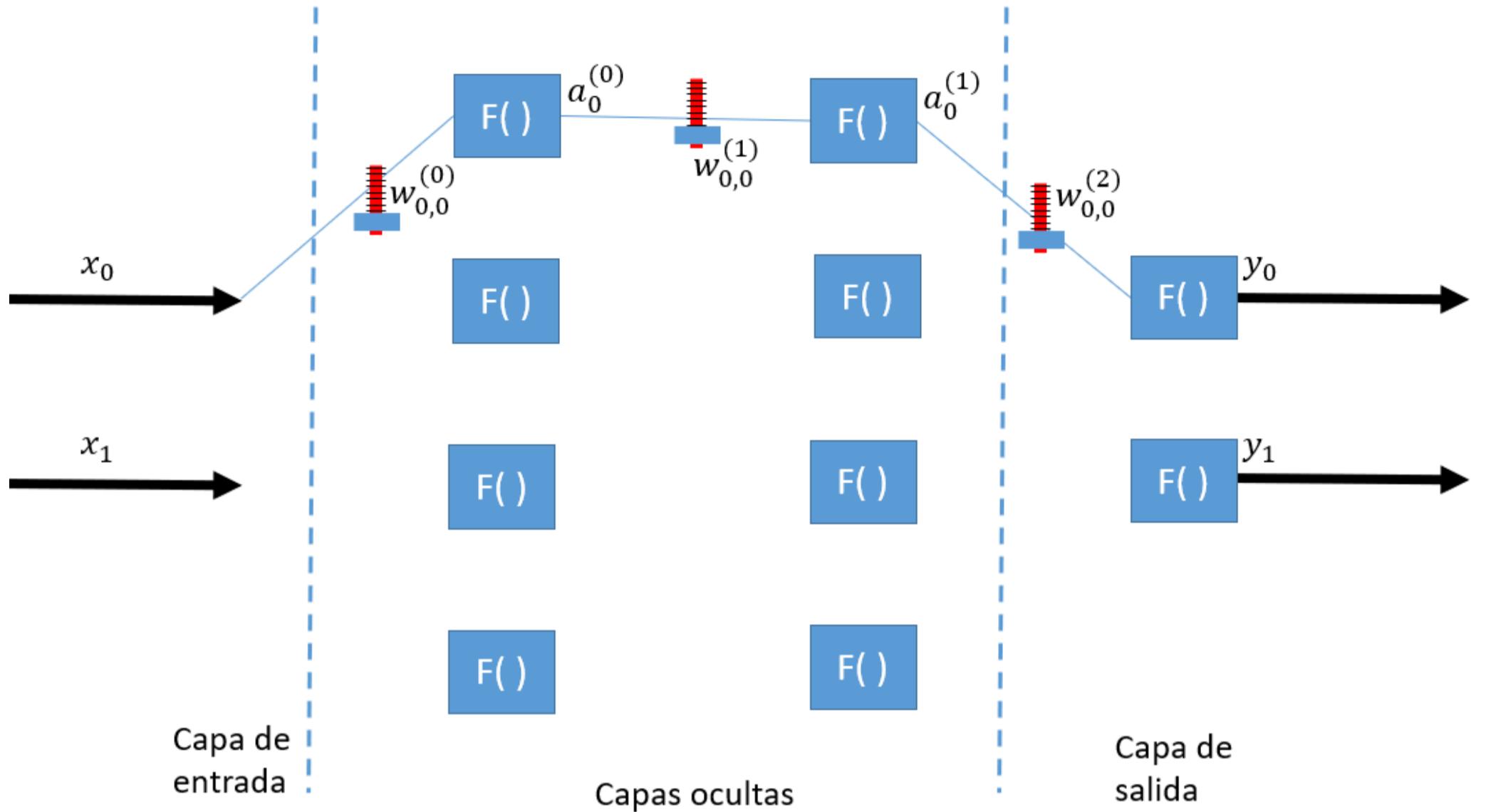


Ilustración 341: Primer camino

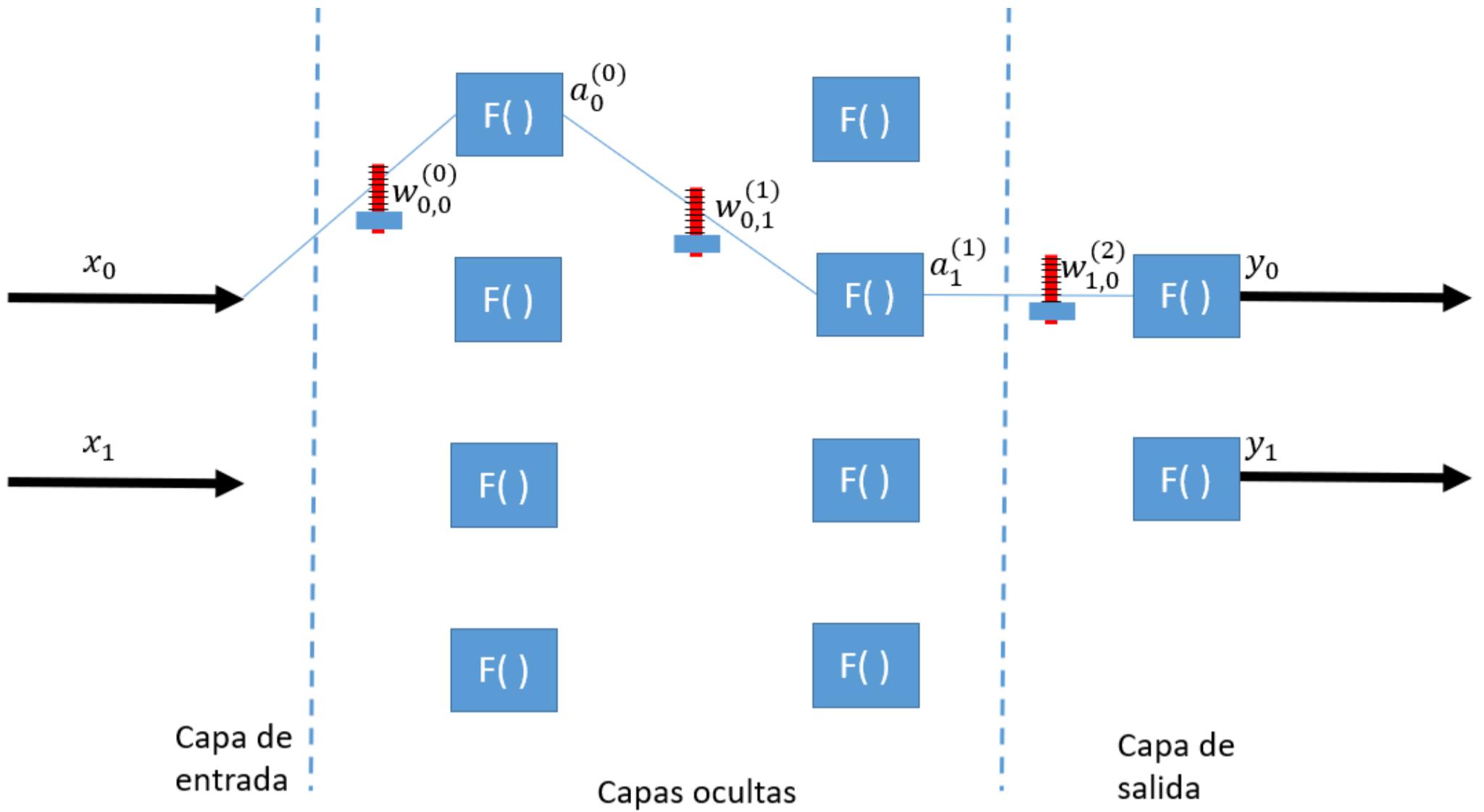


Ilustración 342: Segundo camino

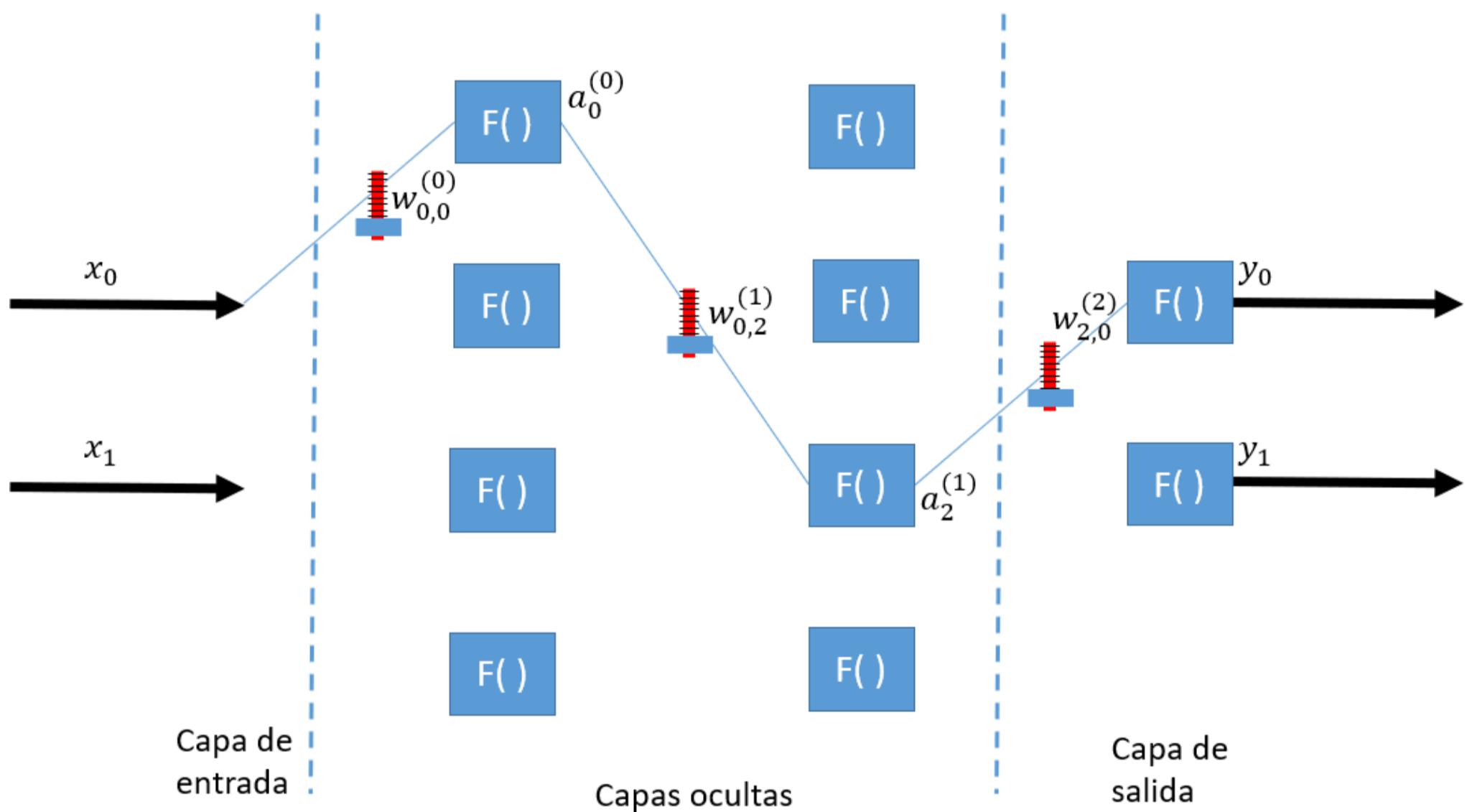


Ilustración 343: Tercer camino

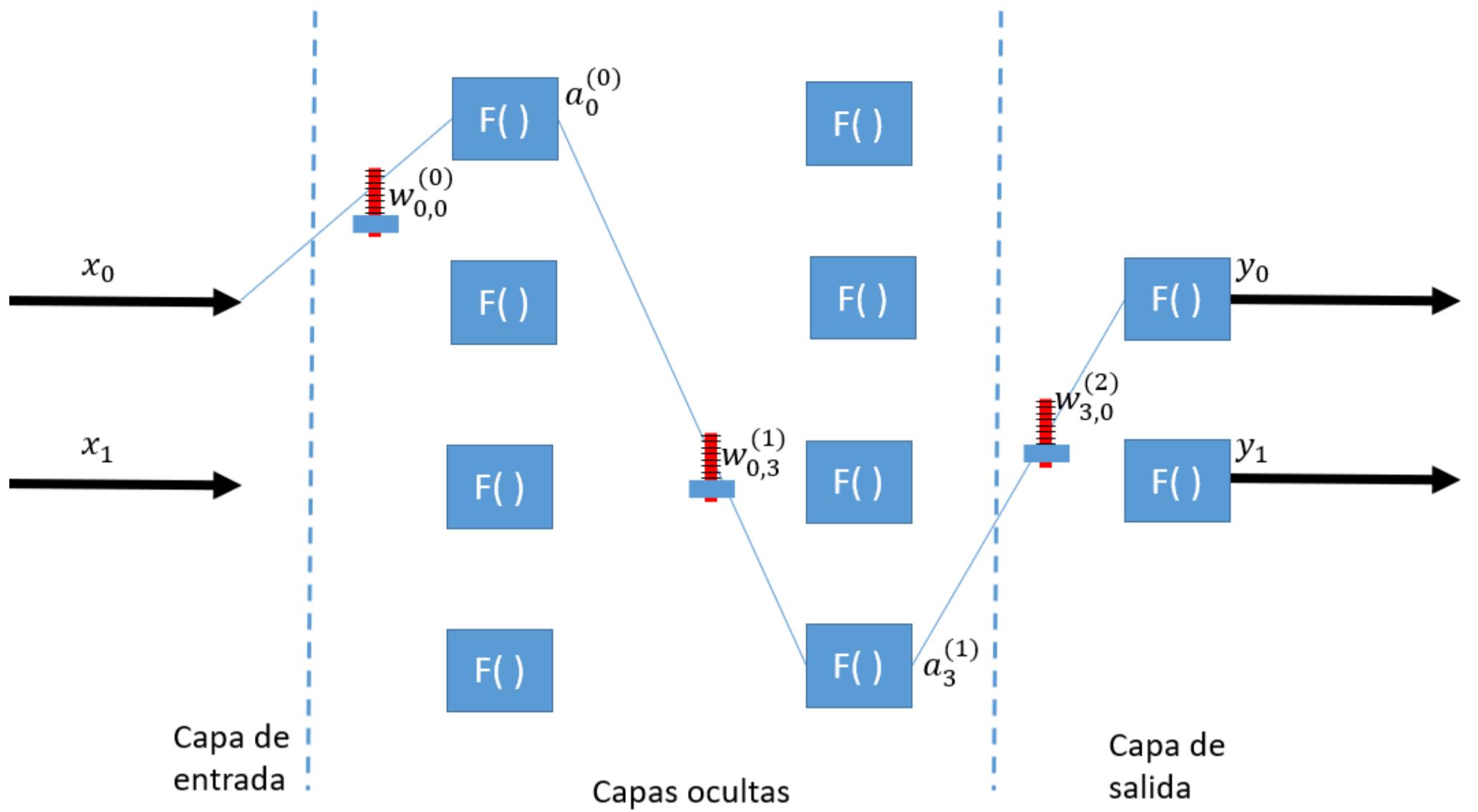


Ilustración 344: Cuarto camino

Luego

$$\frac{\partial y_0}{\partial w_{0,0}^{(0)}} = x_0 * a_0^{(0)} * (1 - a_0^{(0)}) * \left[\sum_{p=0}^{N_1-1} w_{0,p}^{(1)} * a_p^{(1)} * (1 - a_p^{(1)}) * w_{p,0}^{(2)} \right] * y_0 * (1 - y_0)$$

Generalizando

$$\frac{\partial y_i}{\partial w_{j,k}^{(0)}} = x_j * a_k^{(0)} * \left(1 - a_k^{(0)}\right) * \left[\sum_{p=0}^{N_1-1} w_{k,p}^{(1)} * a_p^{(1)} * \left(1 - a_p^{(1)}\right) * w_{p,i}^{(2)} \right] * y_i * (1 - y_i)$$

Donde i=0 a 1, j=0 a 3, k=0 a 3

¿Y para los $W^{(1)}$?

$$\frac{\partial y_i}{\partial w_{j,k}^{(1)}} = a_j^{(0)} * a_k^{(1)} * \left(1 - a_k^{(1)}\right) * w_{k,i}^{(2)} * y_i * (1 - y_i)$$

¿Y para los $W^{(2)}$?

$$\frac{\partial y_i}{\partial w_{j,i}^{(2)}} = a_j^{(1)} * y_i * (1 - y_i)$$

¿Y los umbrales $u^{(0)}$?

$$\frac{\partial y_i}{\partial u_k^{(0)}} = 1 * a_k^{(0)} * \left(1 - a_k^{(0)}\right) * \left[\sum_{p=0}^{N_1-1} w_{k,p}^{(1)} * a_p^{(1)} * \left(1 - a_p^{(1)}\right) * w_{p,i}^{(2)} \right] * y_i * (1 - y_i)$$

Donde i=0 a 1, k=0 a 3

¿Y los umbrales $u^{(1)}$?

$$\frac{\partial y_i}{\partial u_k^{(1)}} = 1 * a_k^{(1)} * \left(1 - a_k^{(1)}\right) * w_{k,i}^{(2)} * y_i * (1 - y_i)$$

¿Y los umbrales $u^{(2)}$?

$$\frac{\partial y_i}{\partial u_i^{(2)}} = 1 * y_i * (1 - y_i)$$

Tratamiento del error en el algoritmo de propagación hacia atrás

Se tiene la siguiente tabla

Entrada X_0	Entrada X_1	Valor esperado de salida S_0	Valor esperado de salida S_1
1	0	0	1
0	0	1	1
0	1	0	0

Pero en realidad se está obteniendo con el perceptrón estas salidas

Entrada X_0	Entrada X_1	Salida real Y_0	Salida real Y_1
1	0	1	1
0	0	1	0
0	1	0	1

Hay un error evidente con las salidas porque no coinciden con lo esperado. ¿Qué hacer? Ajustar los pesos y los umbrales.

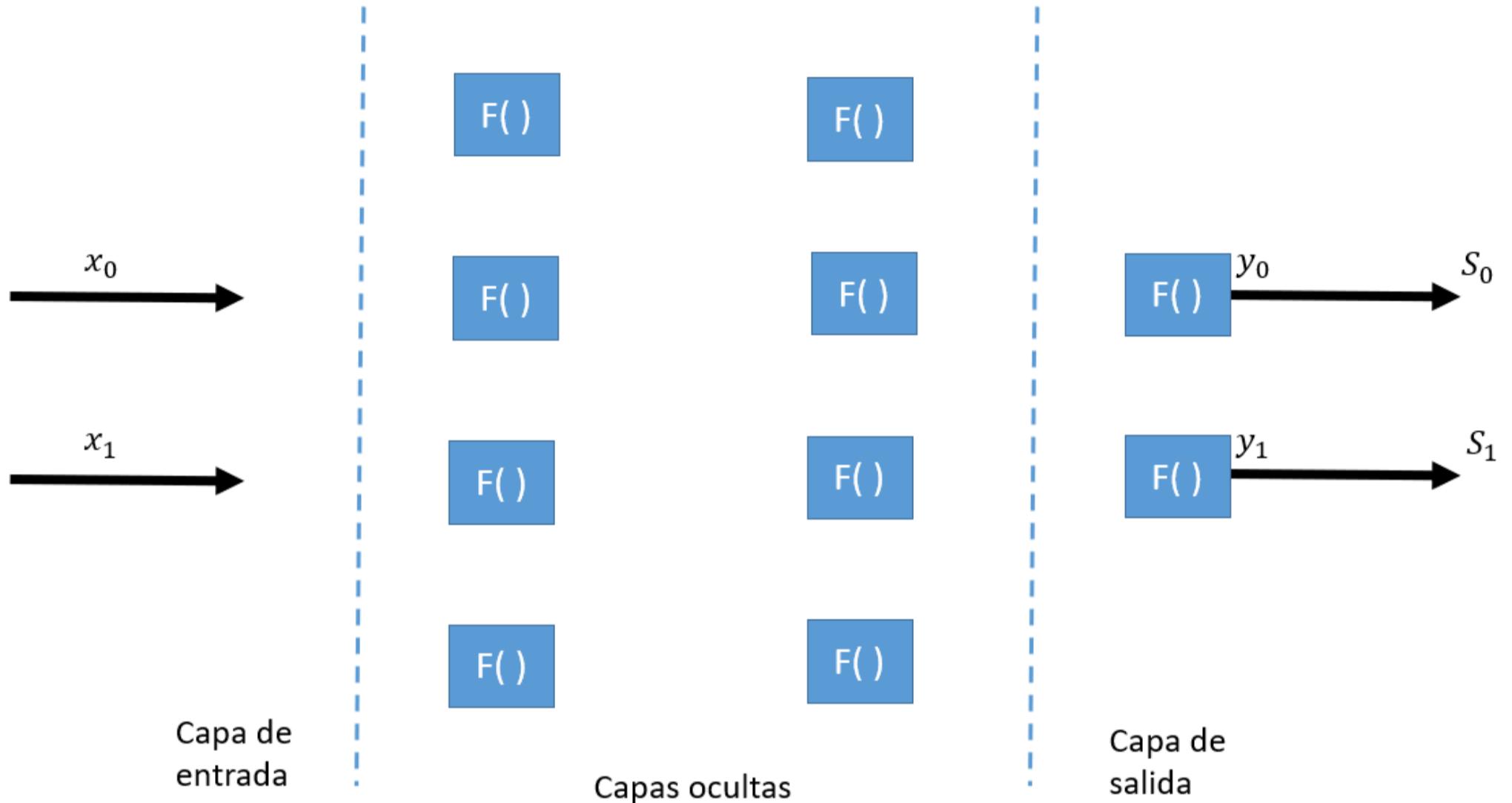


Ilustración 345: Dos entradas y dos salidas

Si se tomaran las salidas y_0 y y_1 como coordenadas e igualmente S_0 y S_1 , se obtiene lo siguiente:

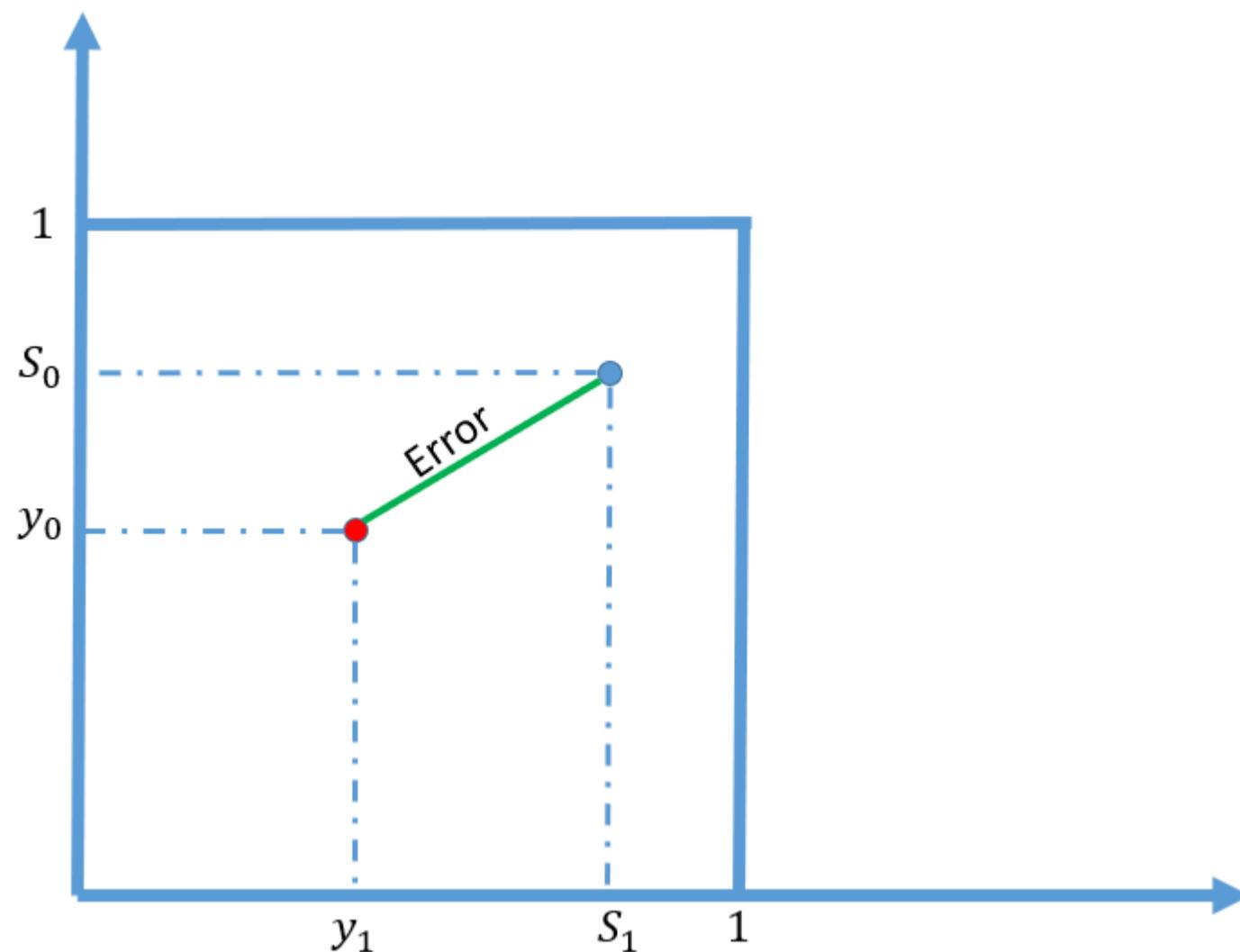


Ilustración 346: Representación del error

Como la función de salida de las neuronas es la sigmoidea, la salida está entre 0 y 1.

En el gráfico de color verde está el error y para calcularlo es usar la fórmula de distancia entre dos puntos en un plano:

$$\text{Error} = \sqrt[2]{(S_1 - y_1)^2 + (S_0 - y_0)^2}$$

Para minimizar ese error hay que considerar que dado:

$$F(x) = \sqrt[2]{g(x)}$$

Al derivar:

$$\partial F(x) = \frac{\partial g(x)}{2 * \sqrt[2]{g(x)}}$$

Y como hay que minimizar se iguala esa derivada a cero

$$\partial F(x) = \frac{\partial g(x)}{2 * \sqrt[2]{g(x)}} = 0$$

Luego

$$\partial g(x) = 0$$

En otras palabras, la raíz cuadrada de $F(x)$, es irrelevante cuando se busca minimizar, porque lo importante es minimizar el interior. Luego la ecuación del error pasa a ser:

$$\text{Error} = (S_1 - y_1)^2 + (S_0 - y_0)^2$$

Que es más sencilla de evaluar. El siguiente paso es multiplicarla por unas constantes quedando así:

$$\text{Error} = \frac{1}{2} (S_1 - y_1)^2 + \frac{1}{2} (S_0 - y_0)^2$$

¿Y por qué se hizo eso? Para hacer que la derivada de Error sea más sencilla. Y no hay que preocuparse porque afecte los resultados: como se busca minimizar, esas constantes no afectan el procedimiento.

iOJO! Hay que recordar que y_0 , y_1 , varían, en cambio, S_0 , S_1 son constantes porque son los valores esperados.

Hay que considerar esta regla matemática: Si P es una función con varias variables independientes, es decir: $P(m,n)$ y Q también es otra función con esas mismas variables independientes, es decir: $Q(m,n)$ y hay una *superfunción* que hace uso de P y Q, es decir: $K(P,Q)$, entonces para derivar a K por una de las variables independientes, tenemos:

$$\frac{\partial K}{\partial m} = \frac{\partial K}{\partial P} * \frac{\partial P}{\partial m} + \frac{\partial K}{\partial Q} * \frac{\partial Q}{\partial m}$$

o

$$\frac{\partial K}{\partial n} = \frac{\partial K}{\partial P} * \frac{\partial P}{\partial n} + \frac{\partial K}{\partial Q} * \frac{\partial Q}{\partial n}$$

Luego

$$\frac{\partial \text{Error}}{\partial \blacksquare} = \frac{\partial \text{Error}}{\partial y_0} * \frac{\partial y_0}{\partial \blacksquare} + \frac{\partial \text{Error}}{\partial y_1} * \frac{\partial y_1}{\partial \blacksquare}$$

¿Y qué es ese cuadro relleno negro? Puede ser algún peso o algún umbral. Generalizando:

$$\frac{\partial \text{Error}}{\partial \blacksquare} = \sum_{i=0}^{N_2-1} \left(\frac{\partial \text{Error}}{\partial y_i} * \frac{\partial y_i}{\partial \blacksquare} \right)$$

Sabiendo que

$$\text{Error} = \frac{1}{2} (S_1 - y_1)^2 + \frac{1}{2} (S_0 - y_0)^2$$

Entonces la derivada de Error con respecto a y_0 es:

$$\frac{\partial \text{Error}}{\partial y_0} = y_0 - S_0$$

Generalizando

$$\frac{\partial \text{Error}}{\partial y_i} = y_i - S_i$$

En el ejemplo, hay dos salidas Y_0 y Y_1 que están en la capa de salida N_2

Luego, la sumatoria sería de 0 a 1

$$\frac{\partial \text{Error}}{\partial \blacksquare} = \sum_{i=0}^{N_2-1} \left((y_i - S_i) * \frac{\partial y_i}{\partial \blacksquare} \right)$$

Queda entonces el cuadro relleno negro que como se mencionó anteriormente puede ser un peso o un umbral. Entonces si se tiene por ejemplo que:

$$\blacksquare = w_{j,i}^{(2)}$$

Entonces como hay una i en particular, la sumatoria se retira luego.

$$\frac{\partial \text{Error}}{\partial w_{j,i}^{(2)}} = (y_i - S_i) * \frac{\partial y_i}{\partial w_{j,i}^{(2)}}$$

Y como

$$\frac{\partial y_i}{\partial w_{j,i}^{(2)}} = a_j^{(1)} * y_i * (1 - y_i)$$

Luego

$$\frac{\partial Error}{\partial w_{j,i}^{(2)}} = (y_i - S_i) * a_j^{(1)} * y_i * (1 - y_i)$$

Ordenando

$$\frac{\partial Error}{\partial w_{j,i}^{(2)}} = a_j^{(1)} * (y_i - S_i) * y_i * (1 - y_i)$$

De nuevo la derivada del error

$$\frac{\partial Error}{\partial \blacksquare} = \sum_{i=0}^{N_2-1} \left((y_i - S_i) * \frac{\partial y_i}{\partial \blacksquare} \right)$$

Suponiendo que

$$\blacksquare = w_{j,k}^{(1)}$$

Entonces

$$\frac{\partial Error}{\partial w_{j,k}^{(1)}} = \sum_{i=1}^{N_2-1} \left((y_i - S_i) * \frac{\partial y_i}{\partial w_{j,k}^{(1)}} \right)$$

Y como

$$\frac{\partial y_i}{\partial w_{j,k}^{(1)}} = a_j^{(0)} * a_k^{(1)} * (1 - a_k^{(1)}) * w_{k,i}^{(2)} * y_i * (1 - y_i)$$

Entonces

$$\frac{\partial Error}{\partial w_{j,k}^{(1)}} = \sum_{i=0}^{N_2-1} \left((y_i - S_i) * a_j^{(0)} * a_k^{(1)} * (1 - a_k^{(1)}) * w_{k,i}^{(2)} * y_i * (1 - y_i) \right)$$

Simplificando

$$\frac{\partial Error}{\partial w_{j,k}^{(1)}} = a_j^{(0)} * a_k^{(1)} * (1 - a_k^{(1)}) * \sum_{i=0}^{N_2-1} \left((y_i - S_i) * w_{k,i}^{(2)} * y_i * (1 - y_i) \right)$$

De nuevo la derivada del error

$$\frac{\partial \text{Error}}{\partial \blacksquare} = \sum_{i=0}^{N_2-1} \left((y_i - S_i) * \frac{\partial y_i}{\partial \blacksquare} \right)$$

Suponiendo que

$$\blacksquare = w_{j,k}^{(0)}$$

Luego la derivada del error es:

$$\frac{\partial \text{Error}}{\partial w_{j,k}^{(0)}} = \sum_{i=0}^{N_2-1} \left((y_i - S_i) * \frac{\partial y_i}{\partial w_{j,k}^{(0)}} \right)$$

Y como se vio anteriormente que

$$\frac{\partial y_i}{\partial w_{j,k}^{(0)}} = x_j * a_k^{(0)} * (1 - a_k^{(0)}) * \left[\sum_{p=0}^{N_1-1} w_{k,p}^{(1)} * a_p^{(1)} * (1 - a_p^{(1)}) * w_{p,i}^{(2)} \right] * y_i * (1 - y_i)$$

Luego reemplazando en la expresión se obtiene:

$$\frac{\partial \text{Error}}{\partial w_{j,k}^{(0)}} = \sum_{i=0}^{N_2-1} \left((y_i - S_i) * x_j * a_k^{(0)} * (1 - a_k^{(0)}) * \left[\sum_{p=0}^{N_1-1} w_{k,p}^{(1)} * a_p^{(1)} * (1 - a_p^{(1)}) * w_{p,i}^{(2)} \right] * y_i * (1 - y_i) \right)$$

Simplificando

$$\begin{aligned} \frac{\partial \text{Error}}{\partial w_{j,k}^{(0)}} &= x_j * a_k^{(0)} * (1 - a_k^{(0)}) * \sum_{i=0}^{N_2-1} \left((y_i - S_i) * \left[\sum_{p=0}^{N_1-1} w_{k,p}^{(1)} * a_p^{(1)} * (1 - a_p^{(1)}) * w_{p,i}^{(2)} \right] * y_i * (1 - y_i) \right) \\ \frac{\partial \text{Error}}{\partial w_{j,k}^{(0)}} &= x_j * a_k^{(0)} * (1 - a_k^{(0)}) * \sum_{p=0}^{N_1-1} \left[w_{k,p}^{(1)} * a_p^{(1)} * (1 - a_p^{(1)}) * \sum_{i=0}^{N_2-1} \left(w_{p,i}^{(2)} * (y_i - S_i) * y_i * (1 - y_i) \right) \right] \end{aligned}$$

En limpio las fórmulas para los pesos son:

$$\frac{\partial \text{Error}}{\partial w_{j,i}^{(2)}} = a_j^{(1)} * (y_i - S_i) * y_i * (1 - y_i)$$

$$\frac{\partial \text{Error}}{\partial w_{j,k}^{(1)}} = a_j^{(0)} * a_k^{(1)} * (1 - a_k^{(1)}) * \sum_{i=0}^{N_2-1} \left(w_{k,i}^{(2)} * (y_i - S_i) * y_i * (1 - y_i) \right)$$

$$\frac{\partial \text{Error}}{\partial w_{j,k}^{(0)}} = x_j * a_k^{(0)} * (1 - a_k^{(0)}) * \sum_{p=0}^{N_1-1} \left[w_{k,p}^{(1)} * a_p^{(1)} * (1 - a_p^{(1)}) * \sum_{i=0}^{N_2-1} \left(w_{p,i}^{(2)} * (y_i - S_i) * y_i * (1 - y_i) \right) \right]$$

Y para los umbrales sería:

$$\frac{\partial \text{Error}}{\partial u_i^{(2)}} = (y_i - S_i) * y_i * (1 - y_i)$$

$$\frac{\partial \text{Error}}{\partial u_k^{(1)}} = a_k^{(1)} * (1 - a_k^{(1)}) * \sum_{i=0}^{N_2-1} \left(w_{k,i}^{(2)} * (y_i - S_i) * y_i * (1 - y_i) \right)$$

$$\frac{\partial \text{Error}}{\partial u_k^{(0)}} = a_k^{(0)} * (1 - a_k^{(0)}) * \sum_{p=0}^{N_1-1} \left[w_{k,p}^{(1)} * a_p^{(1)} * (1 - a_p^{(1)}) * \sum_{i=0}^{N_2-1} \left(w_{p,i}^{(2)} * (y_i - S_i) * y_i * (1 - y_i) \right) \right]$$

Variando los pesos y umbrales con el algoritmo de propagación hacia atrás

La fórmula de variación de los pesos y umbrales es:

$$\begin{aligned} w_{j,i}^{(2)} &\leftarrow w_{j,i}^{(2)} - \alpha * \frac{\partial Error}{\partial w_{j,i}^{(2)}} \\ w_{j,k}^{(1)} &\leftarrow w_{j,k}^{(1)} - \alpha * \frac{\partial Error}{\partial w_{j,k}^{(1)}} \\ w_{j,k}^{(0)} &\leftarrow w_{j,k}^{(0)} - \alpha * \frac{\partial Error}{\partial w_{j,k}^{(0)}} \\ u_i^{(2)} &\leftarrow u_i^{(2)} - \alpha * \frac{\partial Error}{\partial u_i^{(2)}} \\ u_k^{(1)} &\leftarrow u_k^{(1)} - \alpha * \frac{\partial Error}{\partial u_k^{(1)}} \\ u_k^{(0)} &\leftarrow u_k^{(0)} - \alpha * \frac{\partial Error}{\partial u_k^{(0)}} \end{aligned}$$

Donde α es el factor de aprendizaje con un valor pequeño entre 0.1 y 0.9

Implementación en C# del perceptrón multicapa

El siguiente modelo entidad-relación muestra cómo se compone un perceptrón multicapa



Ilustración 347: Modelo del perceptrón

Un perceptrón tiene dos o más capas (mínimo una oculta y la de salida). Una capa tiene uno o más neuronas.

Para implementarlo se hace uso de clases y listas.



Ilustración 348: Modelo del perceptrón

Esta sería la plantilla del programa:

K/008.cs

```
namespace Ejemplo {
    class Program {
        static void Main() {
        }
    }

    class Perceptron {
        List<Capa> Capas;
    }

    class Capa {
        List<Neurona> Neuronas;
    }

    class Neurona {
    }
}
```

Cada neurona tiene los pesos de entrada y el umbral. En el constructor se inicializan los pesos y el umbral al azar. Así quedaría el código:

K/009.cs

```
namespace Ejemplo {
    class Program {
        static void Main() {
    }

    class Perceptron {
        List<Capa> Capas;
    }

    class Capa {
        List<Neurona> Neuronas;
    }

    class Neurona {
        private List<double> Pesos; //Los pesos para cada entrada
        double Umbral; //El peso del umbral

        //Inicializa los pesos y umbral con valores al azar
        public Neurona(Random Azar, int TotalEntradas) {
            Pesos = [];
            for (int Contador = 0; Contador < TotalEntradas; Contador++)
                Pesos.Add(Azar.NextDouble());
            Umbral = Azar.NextDouble();
        }
    }
}
```

Se añade a la clase neurona, el método CalculaSalida() que tiene como parámetro un arreglo unidimensional, el cual tiene los datos de entrada.

K/010.cs

```
namespace Ejemplo {
    class Program {
        static void Main() {
        }
    }

    class Perceptron {
        List<Capa> Capas;
    }

    class Capa {
        List<Neurona> Neuronas;
    }

    class Neurona {
        private List<double> Pesos; //Los pesos para cada entrada
        double Umbral; //El peso del umbral

        //Inicializa los pesos y umbral con un valor al azar
        public Neurona(Random Azar, int TotalEntradas) {
            Pesos = [];
            for (int Contador = 0; Contador < TotalEntradas; Contador++)
                Pesos.Add(Azar.NextDouble());
            Umbral = Azar.NextDouble();
        }

        //Calcula la salida de la neurona dependiendo de las entradas
        public double CalculaSalida(List<double> Entradas) {
            double Valor = 0;
            for (int Contador = 0; Contador < Pesos.Count; Contador++)
                Valor += Entradas[Contador] * Pesos[Contador];
            Valor += Umbral;
            return 1 / (1 + Math.Exp(-Valor));
        }
    }
}
```

La clase **Capa** almacena sus propias neuronas y la salida de cada una de esas neuronas en una lista salidas para facilitar los cálculos más adelante. Se añade el método en que calcula la salida de cada neurona y guarda ese resultado en el listado de "salidas".

K/011.cs

```
namespace Ejemplo {
    class Program {
        static void Main() {
        }

    }

    class Perceptron {
        List<Capa> Capas;
    }

    class Capa {
        List<Neurona> Neuronas; //Las neuronas que tendrá la capa
        List<double> Salidas; //Almacena la salida de cada neurona

        public Capa(Random Azar, int TotalNeuronas, int TotalEntradas) {
            Neuronas = [];
            Salidas = [];

            //Genera las neuronas e inicializa sus salidas
            for (int Contador = 0; Contador < TotalNeuronas; Contador++) {
                Neuronas.Add(new Neurona(Azar, TotalEntradas));
                Salidas.Add(0);
            }
        }

        //Calcula la salida de cada neurona de la capa
        public void CalculaCapa(List<double> Entradas) {
            for (int cont = 0; cont < Neuronas.Count; cont++)
                Salidas[cont] = Neuronas[cont].CalculaSalida(Entradas);
        }
    }

    class Neurona {
        private List<double> Pesos; //Los pesos para cada entrada
        double Umbral; //El peso del umbral

        //Inicializa los pesos y umbral con un valor al azar
        public Neurona(Random Azar, int TotalEntradas) {
            Pesos = [];
            for (int Contador = 0; Contador < TotalEntradas; Contador++)
                Pesos.Add(Azar.NextDouble());
            Umbral = Azar.NextDouble();
        }

        //Calcula la salida de la neurona dependiendo de las entradas
        public double CalculaSalida(List<double> Entradas) {
            double Valor = 0;
            for (int Contador = 0; Contador < Pesos.Count; Contador++)
                Valor += Entradas[Contador] * Pesos[Contador];
            Valor += Umbral;
            return 1 / (1 + Math.Exp(-Valor));
        }
    }
}
```

```

namespace Ejemplo {
    class Program {
        static void Main() {
    }

    class Perceptron {
        List<Capa>? Capas;

        //Crea las diversas capas
        public void CreaCapas(Random Azar, int NumeroEntradas, int TotalNeuronasCapa0, int
TotalNeuronasCapa1, int TotalNeuronasCapa2) {
            Capas =
            [
                //Crea la capa 0
                new Capa(Azar, TotalNeuronasCapa0, NumeroEntradas),

                //Crea la capa 1 (el número de entradas es el número de neuronas de la capa anterior)
                new Capa(Azar, TotalNeuronasCapa1, TotalNeuronasCapa0),

                //Crea la capa 2 (el número de entradas es el número de neuronas de la capa anterior)
                new Capa(Azar, TotalNeuronasCapa2, TotalNeuronasCapa1),
            ];
        }
    }

    class Capa {
        List<Neurona> Neuronas; //Las neuronas que tendrá la capa
        List<double> Salidas; //Almacena la salida de cada neurona

        public Capa(Random Azar, int TotalNeuronas, int TotalEntradas) {
            Neuronas = [];
            Salidas = [];

            //Genera las neuronas e inicializa sus salidas
            for (int Contador = 0; Contador < TotalNeuronas; Contador++) {
                Neuronas.Add(new Neurona(Azar, TotalEntradas));
                Salidas.Add(0);
            }
        }

        //Calcula la salida de cada neurona de la capa
        public void CalculaCapa(List<double> Entradas) {
            for (int cont = 0; cont < Neuronas.Count; cont++)
                Salidas[cont] = Neuronas[cont].CalculaSalida(Entradas);
        }
    }
}

class Neurona {
    private List<double> Pesos; //Los pesos para cada entrada
    double Umbral; //El peso del umbral

    //Inicializa los pesos y umbral con un valor al azar
    public Neurona(Random Azar, int TotalEntradas) {
        Pesos = [];
        for (int Contador = 0; Contador < TotalEntradas; Contador++)
            Pesos.Add(Azar.NextDouble());
        Umbral = Azar.NextDouble();
    }

    //Calcula la salida de la neurona dependiendo de las entradas
    public double CalculaSalida(List<double> Entradas) {
        double Valor = 0;
        for (int Contador = 0; Contador < Pesos.Count; Contador++)
            Valor += Entradas[Contador] * Pesos[Contador];
        Valor += Umbral;
        return 1 / (1 + Math.Exp(-Valor));
    }
}

```

```
namespace Ejemplo {
    class Program {
        static void Main() {
        }
    }

    class Perceptron {
        List<Capa>? Capas;

        //Crea las diversas capas
        public void CreaCapas(Random Azar, int NumeroEntradas, int TotalNeuronasCapa0, int
TotalNeuronasCapa1, int TotalNeuronasCapa2) {
            Capas =
            [
                //Crea la capa 0
                new Capa(Azar, TotalNeuronasCapa0, NumeroEntradas),

                //Crea la capa 1 (el número de entradas es el número de neuronas de la capa anterior)
                new Capa(Azar, TotalNeuronasCapa1, TotalNeuronasCapa0),

                //Crea la capa 2 (el número de entradas es el número de neuronas de la capa anterior)
                new Capa(Azar, TotalNeuronasCapa2, TotalNeuronasCapa1),
            ];
        }

        public void calculaSalida(List<double> Entradas) {
            Capas[0].CalculaCapa(Entradas);

            //Las salidas de la capa anterior son las entradas de la siguiente capa
            Capas[1].CalculaCapa(Capas[0].Salidas);
            Capas[2].CalculaCapa(Capas[1].Salidas);
        }
    }

    class Capa {
        List<Neurona> Neuronas; //Las neuronas que tendrá la capa
        public List<double> Salidas; //Almacena la salida de cada neurona

        public Capa(Random Azar, int TotalNeuronas, int TotalEntradas) {
            Neuronas = [];
            Salidas = [];

            //Genera las neuronas e inicializa sus salidas
            for (int Contador = 0; Contador < TotalNeuronas; Contador++) {
                Neuronas.Add(new Neurona(Azar, TotalEntradas));
                Salidas.Add(0);
            }
        }

        //Calcula la salida de cada neurona de la capa
        public void CalculaCapa(List<double> Entradas) {
            for (int cont = 0; cont < Neuronas.Count; cont++)
                Salidas[cont] = Neuronas[cont].CalculaSalida(Entradas);
        }
    }
}

class Neurona {
    private List<double> Pesos; //Los pesos para cada entrada
    double Umbral; //El peso del umbral

    //Inicializa los pesos y umbral con un valor al azar
    public Neurona(Random Azar, int TotalEntradas) {
        Pesos = [];
        for (int Contador = 0; Contador < TotalEntradas; Contador++)
            Pesos.Add(Azar.NextDouble());
        Umbral = Azar.NextDouble();
    }

    //Calcula la salida de la neurona dependiendo de las entradas
    public double CalculaSalida(List<double> Entradas) {
        double Valor = 0;
        for (int Contador = 0; Contador < Pesos.Count; Contador++)
            Valor += Entradas[Contador] * Pesos[Contador];
    }
}
```

```
        Valor += Umbral;  
        return 1 / (1 + Math.Exp(-Valor));  
    }  
}
```

Ejemplo de uso de la clase Perceptron para generar este diseño en particular:

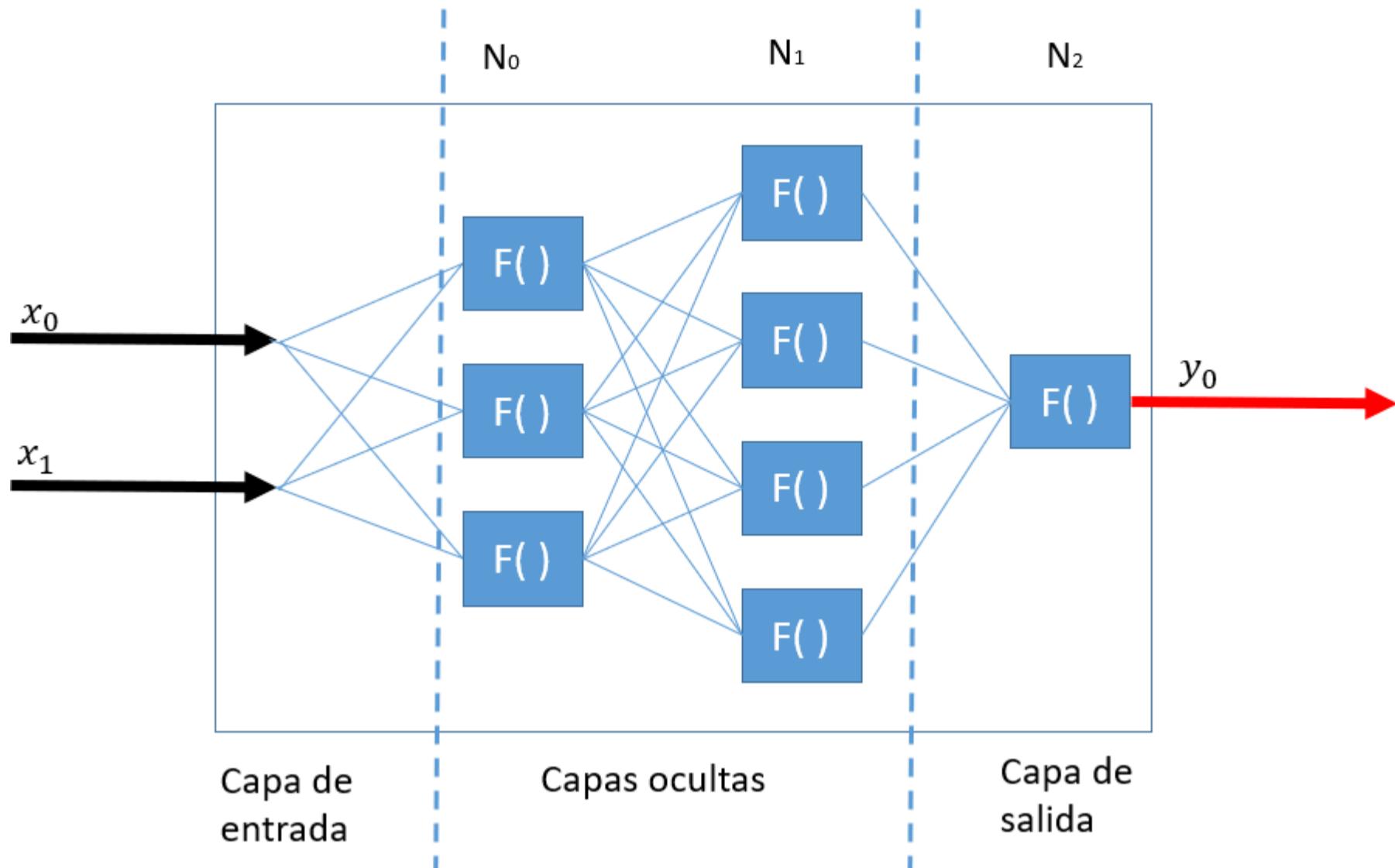


Ilustración 349: Un perceptrón multicapa

K/014.cs

```
class Program {
    static void Main() {
        Random Azar = new(); //Un solo generador de números pseudo-aleatorios
        Perceptron perceptron = new Perceptron();

        int numEntradas = 2; //Número de entradas
        int capa0 = 3; //Total neuronas en la capa 0
        int capa1 = 4; //Total neuronas en la capa 1
        int capa2 = 1; //Total neuronas en la capa 2
        perceptron.CreaCapas(Azar, numEntradas, capa0, capa1, capa2);

        //Estas serán las entradas externas al perceptrón
        List<double> Entradas = new List<double>();
        Entradas.Add(1);
        Entradas.Add(0);

        //Se hace el cálculo
        perceptron.calculaSalida(Entradas);
    }
}
```

Es en el programa principal que se crea el objeto que genera los números pseudoaleatorios y se le envía al perceptrón.

Algoritmo de retro propagación en C#

Las fórmulas del algoritmo de retro propagación para la capa 2:

$$\frac{\partial \text{Error}}{\partial w_{j,i}^{(2)}} = a_j^{(1)} * (y_i - S_i) * y_i * (1 - y_i)$$

Y

$$w_{j,i}^{(2)} \leftarrow w_{j,i}^{(2)} - \alpha * \frac{\partial \text{Error}}{\partial w_{j,i}^{(2)}}$$

Se implementa así:

```
//Procesa pesos capa 2
for (int j = 0; j < NeuronasCapa1; j++) //Va de neurona en neurona de la capa 1
    for (int i = 0; i < NeuronasCapa2; i++) { //Va de neurona en neurona de la capa de salida (capa 2)
        double Yi = Capas[2].Salidas[i]; //Salida de la neurona de la capa de salida
        double Si = SalidaEsperada[i]; //Salida esperada
        double a1j = Capas[1].Salidas[j]; //Salida de la capa 1
        double dE2 = a1j * (Yi - Si) * Yi * (1 - Yi); //La fórmula del error
        Capas[2].Neuronas[i].NuevosPesos[j] = Capas[2].Neuronas[i].Pesos[j] - Alpha * dE2; //Ajusta el
nuevo peso
    }
```

Para la capa 1:

$$\frac{\partial Error}{\partial w_{j,k}^{(1)}} = a_j^{(0)} * a_k^{(1)} * \left(1 - a_k^{(1)}\right) * \sum_{i=0}^{N_2-1} \left(w_{k,i}^{(2)} * (y_i - S_i) * y_i * (1 - y_i)\right)$$

Y

$$w_{j,k}^{(1)} \leftarrow w_{j,k}^{(1)} - \alpha * \frac{\partial Error}{\partial w_{j,k}^{(1)}}$$

Se implementa así:

```
//Procesa pesos capa 1
for (int j = 0; j < NeuronasCapa0; j++) //Va de neurona en neurona de la capa 0
    for (int k = 0; k < NeuronasCapa1; k++) { //Va de neurona en neurona de la capa 1
        double Acumula = 0;
        for (int i = 0; i < NeuronasCapa2; i++) { //Va de neurona en neurona de la capa 2
            double Yi = Capas[2].Salidas[i]; //Salida de la capa 2
            double Si = SalidaEsperada[i]; //Salida esperada
            double W2ki = Capas[2].Neuronas[i].Pesos[k];
            Acumula += W2ki * (Yi - Si) * Yi * (1 - Yi); //Sumatoria
        }
        double a0j = Capas[0].Salidas[j];
        double a1k = Capas[1].Salidas[k];
        double dE1 = a0j * a1k * (1 - a1k) * Acumula;
        Capas[1].Neuronas[k].NuevosPesos[j] = Capas[1].Neuronas[k].Pesos[j] - Alpha * dE1;
    }
```

Para la capa 0:

$$\frac{\partial \text{Error}}{\partial w_{j,k}^{(0)}} = x_j * a_k^{(0)} * (1 - a_k^{(0)}) * \sum_{p=0}^{N_1-1} \left[w_{k,p}^{(1)} * a_p^{(1)} * (1 - a_p^{(1)}) * \sum_{i=0}^{N_2-1} \left(w_{p,i}^{(2)} * (y_i - S_i) * y_i * (1 - y_i) \right) \right]$$

Y

$$w_{j,k}^{(0)} \leftarrow w_{j,k}^{(0)} - \alpha * \frac{\partial \text{Error}}{\partial w_{j,k}^{(0)}}$$

Se implementa así:

```
//Procesa pesos capa 0
for (int j = 0; j < Entradas.Count; j++) //Va de entrada en entrada
    for (int k = 0; k < NeuronasCapa0; k++) { //Va de neurona en neurona de la capa 0
        double Acumula = 0;
        for (int p = 0; p < NeuronasCapa1; p++) { //Va de neurona en neurona de la capa 1
            double InternoAcumula = 0;
            for (int i = 0; i < NeuronasCapa2; i++) { //Va de neurona en neurona de la capa 2
                double Yi = Capas[2].Salidas[i];
                double Si = SalidaEsperada[i]; //Salida esperada
                double W2pi = Capas[2].Neuronas[i].Pesos[p];
                InternoAcumula += W2pi * (Yi - Si) * Yi * (1 - Yi); //Sumatoria interna
            }
            double W1kp = Capas[1].Neuronas[p].Pesos[k];
            double alp = Capas[1].Salidas[p];
            Acumula += W1kp * alp * (1 - alp) * InternoAcumula; //Sumatoria externa
        }
        double xj = Entradas[j];
        double a0k = Capas[0].Salidas[k];
        double dE0 = xj * a0k * (1 - a0k) * Acumula;
        double W0jk = Capas[0].Neuronas[k].Pesos[j];
        Capas[0].Neuronas[k].NuevosPesos[j] = W0jk - Alpha * dE0;
    }
```

Para los umbrales de la capa 2

$$\frac{\partial Error}{\partial u_i^{(2)}} = (y_i - S_i) * y_i * (1 - y_i)$$

$$u_i^{(2)} \leftarrow u_i^{(2)} - \alpha * \frac{\partial Error}{\partial u_i^{(2)}}$$

Se implementa así:

```
//Procesa umbrales capa 2
for (int i = 0; i < NeuronasCapa2; i++) { //Va de neurona en neurona de la capa de salida (capa 2)
    double Yi = Capas[2].Salidas[i]; //Salida de la neurona de la capa de salida
    double Si = SalidaEsperada[i]; //Salida esperada
    double dE2 = (Yi - Si) * Yi * (1 - Yi);
    Capas[2].Neuronas[i].NuevoUmbbral = Capas[2].Neuronas[i].Umbral - Alpha * dE2;
}
```

Para los umbrales de la capa 1

$$\frac{\partial Error}{\partial u_k^{(1)}} = a_k^{(1)} * \left(1 - a_k^{(1)}\right) * \sum_{i=0}^{N_2-1} \left(w_{k,i}^{(2)} * (y_i - S_i) * y_i * (1 - y_i)\right)$$
$$u_k^{(1)} \leftarrow u_k^{(1)} - \alpha * \frac{\partial Error}{\partial u_k^{(1)}}$$

Se implementa así:

```
//Procesa umbrales capa 1
for (int k = 0; k < NeuronasCapa1; k++) { //Va de neurona en neurona de la capa 1
    double Acumula = 0;
    for (int i = 0; i < NeuronasCapa2; i++) { //Va de neurona en neurona de la capa 2
        double Yi = Capas[2].Salidas[i]; //Salida de la capa 2
        double Si = SalidaEsperada[i];
        double W2ki = Capas[2].Neuronas[i].Pesos[k];
        Acumula += W2ki * (Yi - Si) * Yi * (1 - Yi);
    }
    double a1k = Capas[1].Salidas[k];
    double dE1 = a1k * (1 - a1k) * Acumula;
    Capas[1].Neuronas[k].NuevoUmbbral = Capas[1].Neuronas[k].Umbbral - Alpha * dE1;
}
```

Para los umbrales de la capa 0

$$\frac{\partial Error}{\partial u_k^{(0)}} = a_k^{(0)} * (1 - a_k^{(0)}) * \sum_{p=0}^{N_1-1} \left[w_{k,p}^{(1)} * a_p^{(1)} * (1 - a_p^{(1)}) * \sum_{i=0}^{N_2-1} \left(w_{p,i}^{(2)} * (y_i - S_i) * y_i * (1 - y_i) \right) \right]$$

$$u_k^{(0)} \leftarrow u_k^{(0)} - \alpha * \frac{\partial Error}{\partial u_k^{(0)}}$$

Se implementa así:

```
//Procesa umbrales capa 0
for (int k = 0; k < NeuronasCapa0; k++) { //Va de neurona en neurona de la capa 0
    double Acumula = 0;
    for (int p = 0; p < NeuronasCapa1; p++) { //Va de neurona en neurona de la capa 1
        double InternoAcumula = 0;
        for (int i = 0; i < NeuronasCapa2; i++) { //Va de neurona en neurona de la capa 2
            double Yi = Capas[2].Salidas[i];
            double Si = SalidaEsperada[i];
            double W2pi = Capas[2].Neuronas[i].Pesos[p];
            InternoAcumula += W2pi * (Yi - Si) * Yi * (1 - Yi);
        }
        double W1kp = Capas[1].Neuronas[p].Pesos[k];
        double alp = Capas[1].Salidas[p];
        Acumula += W1kp * alp * (1 - alp) * InternoAcumula;
    }
    double a0k = Capas[0].Salidas[k];
    double dE0 = a0k * (1 - a0k) * Acumula;
    Capas[0].Neuronas[k].NuevoUmbbral = Capas[0].Neuronas[k].Umbbral - Alpha * dE0;
}
```

Como se puede observar en los códigos, se deducen nuevos pesos y umbrales. Esos nuevos valores posteriormente deben reemplazar los viejos. Luego hay que hacer un cambio a la clase Neurona para que almacene temporalmente los nuevos valores para que cuando se termine de calcularlos, entonces reemplaza los viejos.

El código del encabezado sería así:

```
class Neurona {  
    public List<double> Pesos; //Los pesos para cada entrada  
    public List<double> NuevosPesos; //Nuevos pesos dados por el algoritmo de "backpropagation"  
    public double Umbral; //El peso del umbral  
    public double NuevoUmbral; //Nuevo umbral dado por el algoritmo de "backpropagation"
```

Y tendría un nuevo método que actualizaría los pesos con los nuevos valores, **después** de ejecutar el algoritmo de "backpropagation"

```
//Reemplaza viejos pesos por nuevos  
public void Actualiza() {  
    for (int Contador = 0; Contador < Pesos.Count; Contador++)  
        Pesos[Contador] = NuevosPesos[Contador];  
    Umbral = NuevoUmbral;  
}
```

Significa que en la clase Capa debe haber un método que llama la actualización de las neuronas

```
//Actualiza los pesos y umbrales de las neuronas  
public void Actualiza() {  
    for (int Contador = 0; Contador < Neuronas.Count; Contador++)  
        Neuronas[Contador].Actualiza();  
}
```

Código completo del perceptrón en C#

A continuación, se muestra el código completo en el que se ha creado una clase que implementa el perceptrón (creación, proceso, entrenamiento) y en la clase principal se pone como datos de prueba la tabla del XOR que el perceptrón debe aprender.

K/015.cs

```
namespace Ejemplo {
    class Program {
        static void Main() {
            //Tabla XOR
            int[][] XORrentra = new int[][] {
                new int[] {1, 1},
                new int[] {1, 0},
                new int[] {0, 1},
                new int[] {0, 0}
            };
            int[] XORsale = new int[] { 0, 1, 1, 0 };

            int TotalEntradas = 2; //Número de entradas
            int NeuronasCapa0 = 3; //Total neuronas en la capa 0
            int NeuronasCapa1 = 2; //Total neuronas en la capa 1
            int NeuronasCapa2 = 1; //Total neuronas en la capa 2
            Perceptron RedNeuronal = new(TotalEntradas, NeuronasCapa0, NeuronasCapa1, NeuronasCapa2);

            //Estas serán las dos entradas externas al perceptrón
            List<double> Entradas = [0, 0];

            //Esta será la salida esperada externa al perceptrón
            List<double> SalidaEsperada = [0];

            //Ciclo que entrena la red neuronal
            int TotalCiclos = 90000; //Ciclos de entrenamiento
            for (int Ciclo = 1; Ciclo <= TotalCiclos; Ciclo++) {

                if (Ciclo % 200 == 0) Console.WriteLine("\r\nCiclo: " + Ciclo);

                //Por cada ciclo, se entrena el perceptrón con toda la tabla de XOR
                for (int Conjunto = 0; Conjunto < XORsale.Length; Conjunto++) {

                    //Entradas y salidas esperadas
                    Entradas[0] = XORrentra[Conjunto][0];
                    Entradas[1] = XORrentra[Conjunto][1];
                    SalidaEsperada[0] = XORsale[Conjunto];

                    //Primero calcula la salida del perceptrón con esas entradas
                    RedNeuronal.CalculaSalida(Entradas);

                    //Luego entrena el perceptrón para ajustar los pesos y umbrales
                    RedNeuronal.Entrena(Entradas, SalidaEsperada);

                    //Cada 200 ciclos muestra como progresa el entrenamiento
                    if (Ciclo % 200 == 0) RedNeuronal.SalidaPerceptron(Entradas, SalidaEsperada[0]);
                }
            }

            Console.WriteLine("Finaliza el entrenamiento");
        }
    }

    class Perceptron {
        public List<Capa> Capas;

        //Imprime los datos de las diferentes capas
        public void SalidaPerceptron(List<double> Entradas, double SalidaEsperada) {

            for (int Contador = 0; Contador < Entradas.Count; Contador++)
                Console.Write(Entradas[Contador] + " | ");

            Console.Write(" Esperada: " + SalidaEsperada + " Calculada: ");
            for (int Contador = 0; Contador < Capas[2].Salidas.Count; Contador++) {
                if (Capas[2].Salidas[Contador] >= 0.5)
                    Console.Write(" 1 | ");
                else
                    Console.Write(" 0 | ");
                Console.Write(Capas[2].Salidas[Contador] + " | ");
            }
            Console.WriteLine(" ");
        }
    }
}
```

```

//Crea las diversas capas
public Perceptron(int TotalEntradas, int NeuronasCapa0, int NeuronasCapa1, int NeuronasCapa2) {
    Random Azar = new();
    Capas =
    [
        new Capa(Azar, NeuronasCapa0, TotalEntradas), //Crea la capa 0
        new Capa(Azar, NeuronasCapa1, NeuronasCapa0), //Crea la capa 1
        new Capa(Azar, NeuronasCapa2, NeuronasCapa1), //Crea la capa 2
    ];
}

//Dada las entradas al perceptrón, se calcula la salida de cada capa.
//Con eso se sabrá que salidas se obtienen con los pesos y umbrales actuales.
//Esas salidas son requeridas para el algoritmo de entrenamiento.
public void CalculaSalida(List<double> Entradas) {
    Capas[0].CalculaCapa(Entradas);
    Capas[1].CalculaCapa(Capas[0].Salidas);
    Capas[2].CalculaCapa(Capas[1].Salidas);
}

//Con las salidas previamente calculadas con unas determinadas entradas
//se ejecuta el algoritmo de entrenamiento "Backpropagation"
public void Entrena(List<double> Entradas, List<double> SalidaEsperada) {
    int NeuronasCapa0 = Capas[0].Neuronas.Count;
    int NeuronasCapa1 = Capas[1].Neuronas.Count;
    int NeuronasCapa2 = Capas[2].Neuronas.Count;

    //Factor de aprendizaje
    double Alpha = 0.4;

    //Procesa pesos capa 2
    for (int j = 0; j < NeuronasCapa1; j++) //Va de neurona en neurona de la capa 1
        for (int i = 0; i < NeuronasCapa2; i++) { //Va de neurona en neurona de la capa de
salida (capa 2)
            double Yi = Capas[2].Salidas[i]; //Salida de la neurona de la capa de salida
            double Si = SalidaEsperada[i]; //Salida esperada
            double a1j = Capas[1].Salidas[j]; //Salida de la capa 1
            double dE2 = a1j * (Yi - Si) * Yi * (1 - Yi); //La fórmula del error
            Capas[2].Neuronas[i].NuevosPesos[j] = Capas[2].Neuronas[i].Pesos[j] - Alpha * dE2;
//Ajusta el nuevo peso
        }

    //Procesa pesos capa 1
    for (int j = 0; j < NeuronasCapa0; j++) //Va de neurona en neurona de la capa 0
        for (int k = 0; k < NeuronasCapa1; k++) { //Va de neurona en neurona de la capa 1
            double Acumula = 0;
            for (int i = 0; i < NeuronasCapa2; i++) { //Va de neurona en neurona de la capa 2
                double Yi = Capas[2].Salidas[i]; //Salida de la capa 2
                double Si = SalidaEsperada[i]; //Salida esperada
                double W2ki = Capas[2].Neuronas[i].Pesos[k];
                Acumula += W2ki * (Yi - Si) * Yi * (1 - Yi); //Sumatoria
            }
            double a0j = Capas[0].Salidas[j];
            double a1k = Capas[1].Salidas[k];
            double dE1 = a0j * a1k * (1 - a1k) * Acumula;
            Capas[1].Neuronas[k].NuevosPesos[j] = Capas[1].Neuronas[k].Pesos[j] - Alpha * dE1;
        }

    //Procesa pesos capa 0
    for (int j = 0; j < Entradas.Count; j++) //Va de entrada en entrada
        for (int k = 0; k < NeuronasCapa0; k++) { //Va de neurona en neurona de la capa 0
            double Acumula = 0;
            for (int p = 0; p < NeuronasCapa1; p++) { //Va de neurona en neurona de la capa 1
                double InternoAcumula = 0;
                for (int i = 0; i < NeuronasCapa2; i++) { //Va de neurona en neurona de la
capa 2
                    double Yi = Capas[2].Salidas[i];
                    double Si = SalidaEsperada[i]; //Salida esperada
                    double W2pi = Capas[2].Neuronas[i].Pesos[p];
                    InternoAcumula += W2pi * (Yi - Si) * Yi * (1 - Yi); //Sumatoria interna
                }
                double W1kp = Capas[1].Neuronas[p].Pesos[k];
                double a1p = Capas[1].Salidas[p];
                Acumula += W1kp * a1p * (1 - a1p) * InternoAcumula; //Sumatoria externa
            }
            double xj = Entradas[j];
            double a0k = Capas[0].Salidas[k];
            double dE0 = xj * a0k * (1 - a0k) * Acumula;
        }
}

```

```

        double W0jk = Capas[0].Neuronas[k].Pesos[j];
        Capas[0].Neuronas[k].NuevosPesos[j] = W0jk - Alpha * dE0;
    }

    //Procesa umbrales capa 2
    for (int i = 0; i < NeuronasCapa2; i++) { //Va de neurona en neurona de la capa de salida
(capa 2)
        double Yi = Capas[2].Salidas[i]; //Salida de la neurona de la capa de salida
        double Si = SalidaEsperada[i]; //Salida esperada
        double dE2 = (Yi - Si) * Yi * (1 - Yi);
        Capas[2].Neuronas[i].NuevoUmbbral = Capas[2].Neuronas[i].Umbbral - Alpha * dE2;
    }

    //Procesa umbrales capa 1
    for (int k = 0; k < NeuronasCapa1; k++) { //Va de neurona en neurona de la capa 1
        double Acumula = 0;
        for (int i = 0; i < NeuronasCapa2; i++) { //Va de neurona en neurona de la capa 2
            double Yi = Capas[2].Salidas[i]; //Salida de la capa 2
            double Si = SalidaEsperada[i];
            double W2ki = Capas[2].Neuronas[i].Pesos[k];
            Acumula += W2ki * (Yi - Si) * Yi * (1 - Yi);
        }
        double a1k = Capas[1].Salidas[k];
        double dE1 = a1k * (1 - a1k) * Acumula;
        Capas[1].Neuronas[k].NuevoUmbbral = Capas[1].Neuronas[k].Umbbral - Alpha * dE1;
    }

    //Procesa umbrales capa 0
    for (int k = 0; k < NeuronasCapa0; k++) { //Va de neurona en neurona de la capa 0
        double Acumula = 0;
        for (int p = 0; p < NeuronasCapa1; p++) { //Va de neurona en neurona de la capa 1
            double InternoAcumula = 0;
            for (int i = 0; i < NeuronasCapa2; i++) { //Va de neurona en neurona de la capa 2
                double Yi = Capas[2].Salidas[i];
                double Si = SalidaEsperada[i];
                double W2pi = Capas[2].Neuronas[i].Pesos[p];
                InternoAcumula += W2pi * (Yi - Si) * Yi * (1 - Yi);
            }
            double W1kp = Capas[1].Neuronas[p].Pesos[k];
            double a1p = Capas[1].Salidas[p];
            Acumula += W1kp * a1p * (1 - a1p) * InternoAcumula;
        }
        double a0k = Capas[0].Salidas[k];
        double dE0 = a0k * (1 - a0k) * Acumula;
        Capas[0].Neuronas[k].NuevoUmbbral = Capas[0].Neuronas[k].Umbbral - Alpha * dE0;
    }

    //Actualiza los pesos
    Capas[0].Actualiza();
    Capas[1].Actualiza();
    Capas[2].Actualiza();
}

}

class Capa {
    public List<Neurona> Neuronas; //Las neuronas que tendrá la capa
    public List<double> Salidas; //Almacena las salidas de cada neurona

    public Capa(Random Azar, int TotalNeuronas, int TotalEntradas) {
        Neuronas = [];
        Salidas = [];

        //Genera las neuronas
        for (int Contador = 0; Contador < TotalNeuronas; Contador++) {
            Neuronas.Add(new Neurona(Azar, TotalEntradas));
            Salidas.Add(0);
        }
    }

    //Calcula las salidas de cada neurona de la capa
    public void CalculaCapa(List<double> Entradas) {
        for (int Contador = 0; Contador < Neuronas.Count; Contador++)
            Salidas[Contador] = Neuronas[Contador].CalculaSalida(Entradas);
    }

    //Actualiza los pesos y umbrales de las neuronas
    public void Actualiza() {
        for (int Contador = 0; Contador < Neuronas.Count; Contador++)

```

```

        Neuronas[Contador].Actualiza();
    }

class Neurona {
    public List<double> Pesos; //Los pesos para cada entrada
    public List<double> NuevosPesos; //Nuevos pesos dados por el algoritmo de "backpropagation"
    public double Umbral; //El peso del umbral
    public double NuevoUmbral; //Nuevo umbral dado por el algoritmo de "backpropagation"

    //Inicializa los pesos y umbral con un valor al azar
    public Neurona(Random Azar, int TotalEntradas) {
        Pesos = [];
        NuevosPesos = [];
        for (int Contador = 0; Contador < TotalEntradas; Contador++) {
            Pesos.Add(Azar.NextDouble());
            NuevosPesos.Add(0);
        }
        Umbral = Azar.NextDouble();
        NuevoUmbral = 0;
    }

    //Calcula la salida de la neurona dependiendo de las entradas
    public double CalculaSalida(List<double> Entradas) {
        double Valor = 0;
        for (int Contador = 0; Contador < Pesos.Count; Contador++)
            Valor += Entradas[Contador] * Pesos[Contador];
        Valor += Umbral;
        return 1 / (1 + Math.Exp(-Valor));
    }

    //Reemplaza viejos pesos por nuevos
    public void Actualiza() {
        for (int Contador = 0; Contador < Pesos.Count; Contador++)
            Pesos[Contador] = NuevosPesos[Contador];
        Umbral = NuevoUmbral;
    }
}

```

```

Ciclo: 89800
1 | 1 | Esperada: 0 Calculada: 0 | 0,003702174383143816 |
1 | 0 | Esperada: 1 Calculada: 1 | 0,9953775432424248 |
0 | 1 | Esperada: 1 Calculada: 1 | 0,9953696734810018 |
0 | 0 | Esperada: 0 Calculada: 0 | 0,0031923050449013304 |

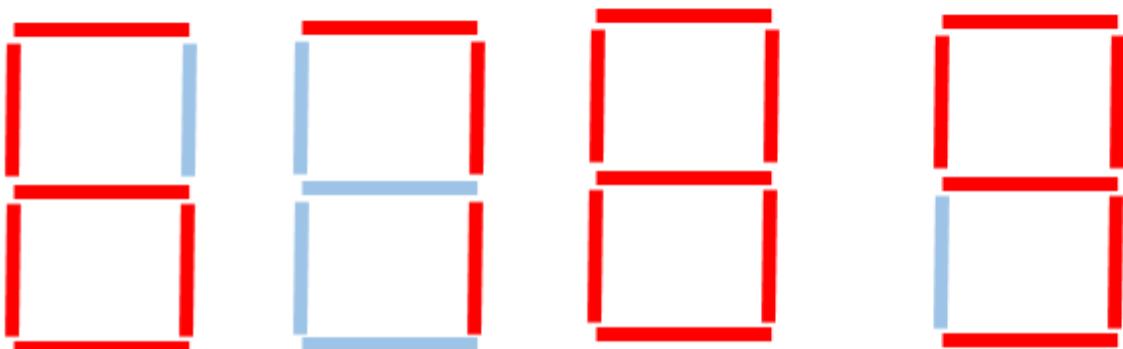
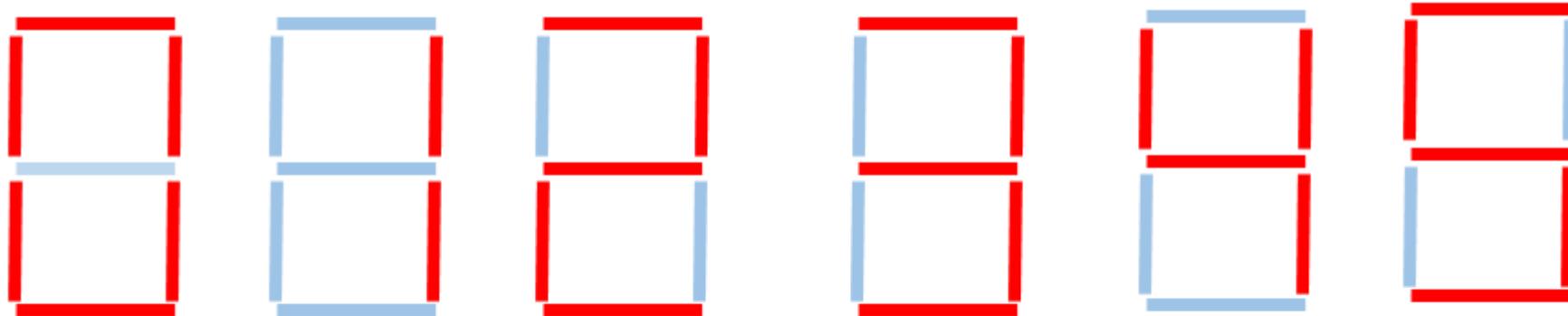
Ciclo: 90000
1 | 1 | Esperada: 0 Calculada: 0 | 0,003695850644007681 |
1 | 0 | Esperada: 1 Calculada: 1 | 0,9953851916858888 |
0 | 1 | Esperada: 1 Calculada: 1 | 0,9953773370654849 |
0 | 0 | Esperada: 0 Calculada: 0 | 0,0031871208730737985 |
Finaliza el entrenamiento

```

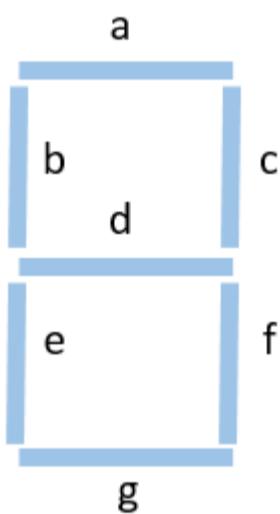
Ilustración 350: Perceptrón aprendiendo la tabla del XOR

Reconocimiento de números de un reloj digital

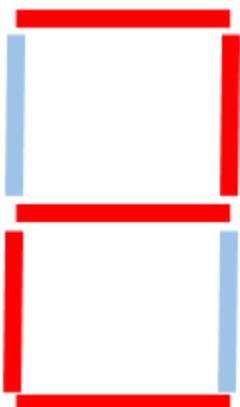
En la imagen, los números del 0 al 9 construidos usando las barras verticales y horizontales. Típicos de un reloj digital.



Se quiere construir una red neuronal tipo perceptrón multicapa que dado ese número al estilo reloj digital se pueda deducir el número como tal. Para iniciar se pone un identificador a cada barra



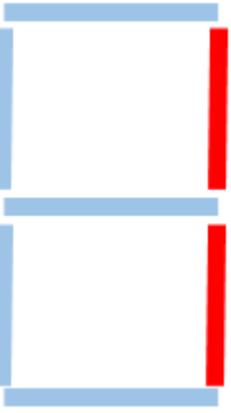
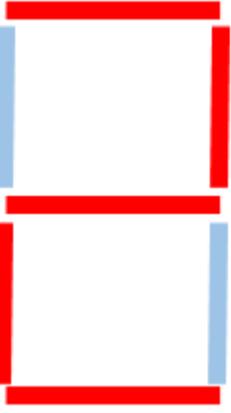
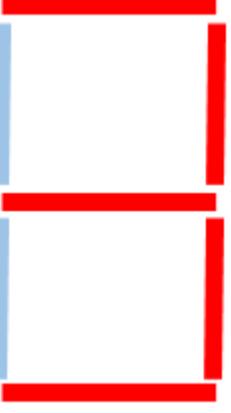
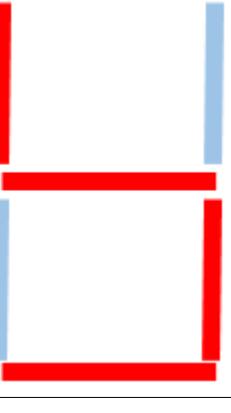
Luego se le da un valor de "1" a la barra que queda en rojo al construir el número y "0" a la barra que queda en azul claro. Por ejemplo:

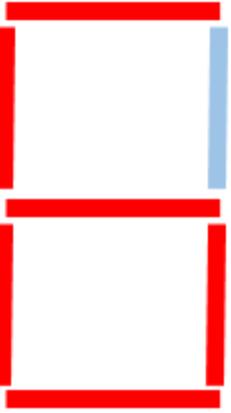


Los valores de a,b,c,d,e,f,g serían: 1,0,1,1,1,0,1

Como se debe deducir que es un 2, este valor se convierte a binario 10_2 o 0,0,1,0 (para convertirlo en salida del perceptrón, como el máximo valor del reloj digital es 9 y este es 1,0,0,1 entonces el número de salidas es 4)

La tabla de entradas y salidas esperadas es:

Imagen	Valor de entrada	Valor de salida esperado
	1,1,1,0,1,1,1	0,0,0,0
	0,0,1,0,0,1,0	0,0,0,1
	1,0,1,1,1,0,1	0,0,1,0
	1,0,1,1,0,1,1	0,0,1,1
	0,1,1,1,0,1,0	0,1,0,0
	1,1,0,1,0,1,1	0,1,0,1

	1,1,0,1,1,1,1	0,1,1,0
	1,0,1,0,0,1,0	0,1,1,1
	1,1,1,1,1,1,1	1,0,0,0
	1,1,1,1,0,1,1	1,0,0,1

El código es el siguiente:

K/016.cs

```
namespace Ejemplo {
    class Program {
        static void Main() {
            //El número "dibujado" en el reloj digital
            int[][] RelojEntra = new int[][] {
                new int[] { 1, 1, 1, 0, 1, 1, 1 },
                new int[] { 0, 0, 1, 0, 0, 1, 0 },
                new int[] { 1, 0, 1, 1, 1, 0, 1 },
                new int[] { 1, 0, 1, 1, 0, 1, 1 },
                new int[] { 0, 1, 1, 1, 0, 1, 0 },
                new int[] { 1, 1, 0, 1, 0, 1, 1 },
                new int[] { 1, 1, 0, 1, 1, 1, 1 },
                new int[] { 1, 0, 1, 0, 0, 1, 0 },
                new int[] { 1, 1, 1, 1, 1, 1, 1 },
                new int[] { 1, 1, 1, 1, 0, 1, 1 }
            };

            //Número esperado en binario
            int[][] NumeroEsperado = new int[][] {
                new int[] { 0, 0, 0, 0 },
                new int[] { 0, 0, 0, 1 },
                new int[] { 0, 0, 1, 0 },
                new int[] { 0, 0, 1, 1 },
                new int[] { 0, 1, 0, 0 },
                new int[] { 0, 1, 0, 1 },
                new int[] { 0, 1, 1, 0 },
                new int[] { 0, 1, 1, 1 },
                new int[] { 1, 0, 0, 0 },
                new int[] { 1, 0, 0, 1 }
            };

            int TotalEntradas = 7; //Número de entradas
            int NeuronasCapa0 = 5; //Total neuronas en la capa 0
            int NeuronasCapa1 = 5; //Total neuronas en la capa 1
            int NeuronasCapa2 = 4; //Total neuronas en la capa 2
            Perceptron RedNeuronal = new(7, 5, 5, 4);

            //Estas serán las entradas externas al perceptrón
            List<double> Entradas = [0, 0, 0, 0, 0, 0, 0];

            //Esta será las salidas esperadas externas al perceptrón
            List<double> SalidaEsperada = [0, 0, 0, 0];

            //Ciclo que entrena la red neuronal
            int TotalCiclos = 10000; //Ciclos de entrenamiento
            for (int Ciclo = 1; Ciclo <= TotalCiclos; Ciclo++) {

                if (Ciclo % 1000 == 0) Console.WriteLine("\r\nCiclo: " + Ciclo);

                //Por cada ciclo, se entrena el perceptrón con toda la tabla
                for (int Conjunto = 0; Conjunto < NumeroEsperado.GetLength(0); Conjunto++) {

                    //Entradas y salidas esperadas
                    for (int Entra = 0; Entra < RelojEntra[0].GetLength(0); Entra++)
                        Entradas[Entra] = RelojEntra[Conjunto][Entra];

                    for (int Sale = 0; Sale < NumeroEsperado[0].GetLength(0); Sale++)
                        SalidaEsperada[Sale] = NumeroEsperado[Conjunto][Sale];

                    //Primero calcula la salida del perceptrón con esas entradas
                    RedNeuronal.CalculaSalida(Entradas);

                    //Luego entrena el perceptrón para ajustar los pesos y umbral
                    RedNeuronal.Entrena(Entradas, SalidaEsperada);

                    //Cada 1000 ciclos muestra como progresa el entrenamiento
                    if (Ciclo % 1000 == 0) RedNeuronal.SalidaPerceptron(Entradas, SalidaEsperada);
                }
            }

            Console.WriteLine("Finaliza el entrenamiento");
        }
    }

    class Perceptron {

```

```

public List<Capa> Capas;

//Imprime los datos de las diferentes capas
public void SalidaPerceptron(List<double> Entradas, List<double> SalidaEsperada) {
    for (int cont = 0; cont < Entradas.Count; cont++) {
        Console.WriteLine(Entradas[cont] + " ");
    }
    Console.WriteLine(" Esperada: ");
    for (int cont = 0; cont < SalidaEsperada.Count; cont++) {
        Console.WriteLine(SalidaEsperada[cont] + " ");
    }
    Console.WriteLine(" Calculada: ");
    for (int cont = 0; cont < Capas[2].Salidas.Count; cont++) {
        if (Capas[2].Salidas[cont] >= 0.5)
            Console.WriteLine("1 ");
        else
            Console.WriteLine("0 ");
    }
    Console.WriteLine(" ");
}

//Crea las diversas capas
public Perceptron(int TotalEntradas, int NeuronasCapa0, int NeuronasCapa1, int NeuronasCapa2) {
    Random Azar = new();
    Capas =
    [
        new Capa(Azar, NeuronasCapa0, TotalEntradas), //Crea la capa 0
        new Capa(Azar, NeuronasCapa1, NeuronasCapa0), //Crea la capa 1
        new Capa(Azar, NeuronasCapa2, NeuronasCapa1), //Crea la capa 2
    ];
}

//Dada las entradas al perceptrón, se calcula la salida de cada capa.
//Con eso se sabrá que salidas se obtienen con los pesos y umbrales actuales.
//Esas salidas son requeridas para el algoritmo de entrenamiento.
public void CalculaSalida(List<double> Entradas) {
    Capas[0].CalculaCapa(Entradas);
    Capas[1].CalculaCapa(Capas[0].Salidas);
    Capas[2].CalculaCapa(Capas[1].Salidas);
}

//Con las salidas previamente calculadas con unas determinadas entradas
//se ejecuta el algoritmo de entrenamiento "Backpropagation"
public void Entrena(List<double> Entradas, List<double> SalidaEsperada) {
    int NeuronasCapa0 = Capas[0].Neuronas.Count;
    int NeuronasCapa1 = Capas[1].Neuronas.Count;
    int NeuronasCapa2 = Capas[2].Neuronas.Count;

    //Factor de aprendizaje
    double Alpha = 0.4;

    //Procesa pesos capa 2
    for (int j = 0; j < NeuronasCapa1; j++) //Va de neurona en neurona de la capa 1
        for (int i = 0; i < NeuronasCapa2; i++) { //Va de neurona en neurona de la capa de
salida (capa 2)
            double Yi = Capas[2].Salidas[i]; //Salida de la neurona de la capa de salida
            double Si = SalidaEsperada[i]; //Salida esperada
            double a1j = Capas[1].Salidas[j]; //Salida de la capa 1
            double dE2 = a1j * (Yi - Si) * Yi * (1 - Yi); //La fórmula del error
            Capas[2].Neuronas[i].NuevosPesos[j] = Capas[2].Neuronas[i].Pesos[j] - Alpha * dE2;
//Ajusta el nuevo peso
        }

    //Procesa pesos capa 1
    for (int j = 0; j < NeuronasCapa0; j++) //Va de neurona en neurona de la capa 0
        for (int k = 0; k < NeuronasCapa1; k++) { //Va de neurona en neurona de la capa 1
            double Acumula = 0;
            for (int i = 0; i < NeuronasCapa2; i++) { //Va de neurona en neurona de la capa 2
                double Yi = Capas[2].Salidas[i]; //Salida de la capa 2
                double Si = SalidaEsperada[i]; //Salida esperada
                double W2ki = Capas[2].Neuronas[i].Pesos[k];
                Acumula += W2ki * (Yi - Si) * Yi * (1 - Yi); //Sumatoria
            }
            double a0j = Capas[0].Salidas[j];
            double a1k = Capas[1].Salidas[k];
            double dE1 = a0j * a1k * (1 - a1k) * Acumula;
            Capas[1].Neuronas[k].NuevosPesos[j] = Capas[1].Neuronas[k].Pesos[j] - Alpha * dE1;
        }
}

```

```

//Procesa pesos capa 0
for (int j = 0; j < Entradas.Count; j++) //Va de entrada en entrada
    for (int k = 0; k < NeuronasCapa0; k++) { //Va de neurona en neurona de la capa 0
        double Acumula = 0;
        for (int p = 0; p < NeuronasCapa1; p++) { //Va de neurona en neurona de la capa 1
            double InternoAcumula = 0;
            for (int i = 0; i < NeuronasCapa2; i++) { //Va de neurona en neurona de la
capa 2
                double Yi = Capas[2].Salidas[i];
                double Si = SalidaEsperada[i]; //Salida esperada
                double W2pi = Capas[2].Neuronas[i].Pesos[p];
                InternoAcumula += W2pi * (Yi - Si) * Yi * (1 - Yi); //Sumatoria interna
            }
            double W1kp = Capas[1].Neuronas[p].Pesos[k];
            double alp = Capas[1].Salidas[p];
            Acumula += W1kp * alp * (1 - alp) * InternoAcumula; //Sumatoria externa
        }
        double xj = Entradas[j];
        double a0k = Capas[0].Salidas[k];
        double dE0 = xj * a0k * (1 - a0k) * Acumula;
        double W0jk = Capas[0].Neuronas[k].Pesos[j];
        Capas[0].Neuronas[k].NuevosPesos[j] = W0jk - Alpha * dE0;
    }
}

//Procesa umbrales capa 2
for (int i = 0; i < NeuronasCapa2; i++) { //Va de neurona en neurona de la capa de salida
(capas 2)
    double Yi = Capas[2].Salidas[i]; //Salida de la neurona de la capa de salida
    double Si = SalidaEsperada[i]; //Salida esperada
    double dE2 = (Yi - Si) * Yi * (1 - Yi);
    Capas[2].Neuronas[i].NuevoUmbbral = Capas[2].Neuronas[i].Umbbral - Alpha * dE2;
}

//Procesa umbrales capa 1
for (int k = 0; k < NeuronasCapa1; k++) { //Va de neurona en neurona de la capa 1
    double Acumula = 0;
    for (int i = 0; i < NeuronasCapa2; i++) { //Va de neurona en neurona de la capa 2
        double Yi = Capas[2].Salidas[i]; //Salida de la capa 2
        double Si = SalidaEsperada[i];
        double W2ki = Capas[2].Neuronas[i].Pesos[k];
        Acumula += W2ki * (Yi - Si) * Yi * (1 - Yi);
    }
    double alk = Capas[1].Salidas[k];
    double dE1 = alk * (1 - alk) * Acumula;
    Capas[1].Neuronas[k].NuevoUmbbral = Capas[1].Neuronas[k].Umbbral - Alpha * dE1;
}

//Procesa umbrales capa 0
for (int k = 0; k < NeuronasCapa0; k++) { //Va de neurona en neurona de la capa 0
    double Acumula = 0;
    for (int p = 0; p < NeuronasCapa1; p++) { //Va de neurona en neurona de la capa 1
        double InternoAcumula = 0;
        for (int i = 0; i < NeuronasCapa2; i++) { //Va de neurona en neurona de la capa 2
            double Yi = Capas[2].Salidas[i];
            double Si = SalidaEsperada[i];
            double W2pi = Capas[2].Neuronas[i].Pesos[p];
            InternoAcumula += W2pi * (Yi - Si) * Yi * (1 - Yi);
        }
        double W1kp = Capas[1].Neuronas[p].Pesos[k];
        double alp = Capas[1].Salidas[p];
        Acumula += W1kp * alp * (1 - alp) * InternoAcumula;
    }
    double a0k = Capas[0].Salidas[k];
    double dE0 = a0k * (1 - a0k) * Acumula;
    Capas[0].Neuronas[k].NuevoUmbbral = Capas[0].Neuronas[k].Umbbral - Alpha * dE0;
}

//Actualiza los pesos
Capas[0].Actualiza();
Capas[1].Actualiza();
Capas[2].Actualiza();
}

class Capa {
    public List<Neurona> Neuronas; //Las neuronas que tendrá la capa
    public List<double> Salidas; //Almacena las salidas de cada neurona
}

```

```

public Capa(Random Azar, int TotalNeuronas, int TotalEntradas) {
    Neuronas = [];
    Salidas = [];

    //Genera las neuronas
    for (int Contador = 0; Contador < TotalNeuronas; Contador++) {
        Neuronas.Add(new Neurona(Azar, TotalEntradas));
        Salidas.Add(0);
    }
}

//Calcula las salidas de cada neurona de la capa
public void CalculaCapa(List<double> Entradas) {
    for (int Contador = 0; Contador < Neuronas.Count; Contador++)
        Salidas[Contador] = Neuronas[Contador].CalculaSalida(Entradas);
}

//Actualiza los pesos y umbrales de las neuronas
public void Actualiza() {
    for (int Contador = 0; Contador < Neuronas.Count; Contador++)
        Neuronas[Contador].Actualiza();
}
}

class Neurona {
    public List<double> Pesos; //Los pesos para cada entrada
    public List<double> NuevosPesos; //Nuevos pesos dados por el algoritmo de "backpropagation"
    public double Umbral; //El peso del umbral
    public double NuevoUmbral; //Nuevo umbral dado por el algoritmo de "backpropagation"

    //Inicializa los pesos y umbral con un valor al azar
    public Neurona(Random Azar, int TotalEntradas) {
        Pesos = [];
        NuevosPesos = [];
        for (int Contador = 0; Contador < TotalEntradas; Contador++) {
            Pesos.Add(Azar.NextDouble());
            NuevosPesos.Add(0);
        }
        Umbral = Azar.NextDouble();
        NuevoUmbral = 0;
    }

    //Calcula la salida de la neurona dependiendo de las entradas
    public double CalculaSalida(List<double> Entradas) {
        double Valor = 0;
        for (int Contador = 0; Contador < Pesos.Count; Contador++)
            Valor += Entradas[Contador] * Pesos[Contador];
        Valor += Umbral;
        return 1 / (1 + Math.Exp(-Valor));
    }

    //Reemplaza viejos pesos por nuevos
    public void Actualiza() {
        for (int Contador = 0; Contador < Pesos.Count; Contador++)
            Pesos[Contador] = NuevosPesos[Contador];
        Umbral = NuevoUmbral;
    }
}
}

```

```
Consola de depuración de Mi... + ▾
Ciclo: 1000
1 1 1 0 1 1 1 Esperada: 0 0 0 0 Calculada: 0 0 0 0
0 0 1 0 0 1 0 Esperada: 0 0 0 1 Calculada: 0 0 0 1
1 0 1 1 1 0 1 Esperada: 0 0 1 0 Calculada: 0 0 1 0
1 0 1 1 0 1 1 Esperada: 0 0 1 1 Calculada: 0 0 1 1
0 1 1 1 0 1 0 Esperada: 0 1 0 0 Calculada: 1 0 0 0
1 1 0 1 0 1 1 Esperada: 0 1 0 1 Calculada: 0 1 0 1
1 1 0 1 1 1 1 Esperada: 0 1 1 0 Calculada: 0 0 1 0
1 0 1 0 0 1 0 Esperada: 0 1 1 1 Calculada: 0 1 1 1
1 1 1 1 1 1 1 Esperada: 1 0 0 0 Calculada: 1 0 0 0
1 1 1 1 0 1 1 Esperada: 1 0 0 1 Calculada: 0 1 0 1
```

Ilustración 351: La red neuronal "va aprendiendo" los patrones para identificar el número digital.

```
Ciclo: 10000
1 1 1 0 1 1 1 Esperada: 0 0 0 0 Calculada: 0 0 0 0
0 0 1 0 0 1 0 Esperada: 0 0 0 1 Calculada: 0 0 0 1
1 0 1 1 1 0 1 Esperada: 0 0 1 0 Calculada: 0 0 1 0
1 0 1 1 0 1 1 Esperada: 0 0 1 1 Calculada: 0 0 1 1
0 1 1 1 0 1 0 Esperada: 0 1 0 0 Calculada: 0 1 0 0
1 1 0 1 0 1 1 Esperada: 0 1 0 1 Calculada: 0 1 0 1
1 1 0 1 1 1 1 Esperada: 0 1 1 0 Calculada: 0 1 1 0
1 0 1 0 0 1 0 Esperada: 0 1 1 1 Calculada: 0 1 1 1
1 1 1 1 1 1 1 Esperada: 1 0 0 0 Calculada: 1 0 0 0
1 1 1 1 0 1 1 Esperada: 1 0 0 1 Calculada: 1 0 0 1
Finaliza el entrenamiento
```

Ilustración 352: La red neuronal "aprendió" los patrones para identificar el número digital.

Detección de patrones en series de tiempo

En los dos ejemplos anteriores, los valores de las entradas externas del perceptrón fueron 0 o 1. Pero no está limitado a eso, las entradas externas pueden tener valores entre 0 y 1 (incluyendo el 0 y el 1) por ejemplo: 0.7321, 0.21896, 0.9173418

El problema que se plantea es dado el comportamiento de un evento en el tiempo, ¿podrá la red neuronal deducir el patrón?

Ejemplo: Se tiene esta tabla

X	Y
0	0
5	0.43577871
10	1.73648178
15	3.88228568
20	6.84040287
25	10.5654565
30	15
35	20.0751753
40	25.7115044
45	31.8198052
50	38.3022222
55	45.0533624
60	51.9615242
65	58.9100062
70	65.7784835
75	72.444437
80	78.7846202
85	84.6765493
90	90
95	94.6384963
100	98.4807753

X es la variable independiente mientras Y es la variable dependiente, en otras palabras $Y=F(X)$. El problema es que no se sabe $F()$, sólo están los datos (por motivos prácticos se muestra la tabla con X llegando hasta 100, realmente llega hasta 1800). Esta sería la gráfica.

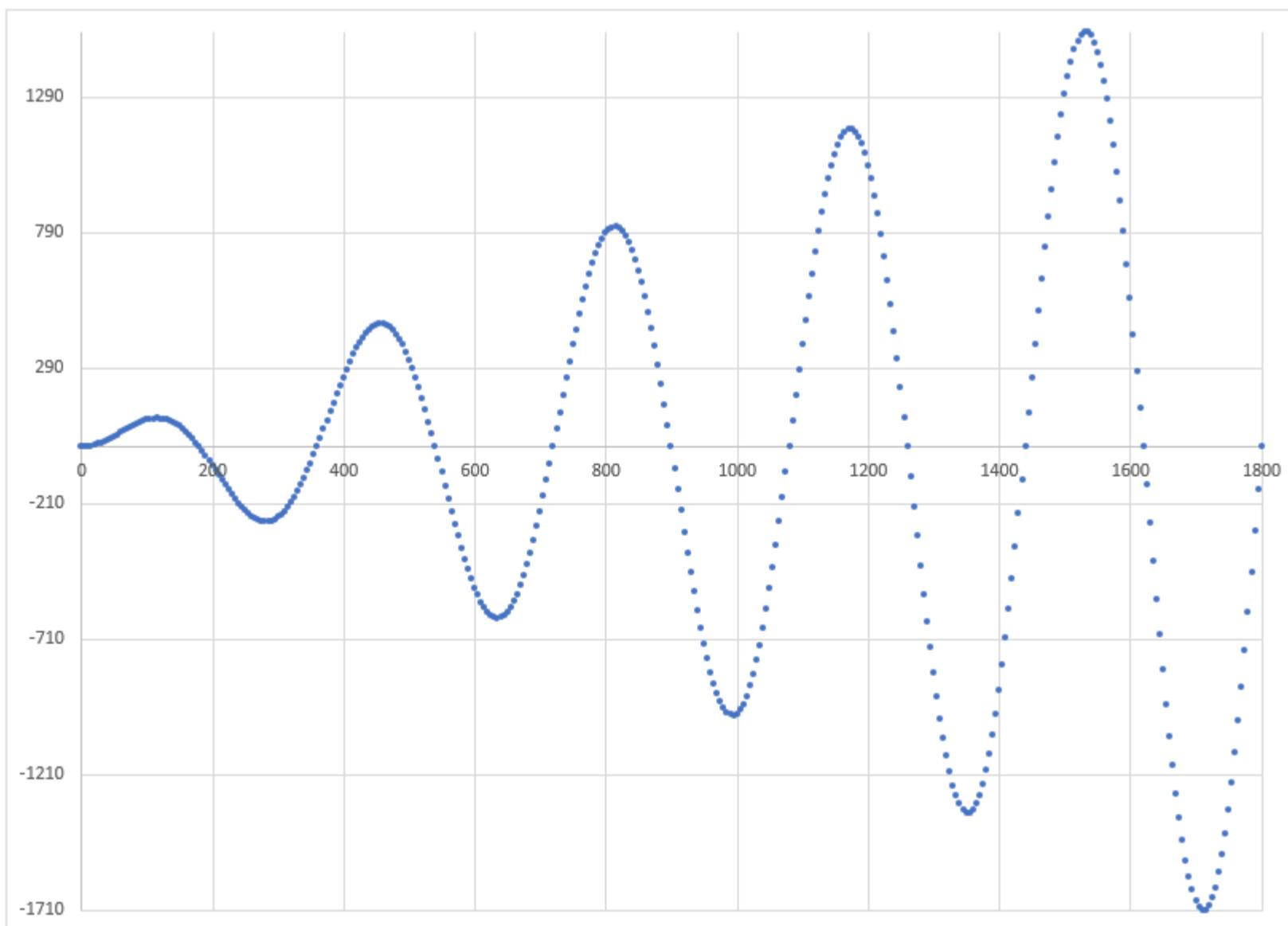


Ilustración 353: Gráfico generado por los puntos

Uniendo los puntos, se obtendría

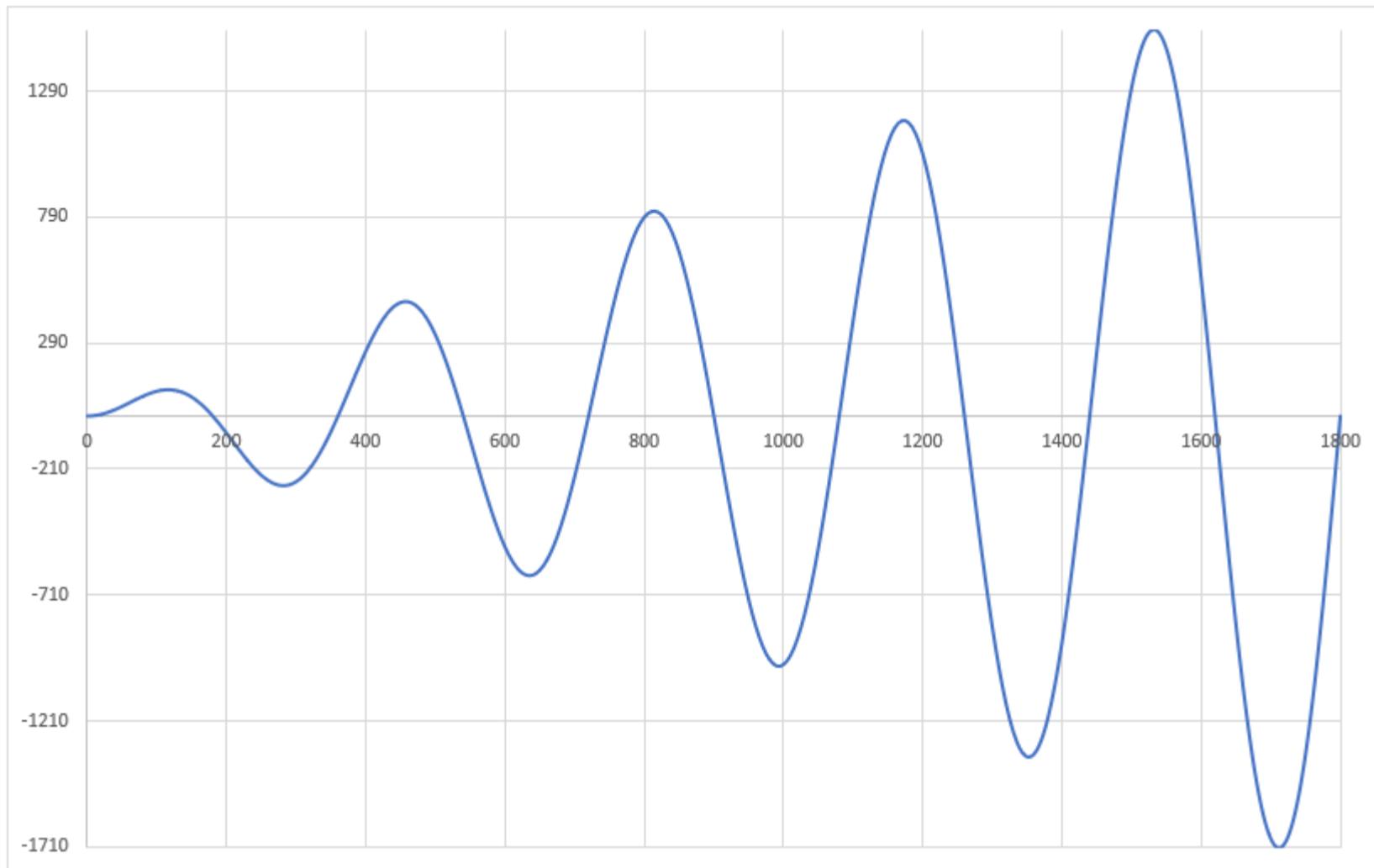


Ilustración 354: Gráfico uniendo los puntos

El primer paso es convertir los valores de X y de Y que dan origen al gráfico, en valores entre 0 y 1 porque la función de activación del perceptrón multicapa $y = \frac{1}{1+e^{-x}}$ solo genera números entre 0 y 1.

Se hace entonces una normalización usando la siguiente fórmula

$$X_{\text{normalizado}} = \frac{X_{\text{original}} - \text{Minimo}X}{\text{Maximo}X - \text{Minimo}X}$$

$$Y_{\text{normalizado}} = \frac{Y_{\text{original}} - \text{Minimo}Y}{\text{Maximo}Y - \text{Minimo}Y}$$

Como indica la fórmula, habría que recorrer todos los datos de X y Y para deducir el mayor valor de X, menor valor de X, mayor valor de Y y menor valor de Y.

357	1775	-750,147415	355	1765	-1012,36241	354	1760	-1131,30619	358	1780	-608,795855
358	1780	-608,795855	356	1770	-885	355	1765	-1012,36241	359	1785	-461,991996
359	1785	-461,991996	357	1775	-750,147415	356	1770	-885	360	1790	-310,830238
360	1790	-310,830238	358	1780	-608,795855	357	1775	-750,147415	361	1795	-156,444558
361	1795	-156,444558	359	1785	-461,991996	358	1780	-608,795855	362	1800	-2,2053E-12
362	1800	-2,2053E-12	360	1790	-310,830238	359	1785	-461,991996	363		
363			361	1795	-156,444558	360	1790	-310,830238	364	Minimo X	Minimo Y
364	Minimo X	Minimo Y	362	1800	-2,2053E-12	361	1795	-156,444558	365	0	-1710
365	0	-1710	363			362	1800	-2,2053E-12	366		
366			364	Minimo X	Minimo Y	363			367	Maximo X	Maximo Y
367	Maximo X	Maximo Y	365	=MIN(A2:A362)		364	Minimo X	Minimo Y	368	1800	=MAX(B2:B362)
368	1800	1530	366			365	0	-1710	369		
369			367	Maximo X	Maximo Y	366					
370			368	1800	1530	367	Maximo X	Maximo Y			
			369			368	=MAX(A2:A362)				

Se aplica la fórmula

C2	A	B	C	D
1	Xreal	Yreal	Xnormalizado	Ynormalizado
2	0	0	0	0,527777778
3	5	0,4357787	0,002777778	0,527912277
4	10	1,7364818	0,005555556	0,528313729
5	15	3,8822857	0,008333333	0,528976014
6	20	6,8404029	0,011111111	0,529889013
7	25	10,565457	0,013888889	0,531038721

SUMA		<input type="button" value="X"/>	<input type="button" value="✓"/>	<input type="button" value="fx"/>	$=(B2-\$B\$365)/(\$B\$368-\$B\$365)$
	A	B	C	D	
1	Xreal	Yreal	Xnormalizado	Ynormalizado	
2	0	0		0	$=(B2-\$B\$365)/(\$B\$368-\$B\$365)$
3	5	0,4357787	0,002777778	0,527912277	
4	10	1,7364818	0,005555556	0,528313729	
5	15	3,8822857	0,008333333	0,528976014	
6	20	6,8404029	0,011111111	0,529889013	
7	25	10,565457	0,013888889	0,531038721	

Se realiza la gráfica con los valores normalizados (datos entre 0 y 1 tanto en X como en Y)

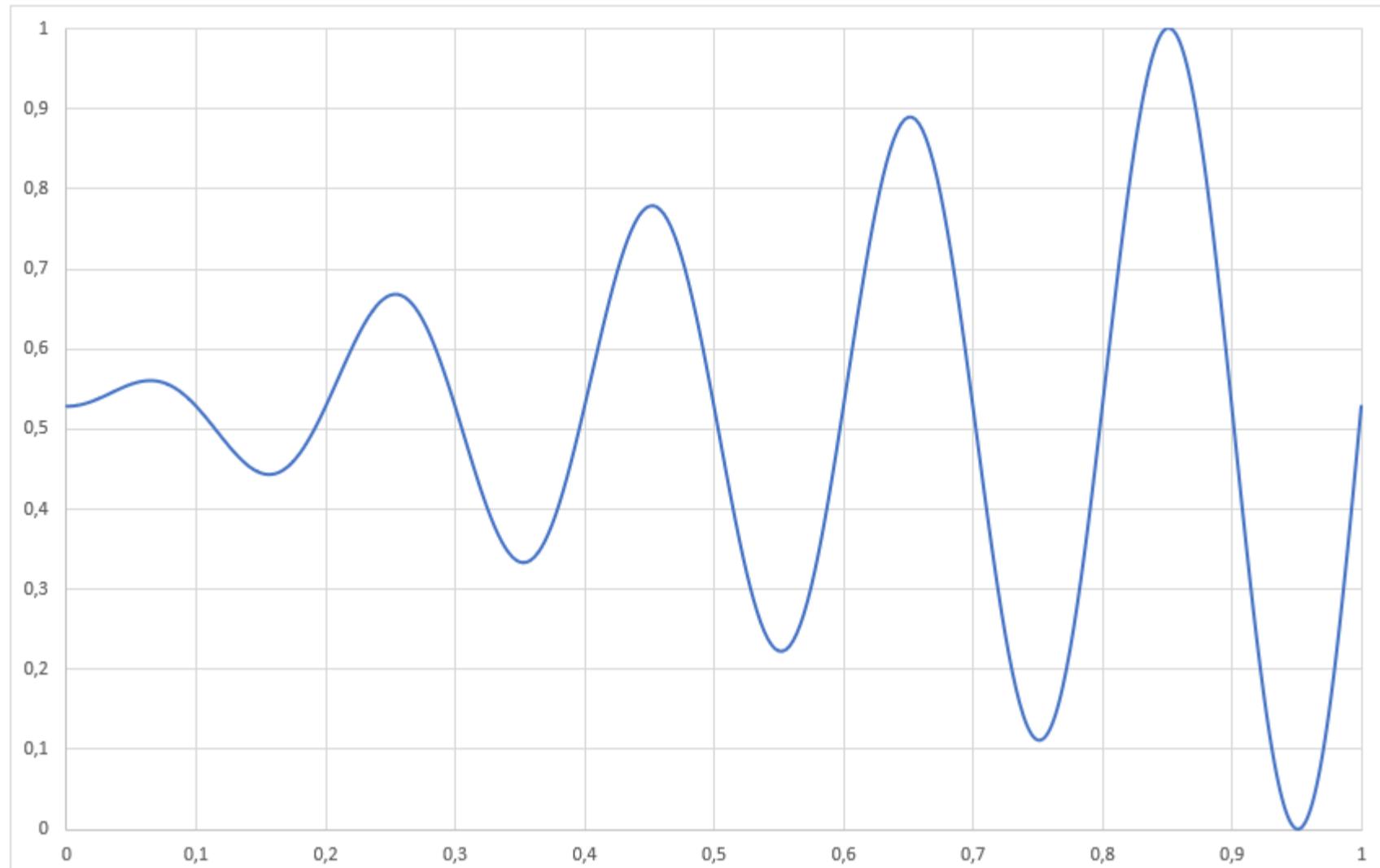


Ilustración 355: Gráfico al normalizar los datos

Con esos datos normalizados, se alimenta la red neuronal para entrenarla. Este sería el código en C#:

K/017.cs

```
namespace Ejemplo {
    class Program {
        static void Main() {
            Random Azar = new();

            //Genera una ecuación al azar para generar el dataset
            double CoefA, CoefB, CoefC, CoefD, CoefE, CoeffF, CoefG, CoefH, CoefI;

            double CoefMinimo = -10;
            double CoefMaximo = 10;
            CoefA = Azar.NextDouble() * (CoefMaximo - CoefMinimo) + CoefMinimo;
            CoefB = Azar.NextDouble() * (CoefMaximo - CoefMinimo) + CoefMinimo;
            CoefC = Azar.NextDouble() * (CoefMaximo - CoefMinimo) + CoefMinimo;
            CoefD = Azar.NextDouble() * (CoefMaximo - CoefMinimo) + CoefMinimo;
            CoefE = Azar.NextDouble() * (CoefMaximo - CoefMinimo) + CoefMinimo;
            CoeffF = Azar.NextDouble() * (CoefMaximo - CoefMinimo) + CoefMinimo;
            CoefG = Azar.NextDouble() * (CoefMaximo - CoefMinimo) + CoefMinimo;
            CoefH = Azar.NextDouble() * (CoefMaximo - CoefMinimo) + CoefMinimo;
            CoefI = Azar.NextDouble() * (CoefMaximo - CoefMinimo) + CoefMinimo;

            List<double> Xentra = new List<double>();
            List<double> Ysale = new List<double>();
```

```

//Genera el dataset con esta ecuación
for (double X = -360; X <= 360; X++) {
    double Y = CoefA * Math.Sin((CoefB * X + CoefC) * Math.PI / 180);
    Y += CoefD * Math.Sin((CoefE * X + CoefF) * Math.PI / 180);
    Y += CoefG * Math.Sin((CoefH * X + CoefI) * Math.PI / 180);

    Xentra.Add(X);
    Ysale.Add(Y);
}

//Con los datos de entrada y salida generados (el dataset), ahora procede a normalizarlos
double MinX = double.MaxValue;
double MaxX = double.MinValue;

double MinY = double.MaxValue;
double MaxY = double.MinValue;

for (int Datos = 0; Datos < Ysale.Count; Datos++) {
    if (Xentra[Datos] < MinX) MinX = Xentra[Datos];
    if (Xentra[Datos] > MaxX) MaxX = Xentra[Datos];
    if (Ysale[Datos] < MinY) MinY = Ysale[Datos];
    if (Ysale[Datos] > MaxY) MaxY = Ysale[Datos];
}

for (int Datos = 0; Datos < Ysale.Count; Datos++) {
    Xentra[Datos] = (Xentra[Datos] - MinX) / (MaxX - MinX);
    Ysale[Datos] = (Ysale[Datos] - MinY) / (MaxY - MinY);
}

//Con los datos normalizados, ahora se entrena a la red neuronal para que detecte el patrón
int TotalEntradas = 1; //Número de entradas
int NeuronasCapa0 = 5; //Total neuronas en la capa 0
int NeuronasCapa1 = 5; //Total neuronas en la capa 1
int NeuronasCapa2 = 1; //Total neuronas en la capa 2
Perceptron RedNeuronal = new(TotalEntradas, NeuronasCapa0, NeuronasCapa1, NeuronasCapa2);

//Esta será la única entrada externa al perceptrón, es decir, X
List<double> Entrada = [0];

//Esta será la salida esperada externa al perceptrón, es decir, Y
List<double> SalidaEsperada = [0];

//Ciclo que entrena la red neuronal
int TotalCiclos = 10000; //Ciclos de entrenamiento
for (int Ciclo = 1; Ciclo <= TotalCiclos; Ciclo++) {

    if (Ciclo % 5000 == 0) Console.WriteLine("\r\nCiclo: " + Ciclo);

    //Por cada ciclo, se entrena el perceptrón con toda la tabla
    for (int Conjunto = 0; Conjunto < Xentra.Count; Conjunto++) {

        //Entrada y salida esperadas
        Entrada[0] = Xentra[Conjunto];
        SalidaEsperada[0] = Ysale[Conjunto];

        //Primero calcula la salida del perceptrón con esa entrada
        RedNeuronal.CalculaSalida(Entrada);

        //Luego entrena el perceptrón para ajustar los pesos y umbrales
        RedNeuronal.Entrena(Entrada, SalidaEsperada);
    }
}

Console.WriteLine("Entrada normalizada | Salida esperada normalizada | Salida perceptrón
normalizada");
for (int Conjunto = 0; Conjunto < Xentra.Count; Conjunto++) {
    //Entradas y salidas esperadas
    Entrada[0] = Xentra[Conjunto];
    SalidaEsperada[0] = Ysale[Conjunto];

    //Calcula la salida del perceptrón con esas entradas
    RedNeuronal.CalculaSalida(Entrada);

    //Muestra la salida
    RedNeuronal.SalidaPerceptron(Entrada, SalidaEsperada);
}
Console.WriteLine("Finaliza el entrenamiento");
}
}

```

```

class Perceptron {
    public List<Capa> Capas;

    //Imprime los datos de las diferentes capas
    public void SalidaPerceptron(List<double> entradas, List<double> salidaesperada) {
        for (int cont = 0; cont < entradas.Count; cont++) {
            Console.WriteLine(entradas[cont].ToString() + " ");
        }
        Console.WriteLine(" | ");
        for (int cont = 0; cont < salidaesperada.Count; cont++) {
            Console.WriteLine(salidaesperada[cont].ToString() + " ");
        }
        Console.WriteLine(" | ");
        for (int cont = 0; cont < Capas[2].Salidas.Count; cont++) {
            Console.WriteLine(Capas[2].Salidas[cont].ToString());
        }
        Console.WriteLine(" ");
    }

    //Crea las diversas capas
    public Perceptron(int TotalEntradas, int NeuronasCapa0, int NeuronasCapa1, int NeuronasCapa2) {
        Random Azar = new();
        Capas =
        [
            new Capa(Azar, NeuronasCapa0, TotalEntradas), //Crea la capa 0
            new Capa(Azar, NeuronasCapa1, NeuronasCapa0), //Crea la capa 1
            new Capa(Azar, NeuronasCapa2, NeuronasCapa1), //Crea la capa 2
        ];
    }

    //Dada las entradas al perceptrón, se calcula la salida de cada capa.
    //Con eso se sabrá que salidas se obtienen con los pesos y umbrales actuales.
    //Esas salidas son requeridas para el algoritmo de entrenamiento.
    public void CalculaSalida(List<double> Entradas) {
        Capas[0].CalculaCapa(Entradas);
        Capas[1].CalculaCapa(Capas[0].Salidas);
        Capas[2].CalculaCapa(Capas[1].Salidas);
    }

    //Con las salidas previamente calculadas con unas determinadas entradas
    //se ejecuta el algoritmo de entrenamiento "Backpropagation"
    public void Entrena(List<double> Entradas, List<double> SalidaEsperada) {
        int NeuronasCapa0 = Capas[0].Neuronas.Count;
        int NeuronasCapa1 = Capas[1].Neuronas.Count;
        int NeuronasCapa2 = Capas[2].Neuronas.Count;

        //Factor de aprendizaje
        double Alpha = 0.4;

        //Procesa pesos capa 2
        for (int j = 0; j < NeuronasCapa1; j++) //Va de neurona en neurona de la capa 1
            for (int i = 0; i < NeuronasCapa2; i++) { //Va de neurona en neurona de la capa de
salida (capa 2)
                double Yi = Capas[2].Salidas[i]; //Salida de la neurona de la capa de salida
                double Si = SalidaEsperada[i]; //Salida esperada
                double a1j = Capas[1].Salidas[j]; //Salida de la capa 1
                double dE2 = a1j * (Yi - Si) * Yi * (1 - Yi); //La fórmula del error
                Capas[2].Neuronas[i].NuevosPesos[j] = Capas[2].Neuronas[i].Pesos[j] - Alpha * dE2;
//Ajusta el nuevo peso
            }

        //Procesa pesos capa 1
        for (int j = 0; j < NeuronasCapa0; j++) //Va de neurona en neurona de la capa 0
            for (int k = 0; k < NeuronasCapa1; k++) { //Va de neurona en neurona de la capa 1
                double Acumula = 0;
                for (int i = 0; i < NeuronasCapa2; i++) { //Va de neurona en neurona de la capa 2
                    double Yi = Capas[2].Salidas[i]; //Salida de la capa 2
                    double Si = SalidaEsperada[i]; //Salida esperada
                    double W2ki = Capas[2].Neuronas[i].Pesos[k];
                    Acumula += W2ki * (Yi - Si) * Yi * (1 - Yi); //Sumatoria
                }
                double a0j = Capas[0].Salidas[j];
                double a1k = Capas[1].Salidas[k];
                double dE1 = a0j * a1k * (1 - a1k) * Acumula;
                Capas[1].Neuronas[k].NuevosPesos[j] = Capas[1].Neuronas[k].Pesos[j] - Alpha * dE1;
            }

        //Procesa pesos capa 0
        for (int j = 0; j < Entradas.Count; j++) //Va de entrada en entrada
    }
}

```

```

        for (int k = 0; k < NeuronasCapa0; k++) { //Va de neurona en neurona de la capa 0
            double Acumula = 0;
            for (int p = 0; p < NeuronasCapa1; p++) { //Va de neurona en neurona de la capa 1
                double InternoAcumula = 0;
                for (int i = 0; i < NeuronasCapa2; i++) { //Va de neurona en neurona de la
capa 2
                    double Yi = Capas[2].Salidas[i];
                    double Si = SalidaEsperada[i]; //Salida esperada
                    double W2pi = Capas[2].Neuronas[i].Pesos[p];
                    InternoAcumula += W2pi * (Yi - Si) * Yi * (1 - Yi); //Sumatoria interna
                }
                double W1kp = Capas[1].Neuronas[p].Pesos[k];
                double alp = Capas[1].Salidas[p];
                Acumula += W1kp * alp * (1 - alp) * InternoAcumula; //Sumatoria externa
            }
            double xj = Entradas[j];
            double a0k = Capas[0].Salidas[k];
            double dE0 = xj * a0k * (1 - a0k) * Acumula;
            double W0jk = Capas[0].Neuronas[k].Pesos[j];
            Capas[0].Neuronas[k].NuevosPesos[j] = W0jk - Alpha * dE0;
        }

        //Procesa umbrales capa 2
        for (int i = 0; i < NeuronasCapa2; i++) { //Va de neurona en neurona de la capa de salida
(capta 2)
            double Yi = Capas[2].Salidas[i]; //Salida de la neurona de la capa de salida
            double Si = SalidaEsperada[i]; //Salida esperada
            double dE2 = (Yi - Si) * Yi * (1 - Yi);
            Capas[2].Neuronas[i].NuevoUmbral = Capas[2].Neuronas[i].Umbral - Alpha * dE2;
        }

        //Procesa umbrales capa 1
        for (int k = 0; k < NeuronasCapa1; k++) { //Va de neurona en neurona de la capa 1
            double Acumula = 0;
            for (int i = 0; i < NeuronasCapa2; i++) { //Va de neurona en neurona de la capa 2
                double Yi = Capas[2].Salidas[i]; //Salida de la capa 2
                double Si = SalidaEsperada[i];
                double W2ki = Capas[2].Neuronas[i].Pesos[k];
                Acumula += W2ki * (Yi - Si) * Yi * (1 - Yi);
            }
            double a1k = Capas[1].Salidas[k];
            double dE1 = a1k * (1 - a1k) * Acumula;
            Capas[1].Neuronas[k].NuevoUmbral = Capas[1].Neuronas[k].Umbral - Alpha * dE1;
        }

        //Procesa umbrales capa 0
        for (int k = 0; k < NeuronasCapa0; k++) { //Va de neurona en neurona de la capa 0
            double Acumula = 0;
            for (int p = 0; p < NeuronasCapa1; p++) { //Va de neurona en neurona de la capa 1
                double InternoAcumula = 0;
                for (int i = 0; i < NeuronasCapa2; i++) { //Va de neurona en neurona de la capa 2
                    double Yi = Capas[2].Salidas[i];
                    double Si = SalidaEsperada[i];
                    double W2pi = Capas[2].Neuronas[i].Pesos[p];
                    InternoAcumula += W2pi * (Yi - Si) * Yi * (1 - Yi);
                }
                double W1kp = Capas[1].Neuronas[p].Pesos[k];
                double alp = Capas[1].Salidas[p];
                Acumula += W1kp * alp * (1 - alp) * InternoAcumula;
            }
            double a0k = Capas[0].Salidas[k];
            double dE0 = a0k * (1 - a0k) * Acumula;
            Capas[0].Neuronas[k].NuevoUmbral = Capas[0].Neuronas[k].Umbral - Alpha * dE0;
        }

        //Actualiza los pesos
        Capas[0].Actualiza();
        Capas[1].Actualiza();
        Capas[2].Actualiza();
    }

}

class Capa {
    public List<Neurona> Neuronas; //Las neuronas que tendrá la capa
    public List<double> Salidas; //Almacena las salidas de cada neurona

    public Capa(Random Azar, int TotalNeuronas, int TotalEntradas) {
        Neuronas = [];

```

```

    Salidas = [];

    //Genera las neuronas
    for (int Contador = 0; Contador < TotalNeuronas; Contador++) {
        Neuronas.Add(new Neurona(Azar, TotalEntradas));
        Salidas.Add(0);
    }

    //Calcula las salidas de cada neurona de la capa
    public void CalculaCapa(List<double> Entradas) {
        for (int Contador = 0; Contador < Neuronas.Count; Contador++)
            Salidas[Contador] = Neuronas[Contador].CalculaSalida(Entradas);
    }

    //Actualiza los pesos y umbrales de las neuronas
    public void Actualiza() {
        for (int Contador = 0; Contador < Neuronas.Count; Contador++)
            Neuronas[Contador].Actualiza();
    }
}

class Neurona {
    public List<double> Pesos; //Los pesos para cada entrada
    public List<double> NuevosPesos; //Nuevos pesos dados por el algoritmo de "backpropagation"
    public double Umbral; //El peso del umbral
    public double NuevoUmbral; //Nuevo umbral dado por el algoritmo de "backpropagation"

    //Inicializa los pesos y umbral con un valor al azar
    public Neurona(Random Azar, int TotalEntradas) {
        Pesos = [];
        NuevosPesos = [];
        for (int Contador = 0; Contador < TotalEntradas; Contador++) {
            Pesos.Add(Azar.NextDouble());
            NuevosPesos.Add(0);
        }
        Umbral = Azar.NextDouble();
        NuevoUmbral = 0;
    }

    //Calcula la salida de la neurona dependiendo de las entradas
    public double CalculaSalida(List<double> Entradas) {
        double Valor = 0;
        for (int Contador = 0; Contador < Pesos.Count; Contador++)
            Valor += Entradas[Contador] * Pesos[Contador];
        Valor += Umbral;
        return 1 / (1 + Math.Exp(-Valor));
    }

    //Reemplaza viejos pesos por nuevos
    public void Actualiza() {
        for (int Contador = 0; Contador < Pesos.Count; Contador++)
            Pesos[Contador] = NuevosPesos[Contador];
        Umbral = NuevoUmbral;
    }
}

```

Un ejemplo de cómo la red neuronal se adapta a la serie:

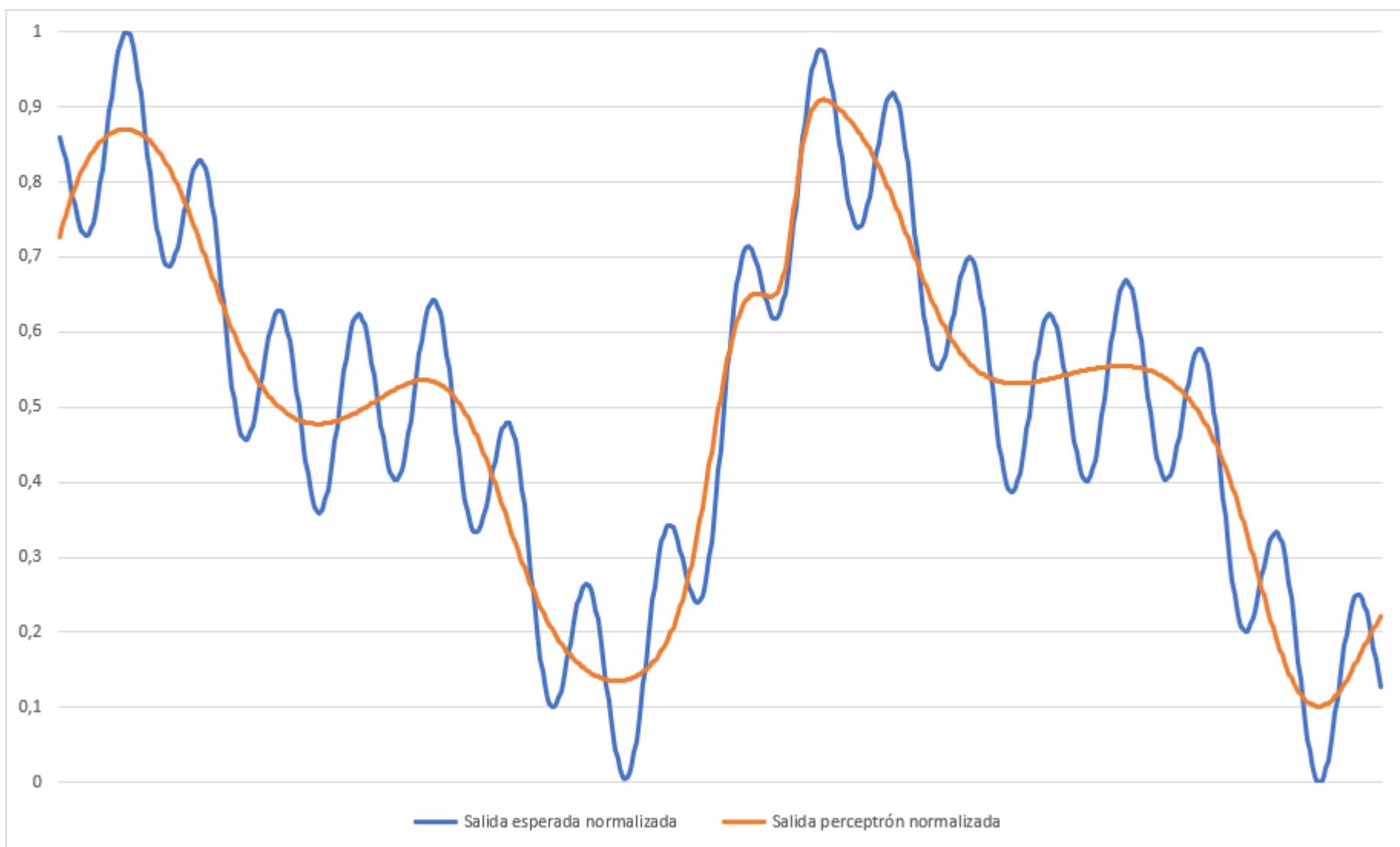


Ilustración 356: Red neuronal adaptándose a una serie temporal

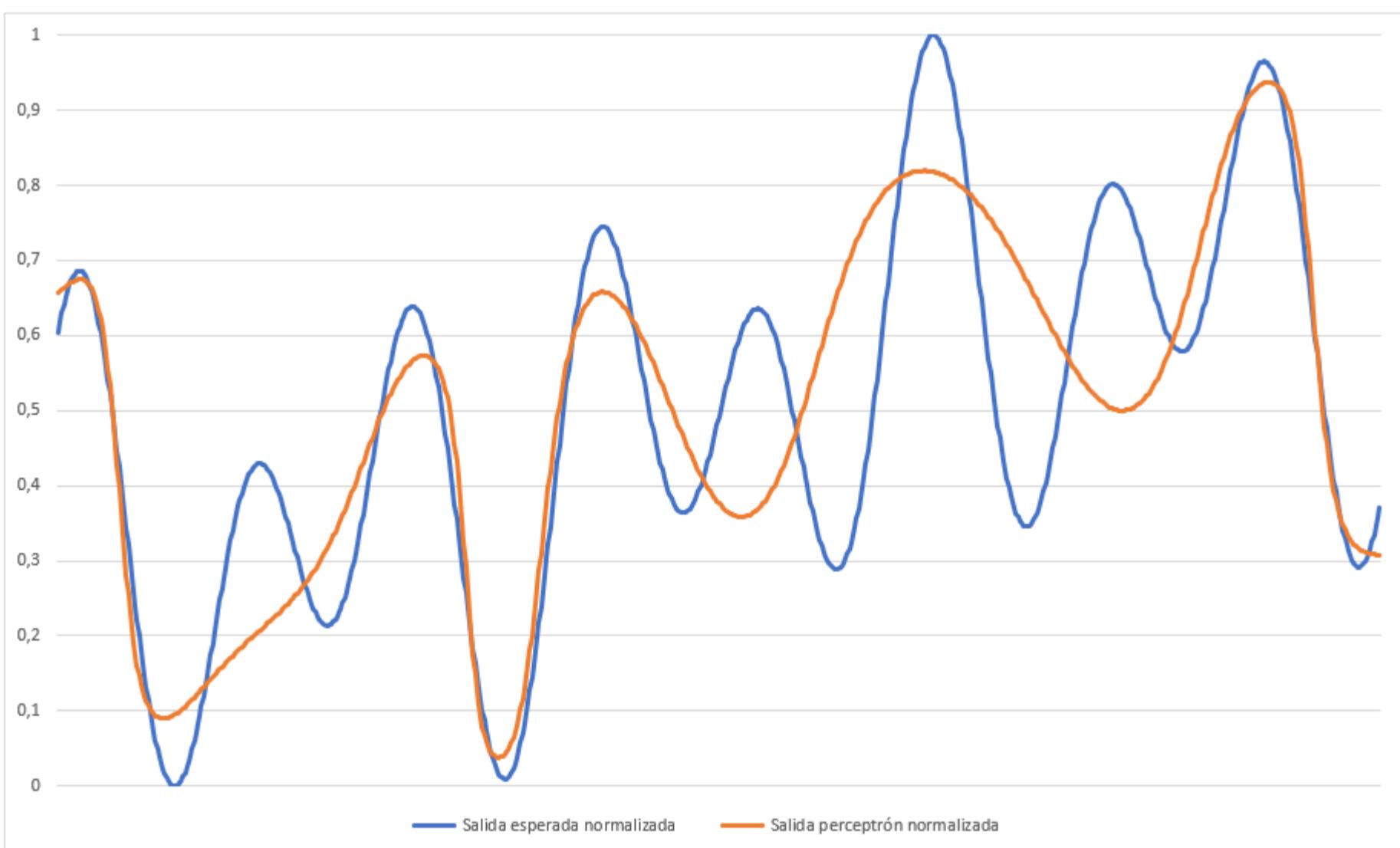


Ilustración 357: Red neuronal adaptándose a una serie temporal

Parte 12: Gráficos en C#

Para trabajar con gráficos, se debe crear un proyecto de escritorio gráfico

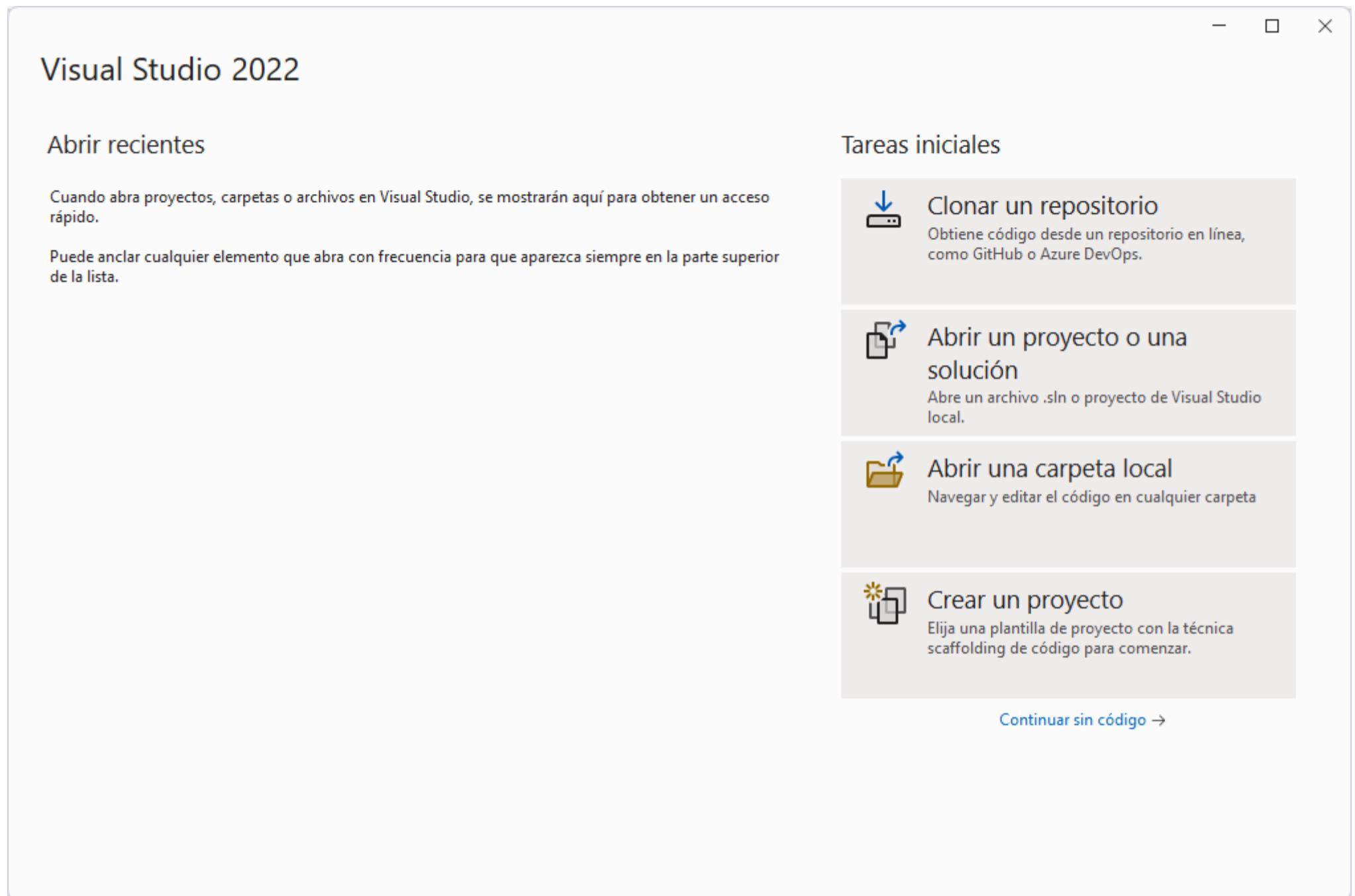


Ilustración 358: Inicia proyecto en Visual Studio 2022

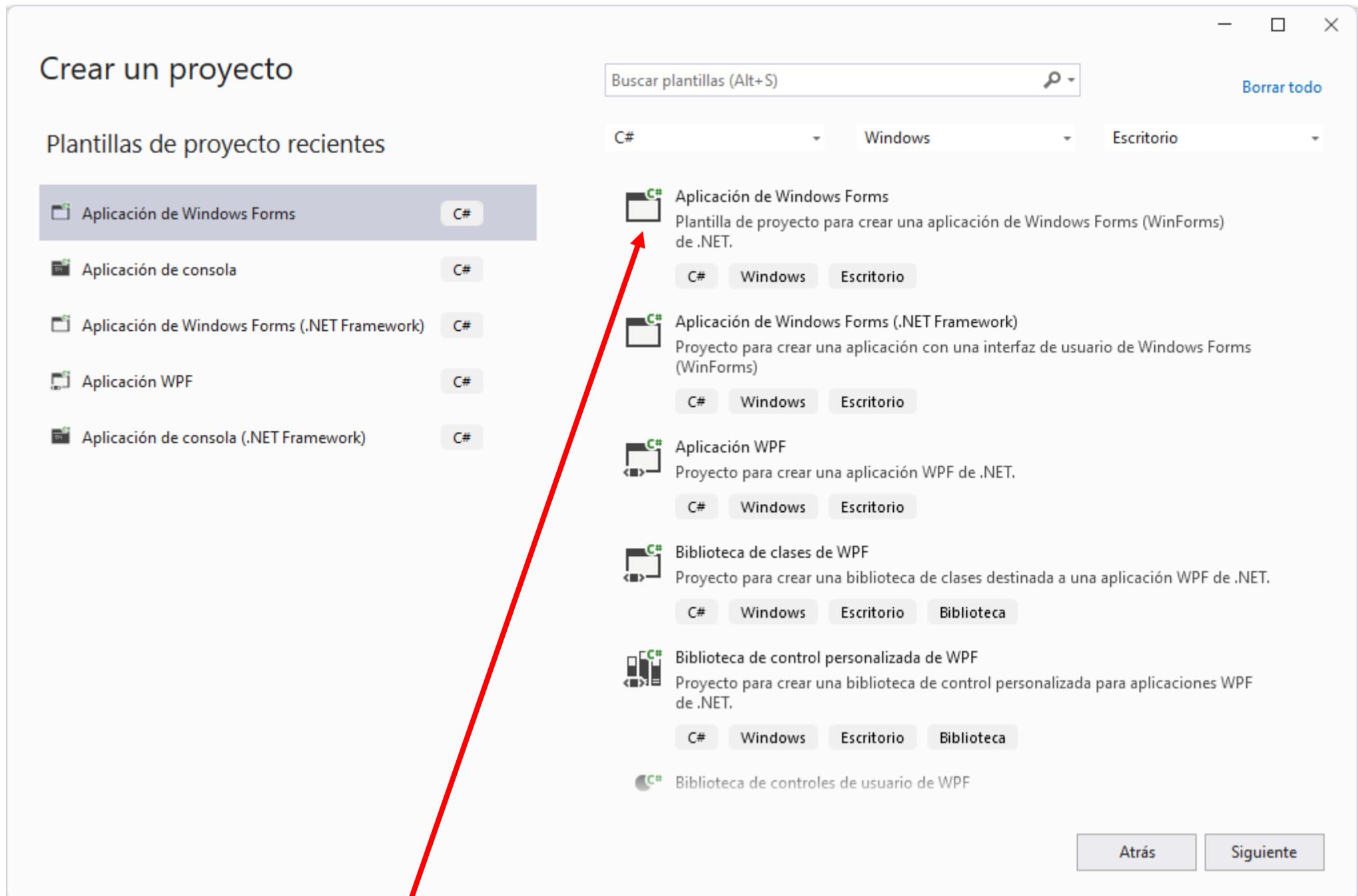


Ilustración 359: Selecciona "Aplicación de Windows Forms"

Una aplicación de tipo Windows Forms

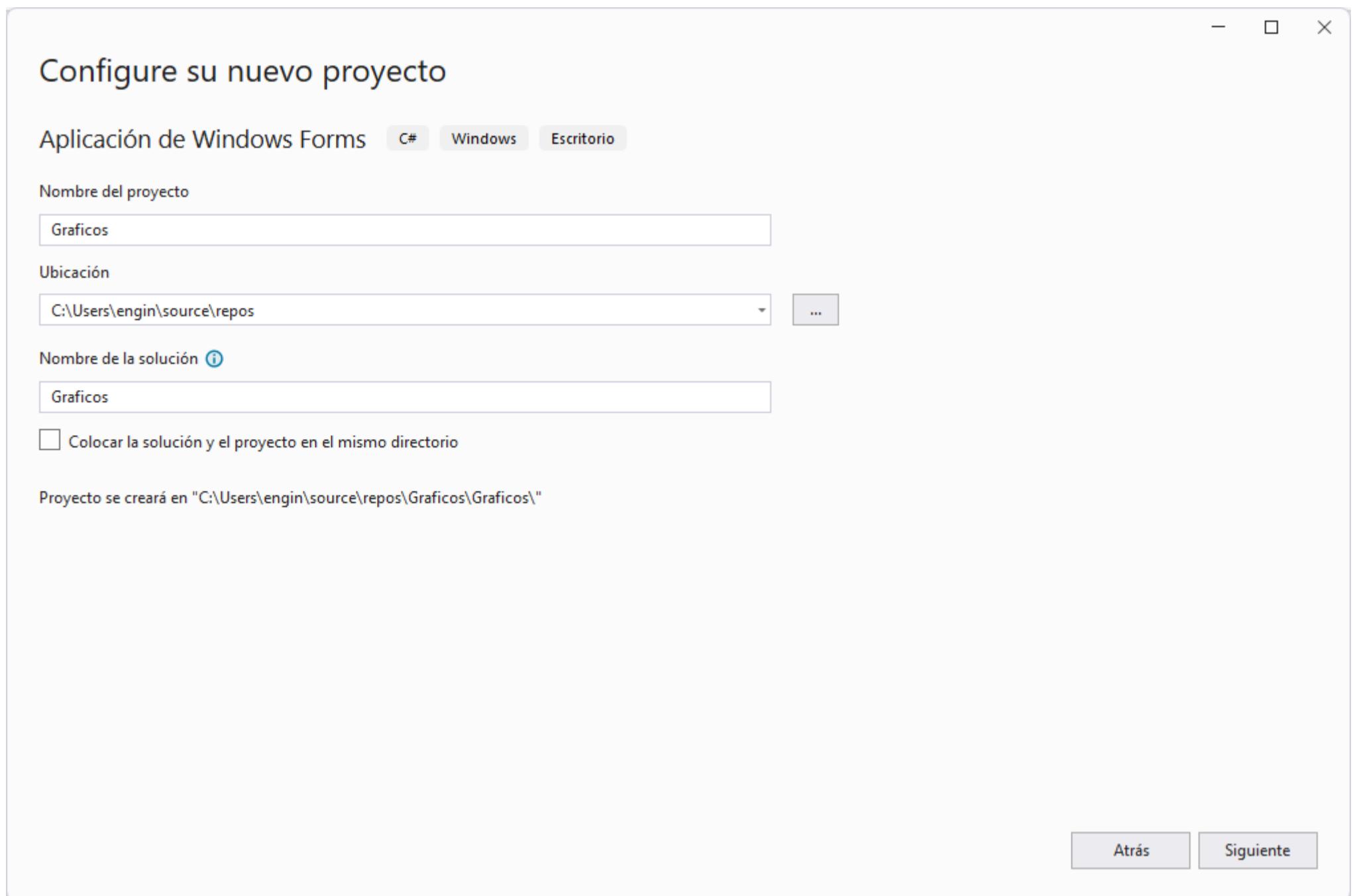


Ilustración 360: Pone nombre al proyecto, para este libro es "Graficos" (sin tilde)

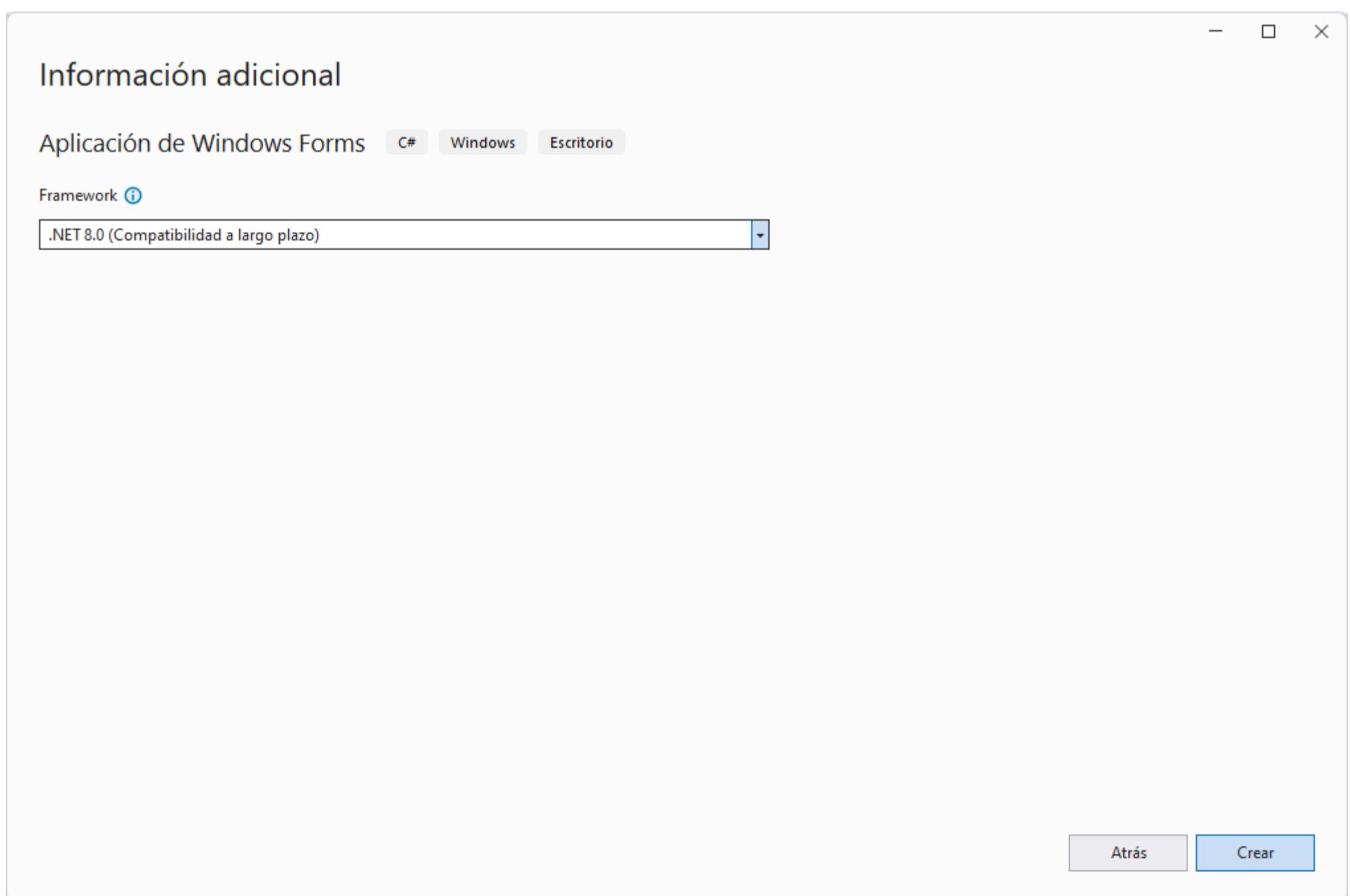


Ilustración 361: Selecciona .NET 8.0

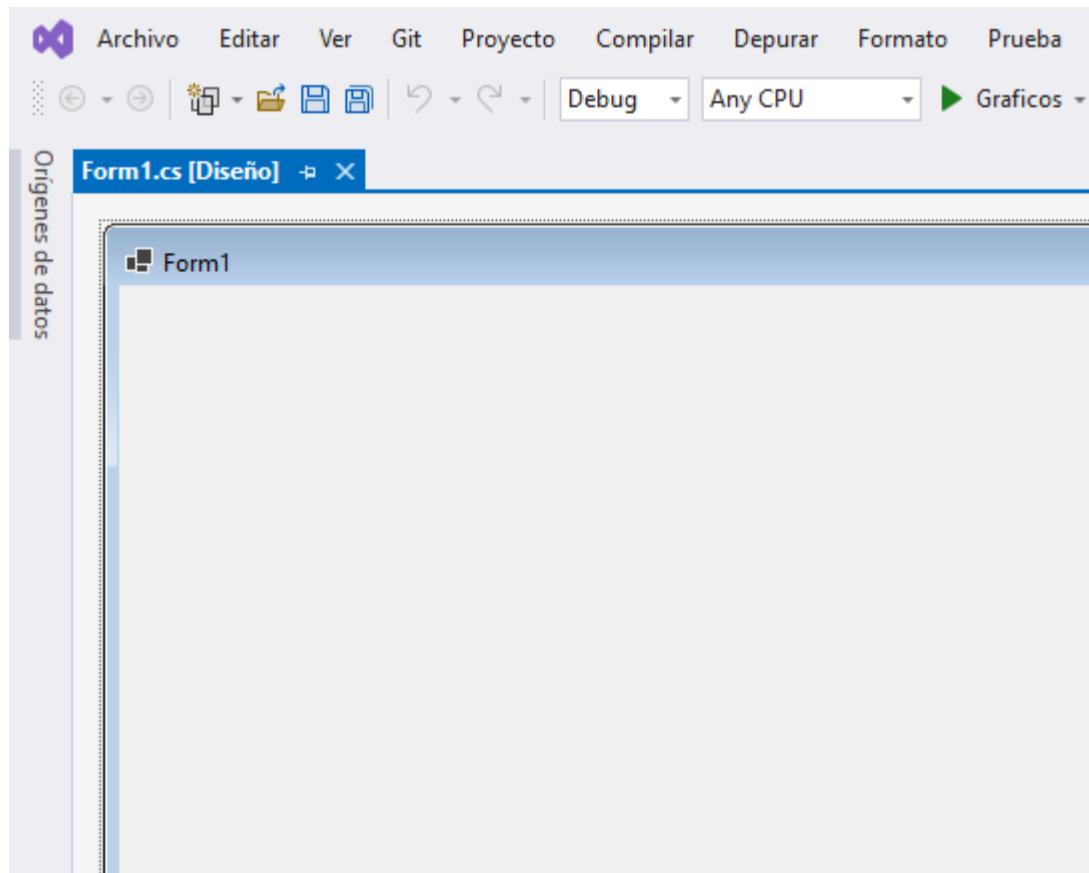


Ilustración 362: Una aplicación de escritorio típica

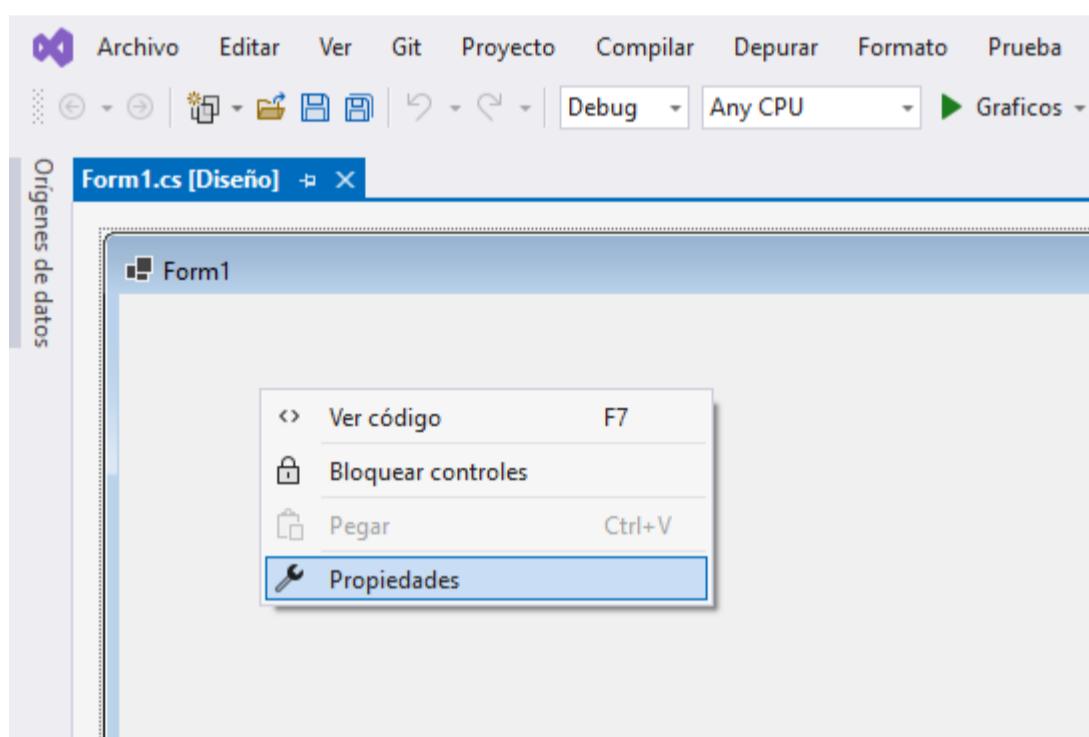


Ilustración 363: Clic botón derecho sobre la ventana y selecciona "Propiedades"

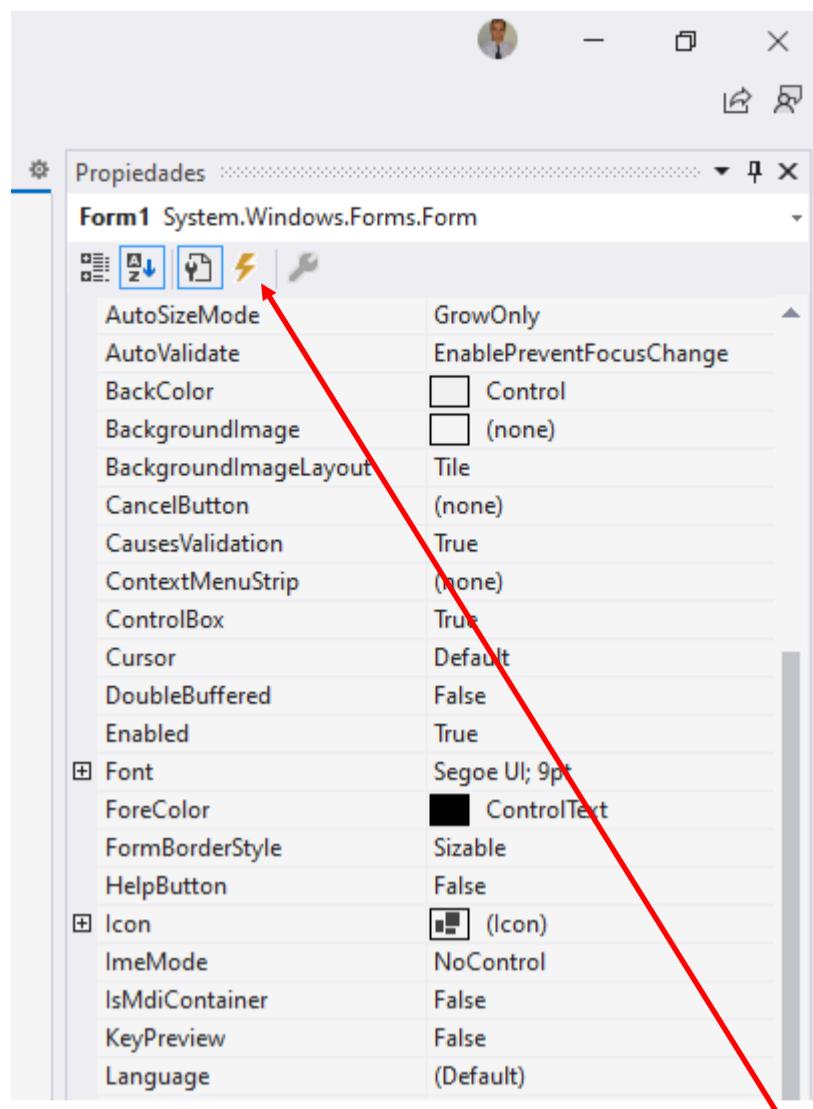


Ilustración 364: En la ventana de "Propiedades", selecciona "Eventos"

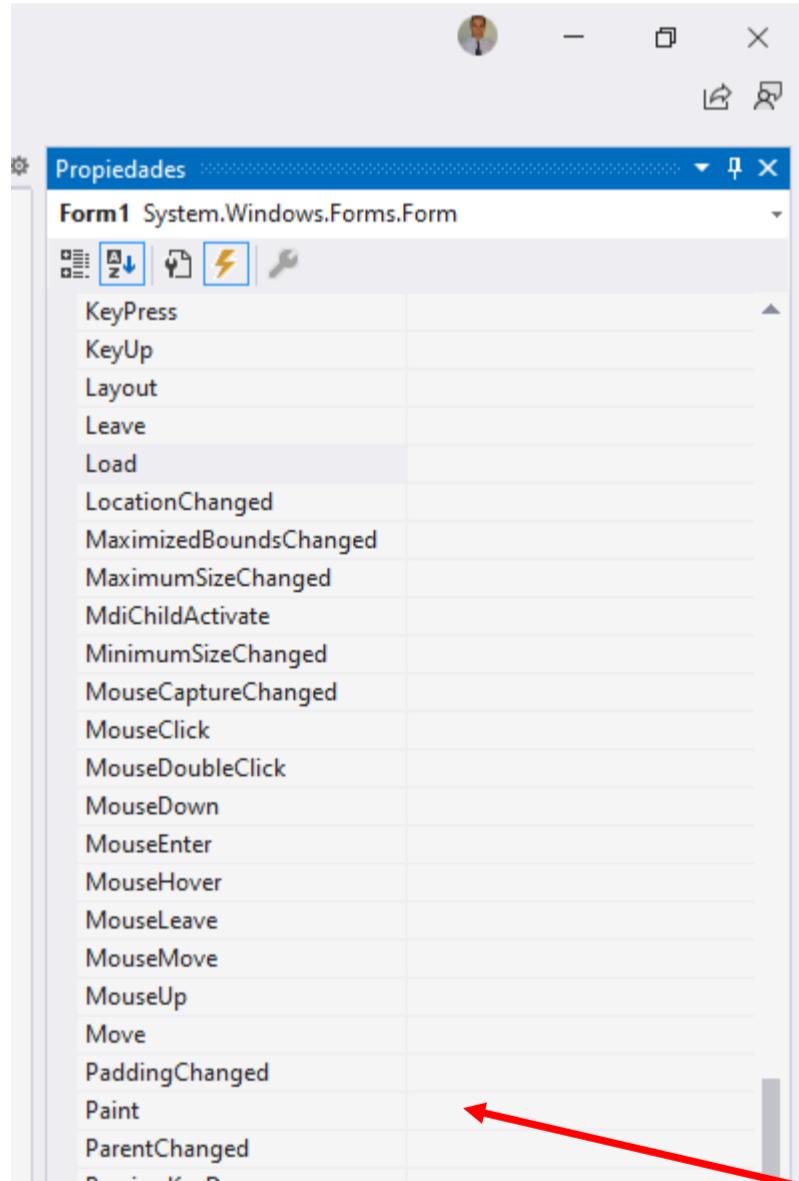


Ilustración 365: En "Eventos" se da doble clic al frente del evento "Paint"

```
namespace Graficos {
    public partial class Form1 : Form {
        public Form1() {
            InitializeComponent();
        }

        private void Form1_Paint(object sender, PaintEventArgs e) {
        }
    }
}
```

Ilustración 366: Esta es el código que se genera automáticamente

```
namespace Graficos {
    public partial class Form1 : Form {
        public Form1() {
            InitializeComponent();
        }

        private void Form1_Paint(object sender, PaintEventArgs e) {
            //Lienzo donde va a hacer el gráfico
            Graphics lienzo = e.Graphics;

            //Lápiz con que dibuja. Color, grosor
            Pen lapiz = new Pen(Color.Blue, 2);

            //=====
            //Línea: Xini, Yini, Xfin, Yfin
            //=====
            lienzo.DrawLine(lapiz, 10, 10, 130, 140);
        }
    }
}
```

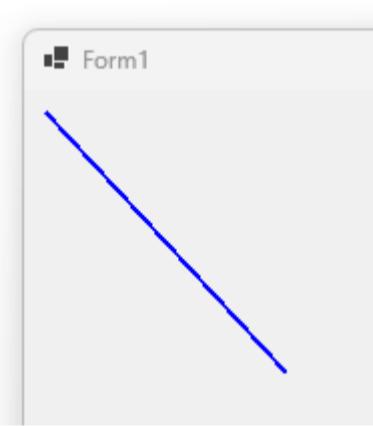


Ilustración 367: Línea

```
namespace Graficos {
    public partial class Form1 : Form {
        public Form1() {
            InitializeComponent();
        }

        private void Form1_Paint(object sender, PaintEventArgs e) {
            //Lienzo donde va a hacer el gráfico
            Graphics lienzo = e.Graphics;

            //Lápiz con que dibuja. Color, grosor
            Pen lapiz = new Pen(Color.Blue, 2);

            //=====
            //Arco: Xpos, Ypos, Ancho, Alto, Ángulo Inicial, Ángulo Final
            //=====
            lienzo.DrawArc(lapiz, 10, 90, 160, 170, 0, 270);
        }
    }
}
```

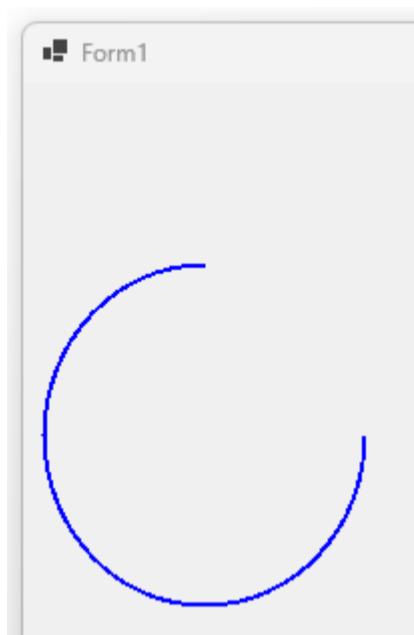


Ilustración 368: Arco

```
namespace Graficos {
    public partial class Form1 : Form {
        public Form1() {
            InitializeComponent();
        }

        private void Form1_Paint(object sender, PaintEventArgs e) {
            //Lienzo donde va a hacer el gráfico
            Graphics lienzo = e.Graphics;

            //Lápiz con que dibuja. Color, grosor
            Pen lapiz = new Pen(Color.Blue, 2);

            //=====
            //Rectángulo: Xpos, Ypos, Ancho, Alto
            //=====
            lienzo.DrawRectangle(lapiz, 100, 100, 200, 150);
        }
    }
}
```

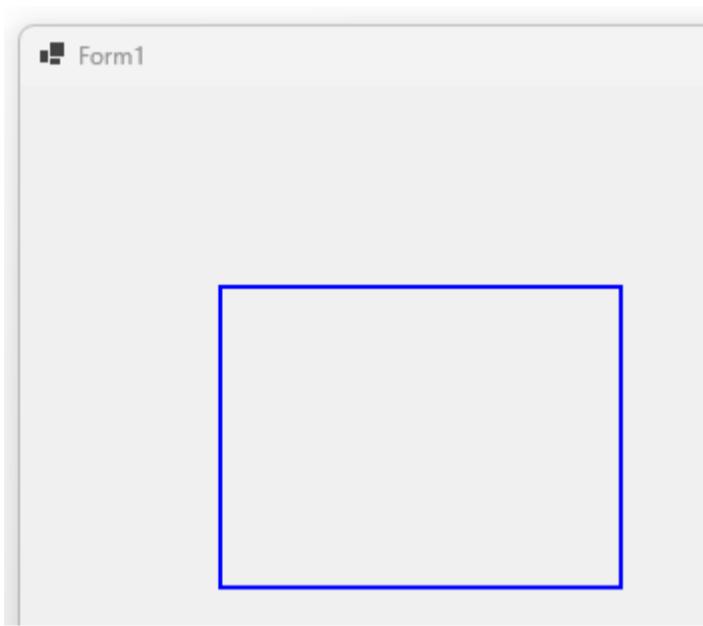


Ilustración 369: Rectángulo

```
namespace Graficos {
    public partial class Form1 : Form {
        public Form1() {
            InitializeComponent();
        }

        private void Form1_Paint(object sender, PaintEventArgs e) {
            //Lienzo donde va a hacer el gráfico
            Graphics lienzo = e.Graphics;

            //Con qué color se llenan las figuras
            SolidBrush Relleno = new SolidBrush(Color.Red);

            //=====
            //Rectángulo: Xpos, Ypos, Ancho, Alto
            //=====
            lienzo.FillRectangle(Relleno, 100, 100, 200, 150);
        }
    }
}
```

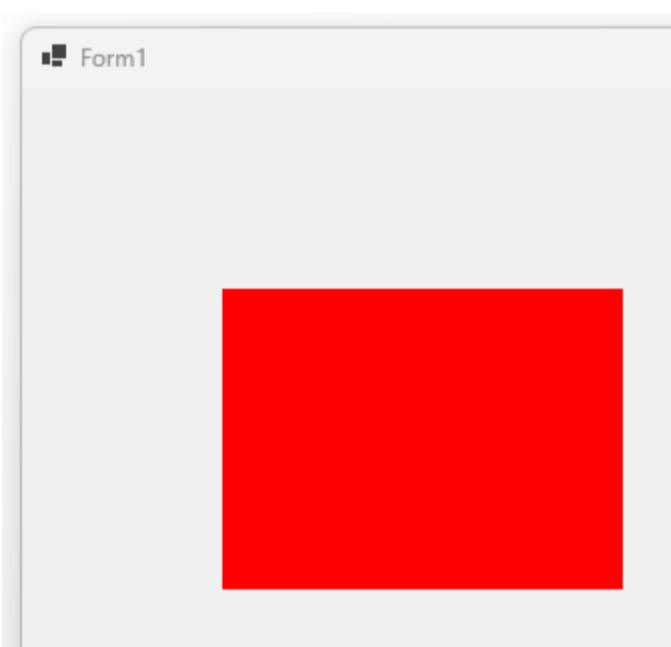


Ilustración 370: Rectángulo relleno

```
namespace Graficos {
    public partial class Form1 : Form {
        public Form1() {
            InitializeComponent();
        }

        private void Form1_Paint(object sender, PaintEventArgs e) {
            //Lienzo donde va a hacer el gráfico
            Graphics lienzo = e.Graphics;

            //Lápiz con que dibuja. Color, grosor
            Pen lapiz = new Pen(Color.Blue, 2);

            //=====
            //Dibujar una Curva de Bézier
            //=====

            Point P0 = new Point(100, 180);
            Point P1 = new Point(200, 10);
            Point P2 = new Point(350, 50);
            Point P3 = new Point(500, 180);
            lienzo.DrawBezier(lapiz, P0, P1, P2, P3);
        }
    }
}
```

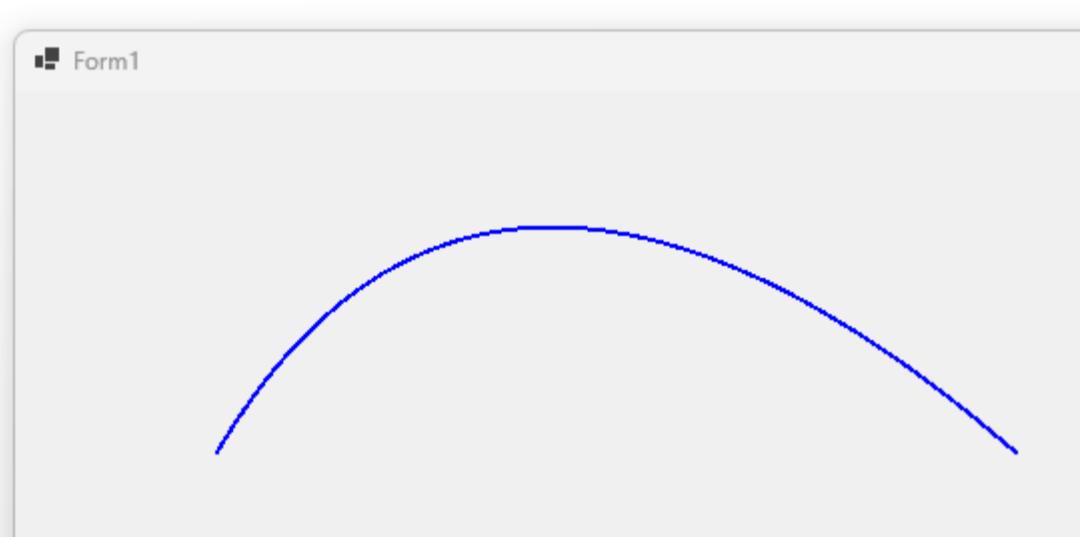


Ilustración 371: Curva Bézier

```
namespace Graficos {
    public partial class Form1 : Form {
        public Form1() {
            InitializeComponent();
        }

        private void Form1_Paint(object sender, PaintEventArgs e) {
            //Lienzo donde va a hacer el gráfico
            Graphics lienzo = e.Graphics;

            //Lápiz con que dibuja. Color, grosor
            Pen lapiz = new Pen(Color.Blue, 2);

            //Conjunto de puntos
            PointF punto1 = new PointF(150.0F, 150.0F);
            PointF punto2 = new PointF(200.0F, 50.0F);
            PointF punto3 = new PointF(300.0F, 140.0F);
            PointF punto4 = new PointF(400.0F, 200.0F);
            PointF punto5 = new PointF(450.0F, 300.0F);
            PointF punto6 = new PointF(350.0F, 350.0F);
            PointF punto7 = new PointF(300.0F, 150.0F);
            PointF[] Puntos = { punto1, punto2, punto3, punto4, punto5, punto6, punto7 };

            //Dibuja líneas rectas para unir los puntos
            lienzo.DrawLines(lapiz, Puntos);
        }
    }
}
```

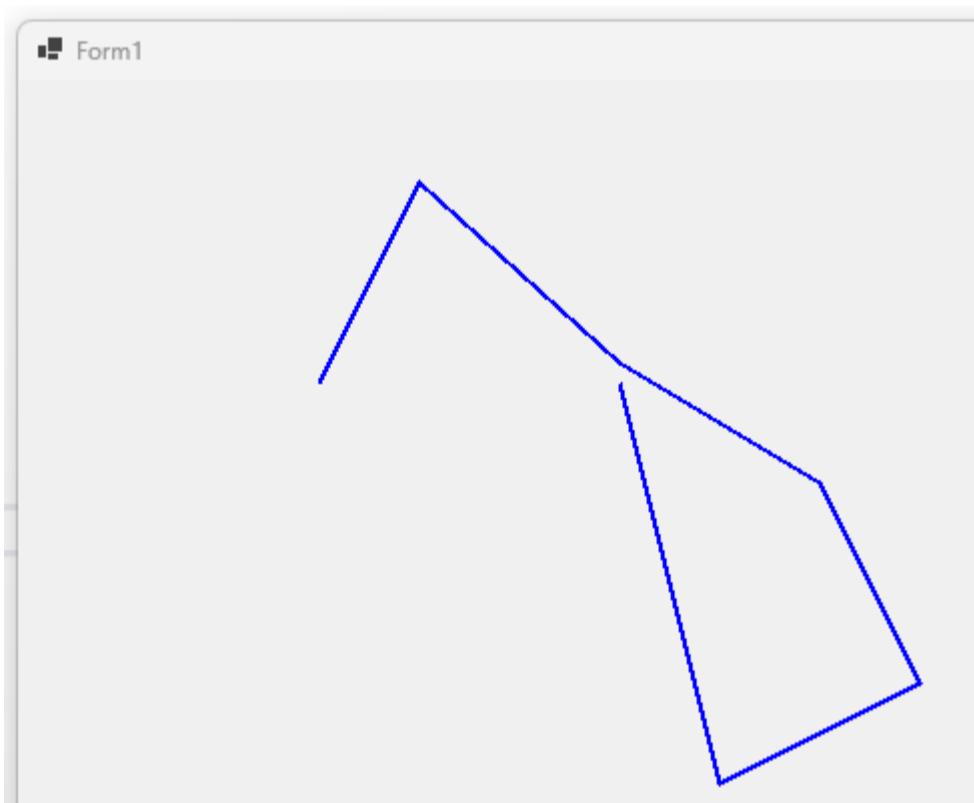


Ilustración 372: Varías líneas rectas

```
namespace Graficos {
    public partial class Form1 : Form {
        public Form1() {
            InitializeComponent();
        }

        private void Form1_Paint(object sender, PaintEventArgs e) {
            //Lienzo donde va a hacer el gráfico
            Graphics lienzo = e.Graphics;

            //Lápiz con que dibuja. Color, grosor
            Pen lapiz = new Pen(Color.Blue, 2);

            //Conjunto de puntos
            PointF punto1 = new PointF(150.0F, 150.0F);
            PointF punto2 = new PointF(200.0F, 50.0F);
            PointF punto3 = new PointF(300.0F, 140.0F);
            PointF punto4 = new PointF(400.0F, 200.0F);
            PointF punto5 = new PointF(450.0F, 300.0F);
            PointF punto6 = new PointF(350.0F, 350.0F);
            PointF punto7 = new PointF(300.0F, 150.0F);
            PointF[] Puntos = { punto1, punto2, punto3, punto4, punto5, punto6, punto7 };

            //Dibuja el polígono
            lienzo.DrawPolygon(lapiz, Puntos);
        }
    }
}
```

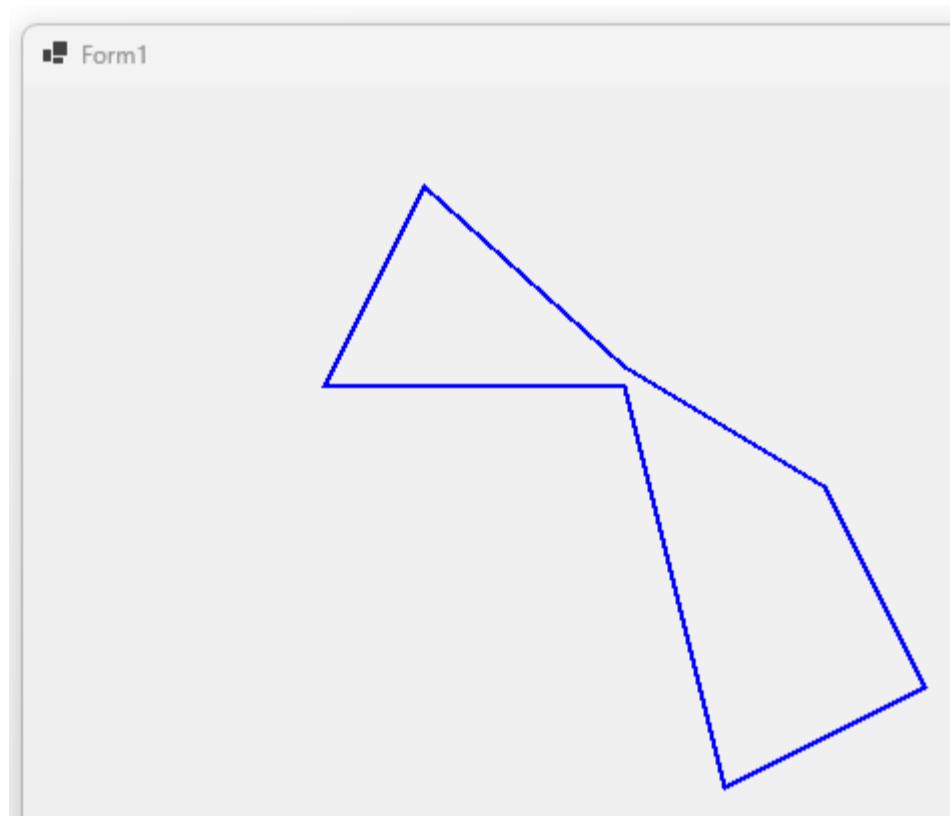


Ilustración 373: Polígono

```
namespace Graficos {
    public partial class Form1 : Form {
        public Form1() {
            InitializeComponent();
        }

        private void Form1_Paint(object sender, PaintEventArgs e) {
            //Lienzo donde va a hacer el gráfico
            Graphics lienzo = e.Graphics;

            //Con qué color se rellenan las figuras
            SolidBrush Relleno = new SolidBrush(Color.Red);

            //Conjunto de puntos
            PointF punto1 = new PointF(150.0F, 150.0F);
            PointF punto2 = new PointF(200.0F, 50.0F);
            PointF punto3 = new PointF(300.0F, 140.0F);
            PointF punto4 = new PointF(400.0F, 200.0F);
            PointF punto5 = new PointF(450.0F, 300.0F);
            PointF punto6 = new PointF(350.0F, 350.0F);
            PointF punto7 = new PointF(300.0F, 150.0F);
            PointF[] Puntos = { punto1, punto2, punto3, punto4, punto5, punto6, punto7 };

            //Dibuja el polígono
            lienzo.FillPolygon(Relleno, Puntos);
        }
    }
}
```

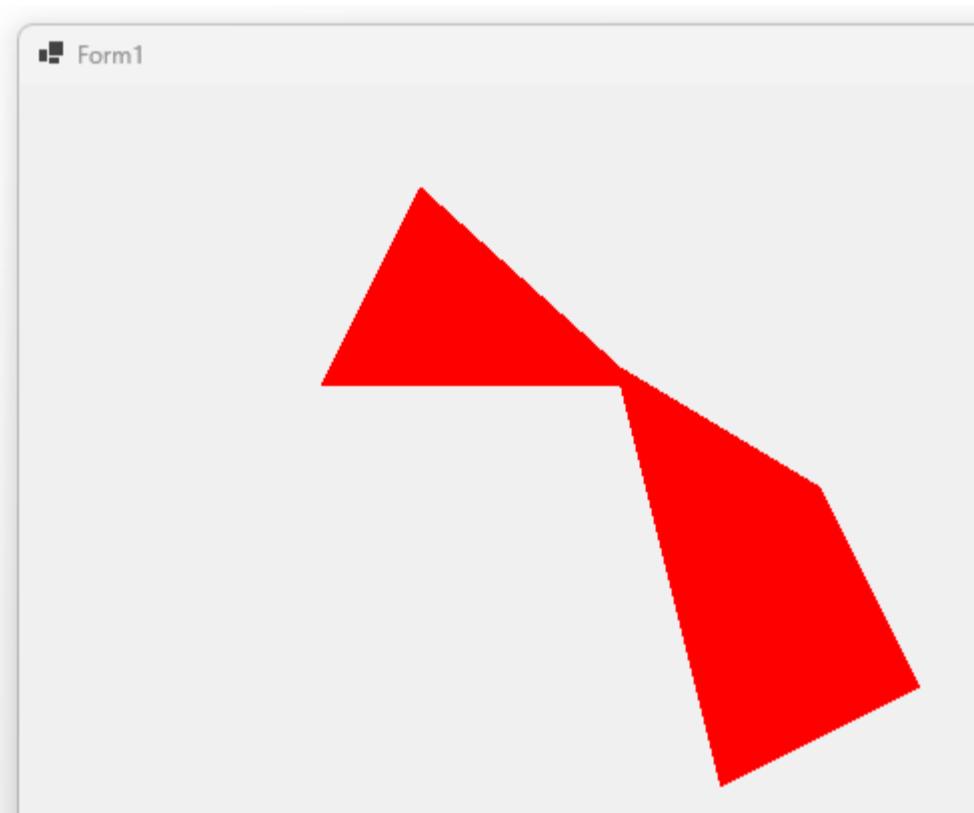


Ilustración 374: Polígono relleno

```
namespace Graficos {
    public partial class Form1 : Form {
        public Form1() {
            InitializeComponent();
        }

        private void Form1_Paint(object sender, PaintEventArgs e) {
            //Lienzo donde va a hacer el gráfico
            Graphics lienzo = e.Graphics;

            //Lápiz con que dibuja. Color, grosor
            Pen lapiz = new Pen(Color.Blue, 2);

            //Conjunto de Puntos
            PointF punto1 = new PointF(150.0F, 150.0F);
            PointF punto2 = new PointF(200.0F, 50.0F);
            PointF punto3 = new PointF(300.0F, 140.0F);
            PointF punto4 = new PointF(400.0F, 200.0F);
            PointF punto5 = new PointF(450.0F, 300.0F);
            PointF punto6 = new PointF(350.0F, 350.0F);
            PointF punto7 = new PointF(300.0F, 150.0F);
            PointF[] Puntos = { punto1, punto2, punto3, punto4, punto5, punto6, punto7 };

            //Dibuja una curva cerrada que une esos puntos
            lienzo.DrawClosedCurve(lapiz, Puntos);
        }
    }
}
```

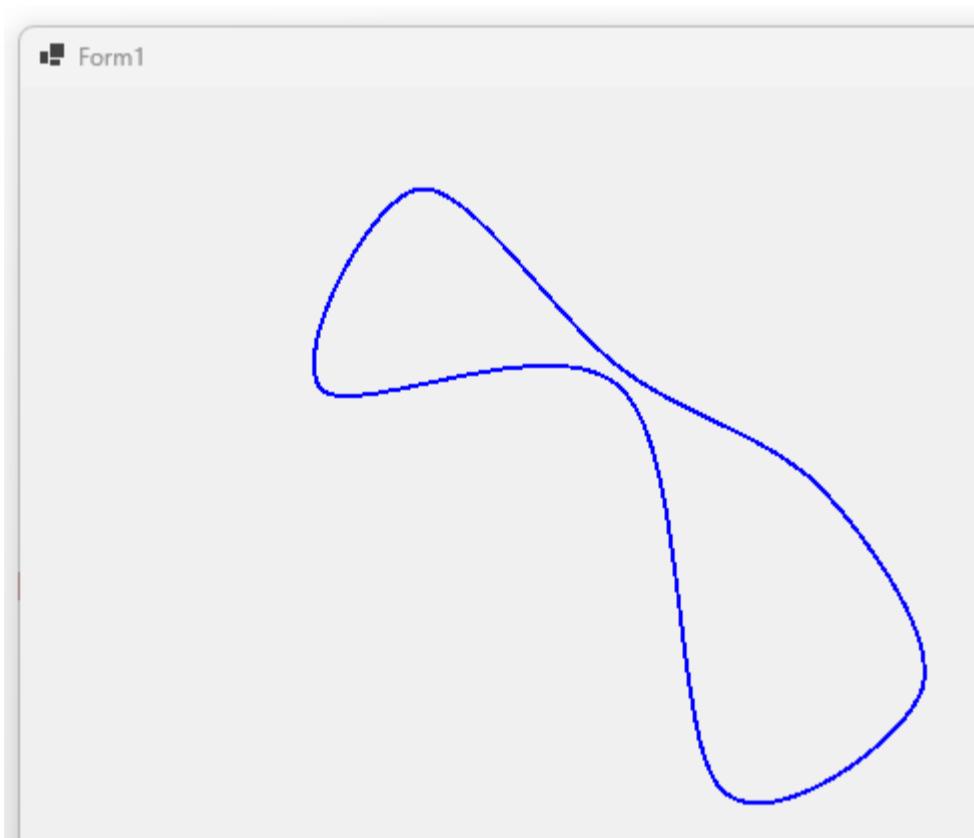


Ilustración 375: Curva cerrada

```
namespace Graficos {
    public partial class Form1 : Form {
        public Form1() {
            InitializeComponent();
        }

        private void Form1_Paint(object sender, PaintEventArgs e) {
            //Lienzo donde va a hacer el gráfico
            Graphics lienzo = e.Graphics;

            //Relleno
            SolidBrush Relleno = new SolidBrush(Color.Gray);

            //Conjunto de Puntos
            PointF punto1 = new PointF(150.0F, 150.0F);
            PointF punto2 = new PointF(200.0F, 50.0F);
            PointF punto3 = new PointF(300.0F, 140.0F);
            PointF punto4 = new PointF(400.0F, 200.0F);
            PointF punto5 = new PointF(450.0F, 300.0F);
            PointF punto6 = new PointF(350.0F, 350.0F);
            PointF punto7 = new PointF(300.0F, 150.0F);
            PointF[] Puntos = { punto1, punto2, punto3, punto4, punto5, punto6, punto7 };

            //Dibuja una curva cerrada rellena que une esos puntos
            lienzo.FillClosedCurve(Relleno, Puntos);
        }
    }
}
```

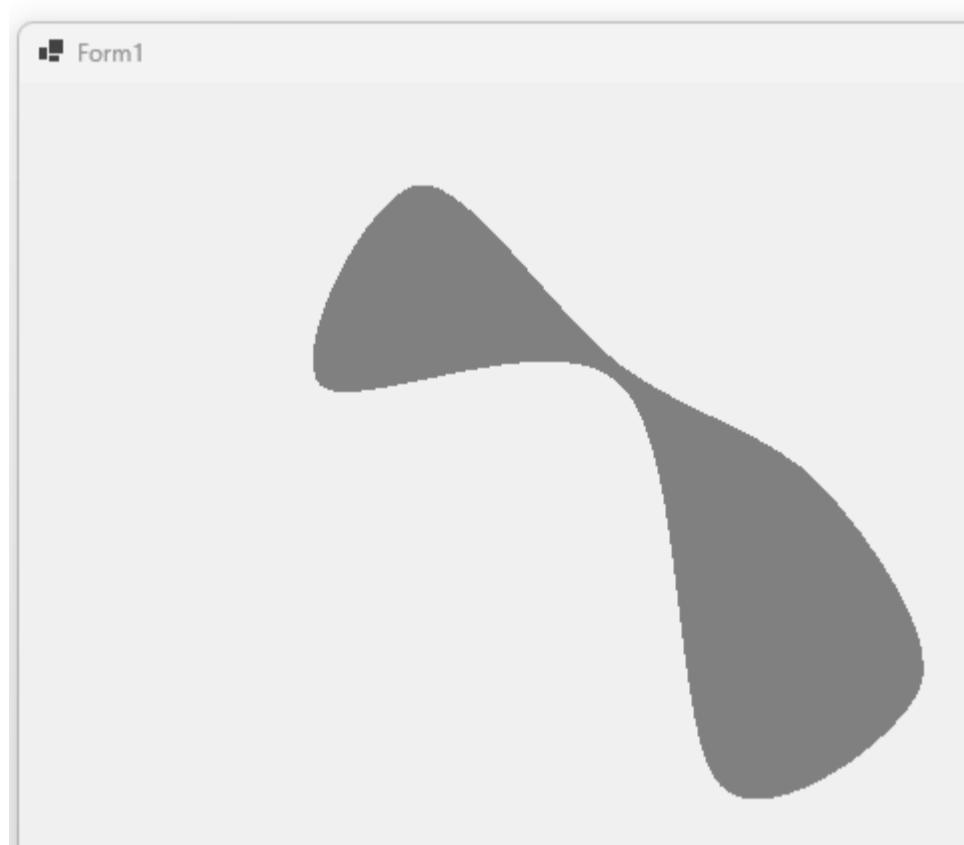


Ilustración 376: Curva cerrada rellena

```
namespace Graficos {
    public partial class Form1 : Form {
        public Form1() {
            InitializeComponent();
        }

        private void Form1_Paint(object sender, PaintEventArgs e) {
            //Lienzo donde va a hacer el gráfico
            Graphics lienzo = e.Graphics;

            //Lápiz con que dibuja. Color, grosor
            Pen lapiz = new Pen(Color.Blue, 2);

            //Conjunto de Puntos
            PointF punto1 = new PointF(150.0F, 150.0F);
            PointF punto2 = new PointF(200.0F, 50.0F);
            PointF punto3 = new PointF(300.0F, 140.0F);
            PointF punto4 = new PointF(400.0F, 200.0F);
            PointF punto5 = new PointF(450.0F, 300.0F);
            PointF punto6 = new PointF(350.0F, 350.0F);
            PointF punto7 = new PointF(300.0F, 150.0F);
            PointF[] Puntos = { punto1, punto2, punto3, punto4, punto5, punto6, punto7 };

            //Dibuja la curva que une esos puntos
            lienzo.DrawCurve(lapiz, Puntos);
        }
    }
}
```

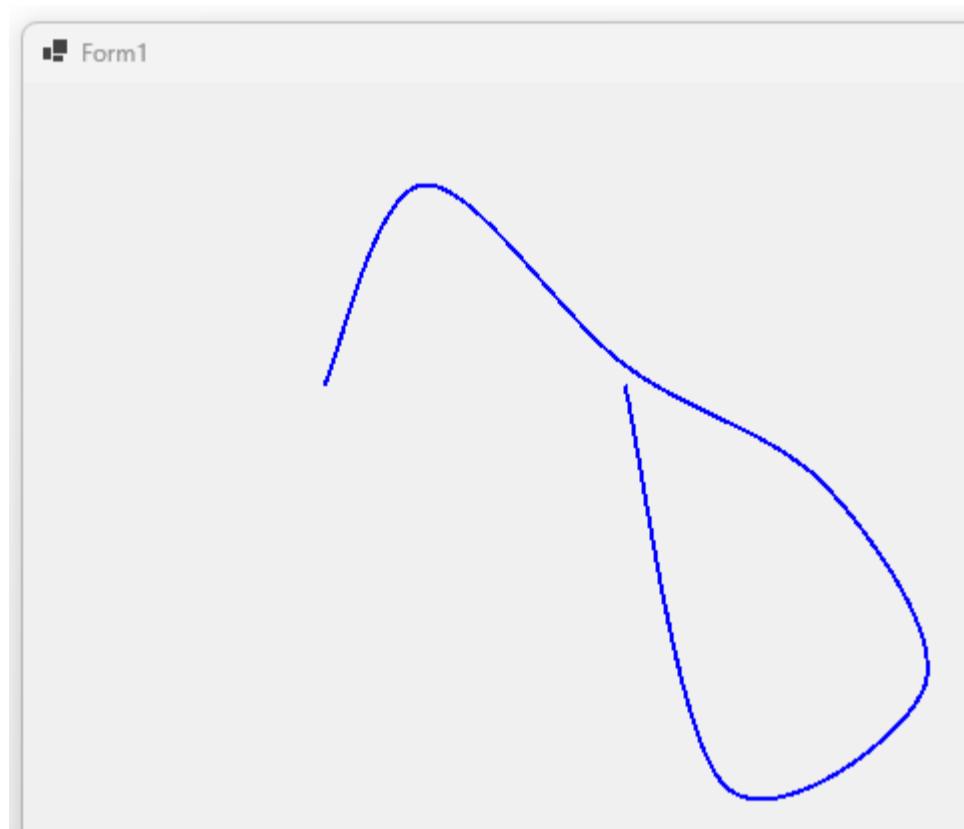


Ilustración 377: Curva que une varios puntos

```
namespace Graficos {
    public partial class Form1 : Form {
        public Form1() {
            InitializeComponent();
        }

        private void Form1_Paint(object sender, PaintEventArgs e) {
            //Lienzo donde va a hacer el gráfico
            Graphics lienzo = e.Graphics;

            //Lápiz con que dibuja. Color, grosor
            Pen lapiz = new Pen(Color.Blue, 2);

            //=====
            //Elipse: Xpos, Ypos, ancho, alto
            //=====
            lienzo.DrawEllipse(lapiz, 200, 30, 250, 90);
        }
    }
}
```

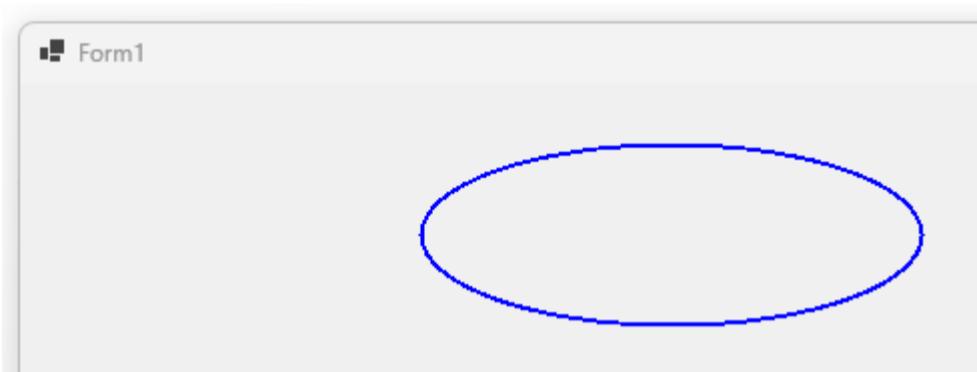


Ilustración 378: Elipse

```
namespace Graficos {
    public partial class Form1 : Form {
        public Form1() {
            InitializeComponent();
        }

        private void Form1_Paint(object sender, PaintEventArgs e) {
            //Lienzo donde va a hacer el gráfico
            Graphics lienzo = e.Graphics;

            //Relleno
            SolidBrush Relleno = new SolidBrush(Color.Chocolate);

            //=====
            //Elipse: Xpos, Ypos, ancho, alto
            //=====
            lienzo.FillEllipse(Relleno, 200, 30, 250, 90);
        }
    }
}
```



Ilustración 379: Elipse rellena

```
namespace Graficos {
    public partial class Form1 : Form {
        public Form1() {
            InitializeComponent();
        }

        private void Form1_Paint(object sender, PaintEventArgs e) {
            //Lienzo donde va a hacer el gráfico
            Graphics lienzo = e.Graphics;

            //Lápiz con que dibuja. Color, grosor
            Pen lapiz = new Pen(Color.Blue, 2);

            //=====
            //Letras
            //=====
            string Cadena = "Esta es una prueba";

            //Fuente y la brocha con que se pinta.
            Font FuenteLetra = new Font("Tahoma", 16);
            SolidBrush BrochaPinta = new SolidBrush(Color.Black);

            //Punto arriba a la izquierda para pintar la cadena
            PointF PuntoCadena = new PointF(100.0F, 140.0F);

            //Formato para dibujar
            StringFormat FormatoDibuja = new StringFormat();
            FormatoDibuja.FormatFlags = StringFormatFlags.DirectionVertical;

            //Dibuja la cadena
            lienzo.DrawString(Cadena, FuenteLetra, BrochaPinta, PuntoCadena, FormatoDibuja);
        }
    }
}
```

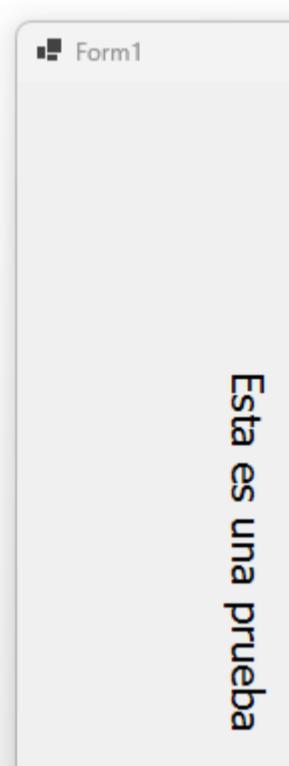


Ilustración 380: Letras

```
namespace Graficos {
    public partial class Form1 : Form {
        public Form1() {
            InitializeComponent();
        }

        private void Form1_Paint(object sender, PaintEventArgs e) {
            //Lienzo donde va a hacer el gráfico
            Graphics lienzo = e.Graphics;

            //Lápiz con que dibuja. Color, grosor
            Pen lapiz = new Pen(Color.Blue, 2);

            //=====
            //Dibuja un pastel
            //=====

            //Crea un rectángulo para la elipse que dibujará el pastel
            Rectangle rectangulo = new Rectangle(10, 10, 400, 400);

            //Pastel: rectángulo que contiene, ángulo inicial, ángulo de apertura
            lienzo.DrawPie(lapiz, rectangulo, 0F, 45F);
            lienzo.DrawPie(lapiz, rectangulo, 90F, 45F);
            lienzo.DrawPie(lapiz, rectangulo, 150F, 45F);
        }
    }
}
```

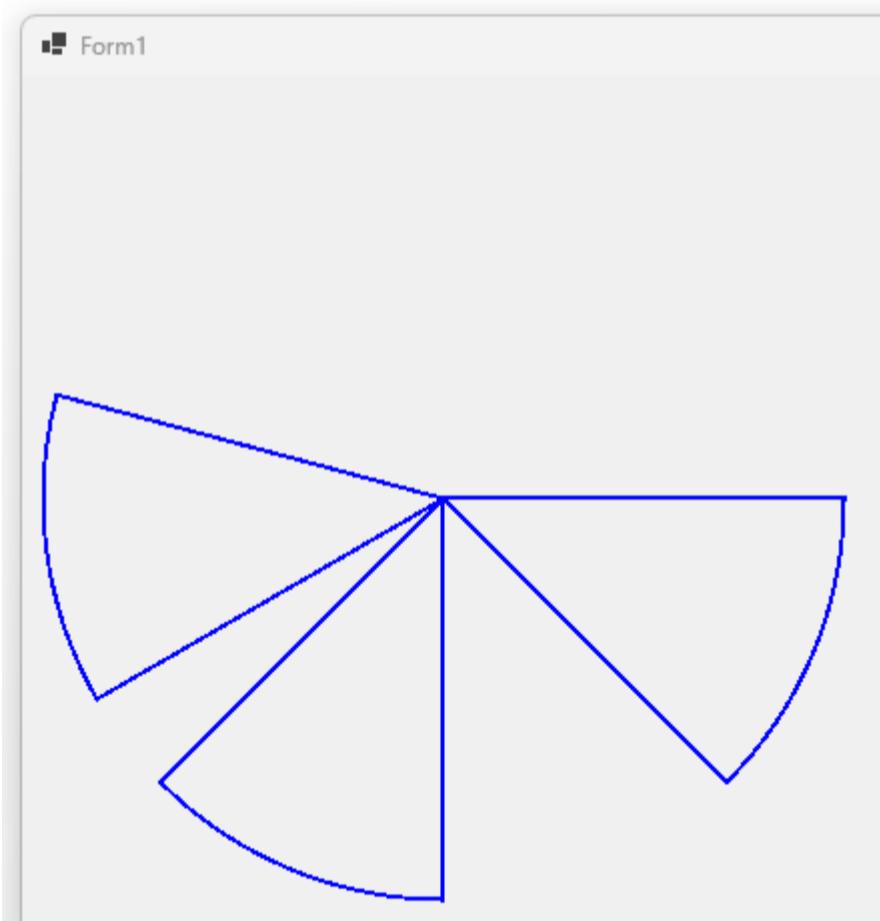


Ilustración 381: Diagrama de pastel

```
namespace Graficos {
    public partial class Form1 : Form {
        public Form1() {
            InitializeComponent();
        }

        private void Form1_Paint(object sender, PaintEventArgs e) {
            //Lienzo donde va a hacer el gráfico
            Graphics lienzo = e.Graphics;

            //Rellenos
            SolidBrush Relleno1 = new SolidBrush(Color.Chocolate);
            SolidBrush Relleno2 = new SolidBrush(Color.Red);
            SolidBrush Relleno3 = new SolidBrush(Color.Blue);

            //=====
            //Dibuja un pastel
            //=====

            //Crea un rectángulo para la ellipse que dibujará el pastel
            Rectangle rectangulo = new Rectangle(10, 10, 400, 400);

            //Pastel: rectángulo que contiene, ángulo inicial, ángulo de apertura
            lienzo.FillPie(Relleno1, rectangulo, 0F, 45F);
            lienzo.FillPie(Relleno2, rectangulo, 90F, 45F);
            lienzo.FillPie(Relleno3, rectangulo, 150F, 45F);
        }
    }
}
```

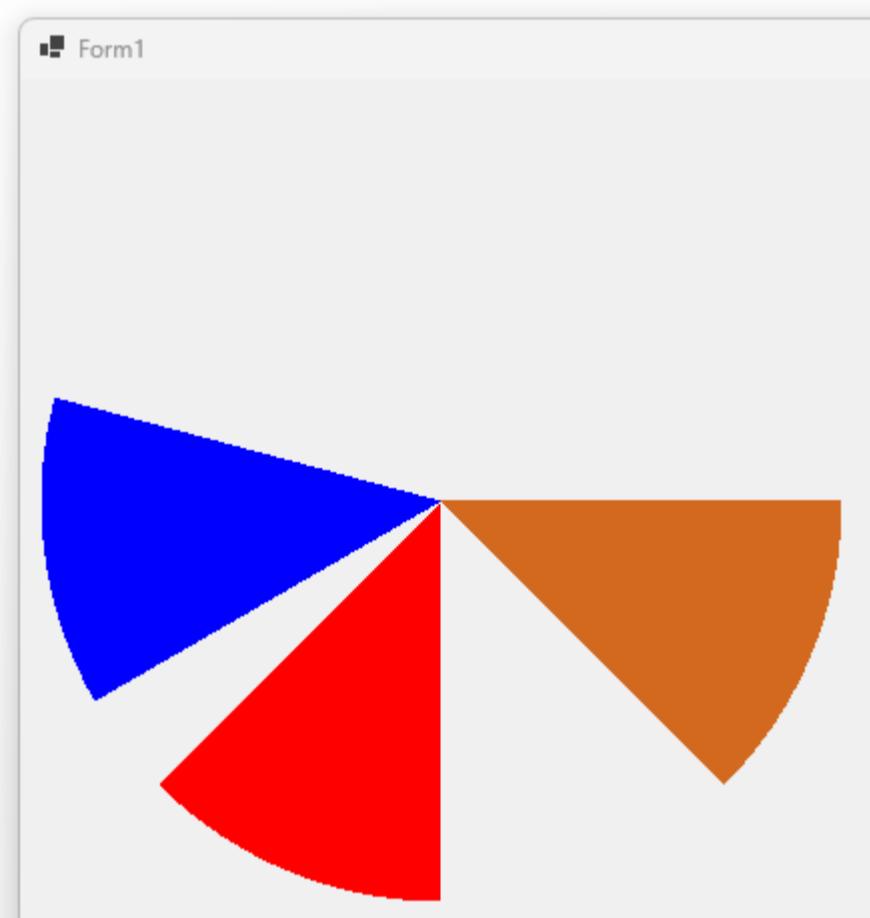


Ilustración 382: Diagrama de pastel relleno

```
namespace Graficos {
    public partial class Form1 : Form {
        public Form1() {
            InitializeComponent();
        }

        private void Form1_Paint(object sender, PaintEventArgs e) {
            //Lienzo
            Graphics grafico = e.Graphics;

            //Primer gráfico
            Pen lapiz = new Pen(Color.Blue, 2);
            int contador = 10;
            int incremento = 2;

            do {
                grafico.DrawLine(lapiz, 10, contador, 300, contador);
                incremento++;
                contador += incremento; //Incrementa el espacio entre línea y línea
            } while (contador <= 602);
            contador -= incremento;

            //Segundo gráfico
            Pen lapiz2 = new Pen(Color.Red, 2);
            incremento = 2;
            do {
                grafico.DrawLine(lapiz2, 300, contador, 590, contador);
                incremento++;
                contador -= incremento;
            } while (contador >= 10);
        }
    }
}
```

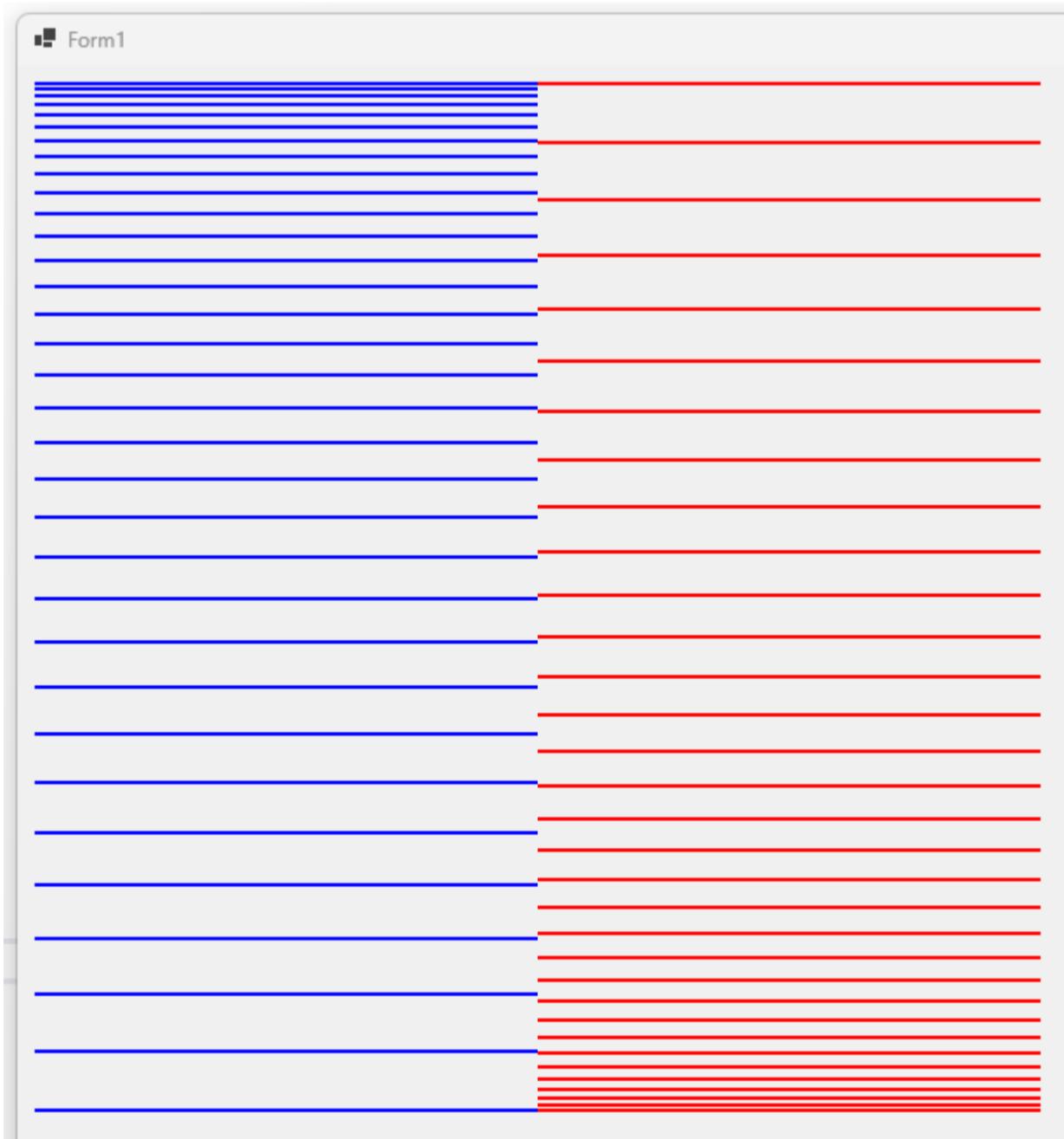


Ilustración 383: Líneas horizontales

```
namespace Graficos {
    public partial class Form1 : Form {
        public Form1() {
            InitializeComponent();
        }

        private void Form1_Paint(object sender, PaintEventArgs e) {
            //Lienzo
            Graphics grafico = e.Graphics;
            Pen lapiz = new Pen(Color.Blue, 2);
            int contador = 10;
            int incremento = 2;

            //Primer gráfico
            do {
                grafico.DrawLine(lapiz, contador, 10, contador, 200);
                incremento++;
                contador += incremento; //Incrementa el espacio entre línea y línea
            }
            while (contador <= 602);
            contador -= incremento;

            //Segundo gráfico
            Pen lapiz2 = new Pen(Color.Red, 2);
            incremento = 2;
            do {
                grafico.DrawLine(lapiz2, contador, 200, contador, 380);
                incremento++;
                contador -= incremento;
            }
            while (contador >= 10);
        }
    }
}
```

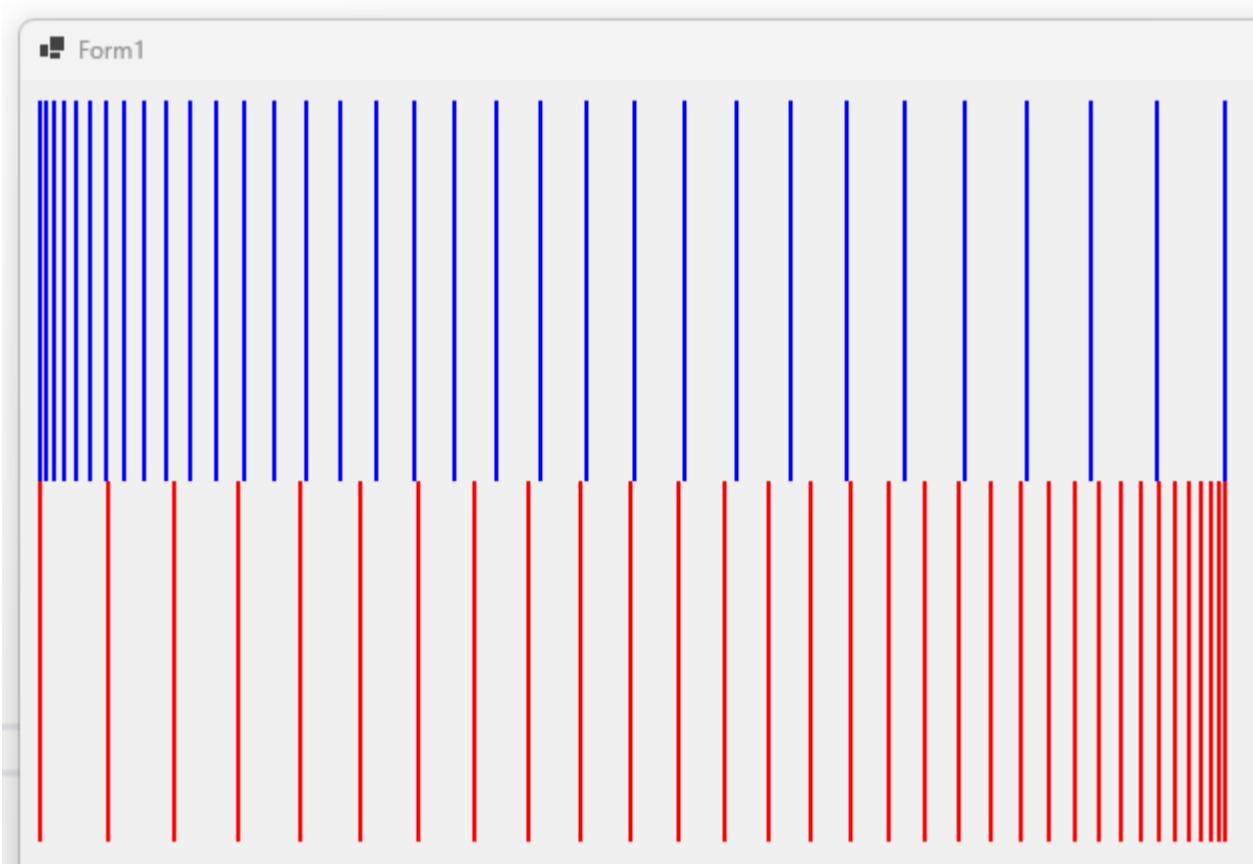


Ilustración 384: Líneas verticales

```
namespace Graficos {
    public partial class Form1 : Form {
        public Form1() {
            InitializeComponent();
        }

        private void Form1_Paint(object sender, PaintEventArgs e) {
            Graphics grafico = e.Graphics;
            Pen lapiz = new Pen(Color.Blue, 2);
            Pen lapiz2 = new Pen(Color.Red, 2);
            int limite = 500;
            int contador = limite;
            int incremento = 2;

            do {
                grafico.DrawLine(lapiz, contador, 10, contador, 200);
                grafico.DrawLine(lapiz, limite - (contador - limite), 10, limite - (contador - limite),
200);

                grafico.DrawLine(lapiz2, contador, 200, contador, 380);
                grafico.DrawLine(lapiz2, limite - (contador - limite), 200, limite - (contador -
limite), 380);
                incremento++;
                contador += incremento;
            }
            while (contador <= 2 * limite);
        }
    }
}
```

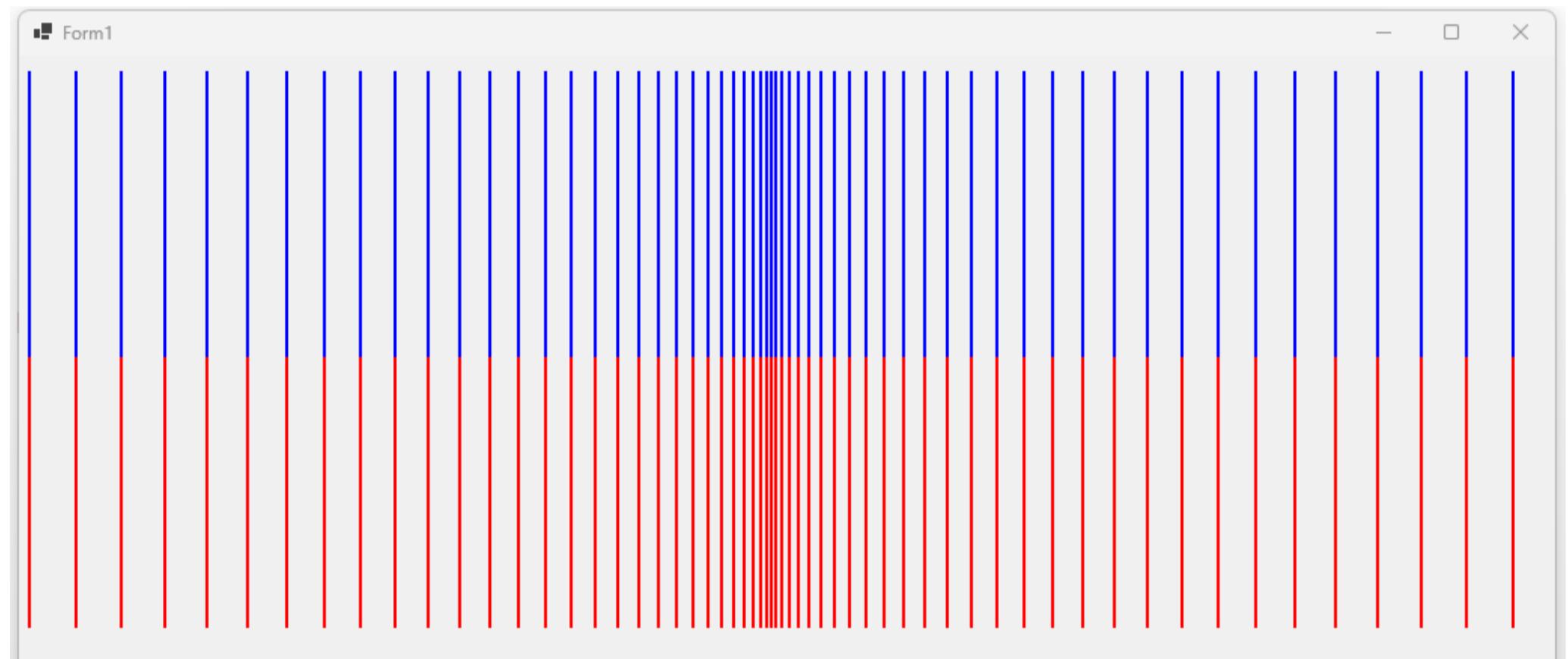


Ilustración 385: Líneas más juntas en el centro

```
namespace Graficos {
    public partial class Form1 : Form {
        public Form1() {
            InitializeComponent();
        }

        private void Form1_Paint(object sender, PaintEventArgs e) {
            Graphics grafico = e.Graphics;
            Pen lapiz = new Pen(Color.Blue, 2);
            Pen lapiz2 = new Pen(Color.Red, 2);
            int xval = 10;
            int yval = 5;

            do {
                grafico.DrawLine(lapiz, xval, 300 - yval, xval, 300 + yval);
                xval += 5;
                yval += 5;
            } while (yval < 300);

            do {
                grafico.DrawLine(lapiz2, xval, 300 - yval, xval, 300 + yval);
                xval += 5;
                yval -= 5;
            } while (yval > 0);
        }
    }
}
```

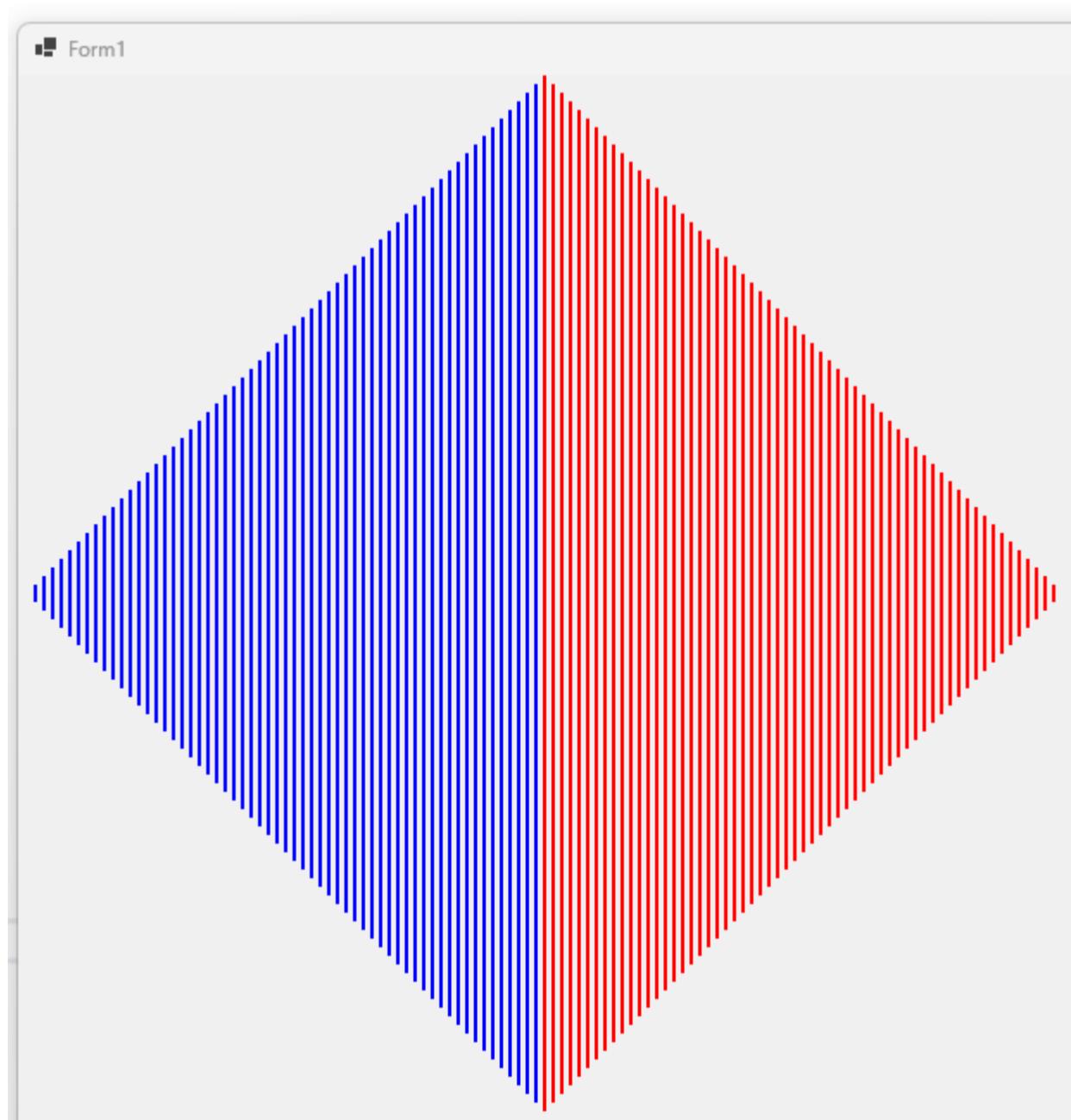


Ilustración 386: Líneas generan rombo

```
namespace Graficos {
    public partial class Form1 : Form {
        public Form1() {
            InitializeComponent();
        }

        private void Form1_Paint(object sender, PaintEventArgs e) {
            Graphics grafico = e.Graphics;
            Pen lapiz = new Pen(Color.Blue, 2);
            Pen lapiz2 = new Pen(Color.Red, 2);
            int xval = 600;
            int yval = 300;

            do {
                grafico.DrawLine(lapiz, xval, 300 - yval, xval, 300 + yval);
                xval -= 5;
                yval -= 5;
            }
            while (yval >= 0);

            xval = 300;
            yval = 0;
            do {
                grafico.DrawLine(lapiz2, 300 - xval, yval, 300 + xval, yval);
                xval -= 5;
                yval += 5;
            }
            while (yval <= 300);
        }
    }
}
```

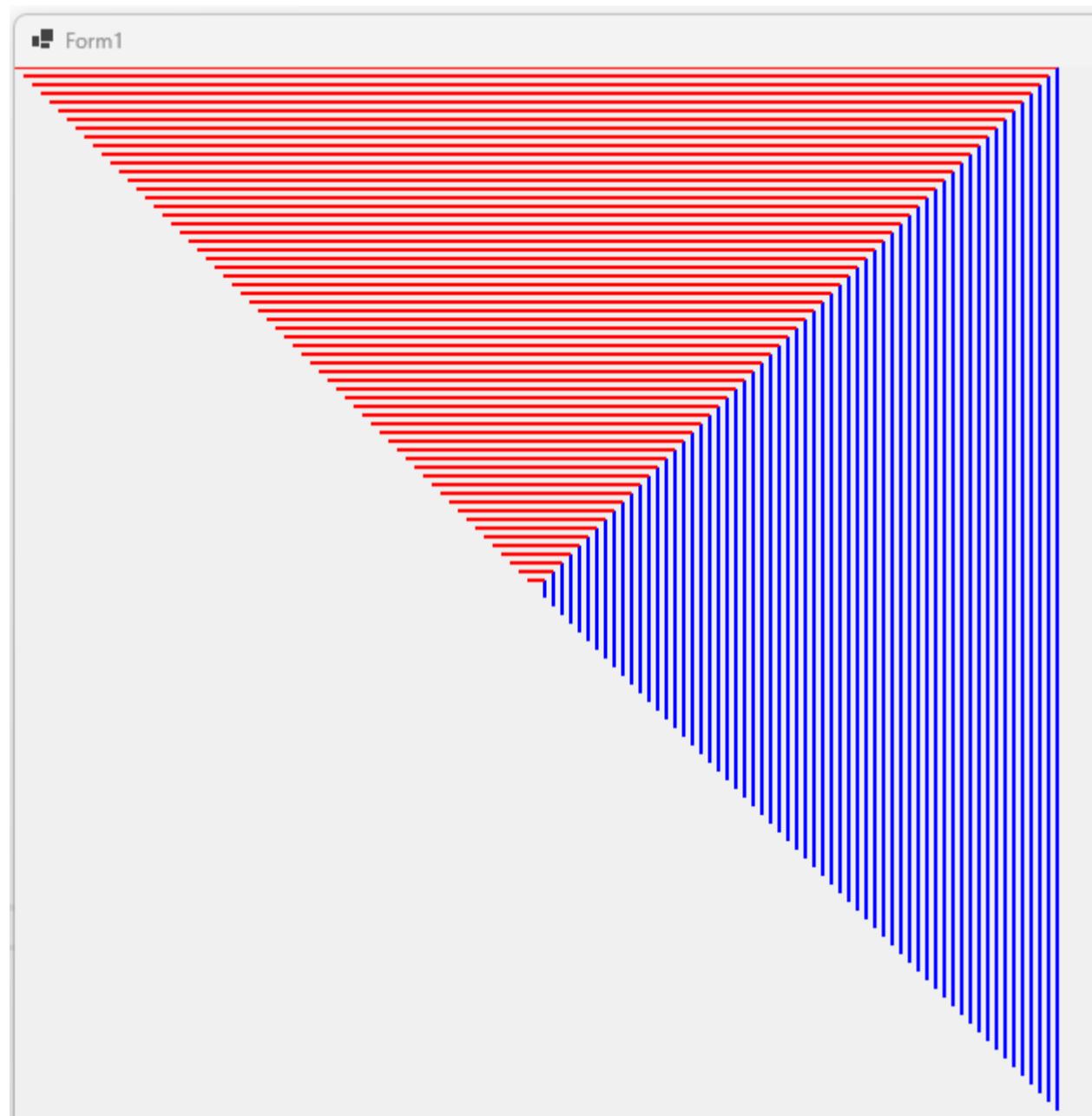


Ilustración 387: Líneas en esquina superior derecha

```
namespace Graficos {
    public partial class Form1 : Form {
        public Form1() {
            InitializeComponent();
        }

        private void Form1_Paint(object sender, PaintEventArgs e) {
            Graphics grafico = e.Graphics;
            Pen lapiz = new Pen(Color.Blue, 2);
            Pen lapiz2 = new Pen(Color.Red, 2);
            int xval = 600;
            int yval = 300;

            do {
                grafico.DrawLine(lapiz, xval, 300 - yval, xval, 300 + yval);
                xval -= 5;
                yval -= 5;
            }
            while (yval >= 0);

            xval = 300;
            yval = 600;
            do {
                grafico.DrawLine(lapiz2, 300 - xval, yval, 300 + xval, yval);
                xval -= 5;
                yval -= 5;
            }
            while (yval >= 300);
        }
    }
}
```

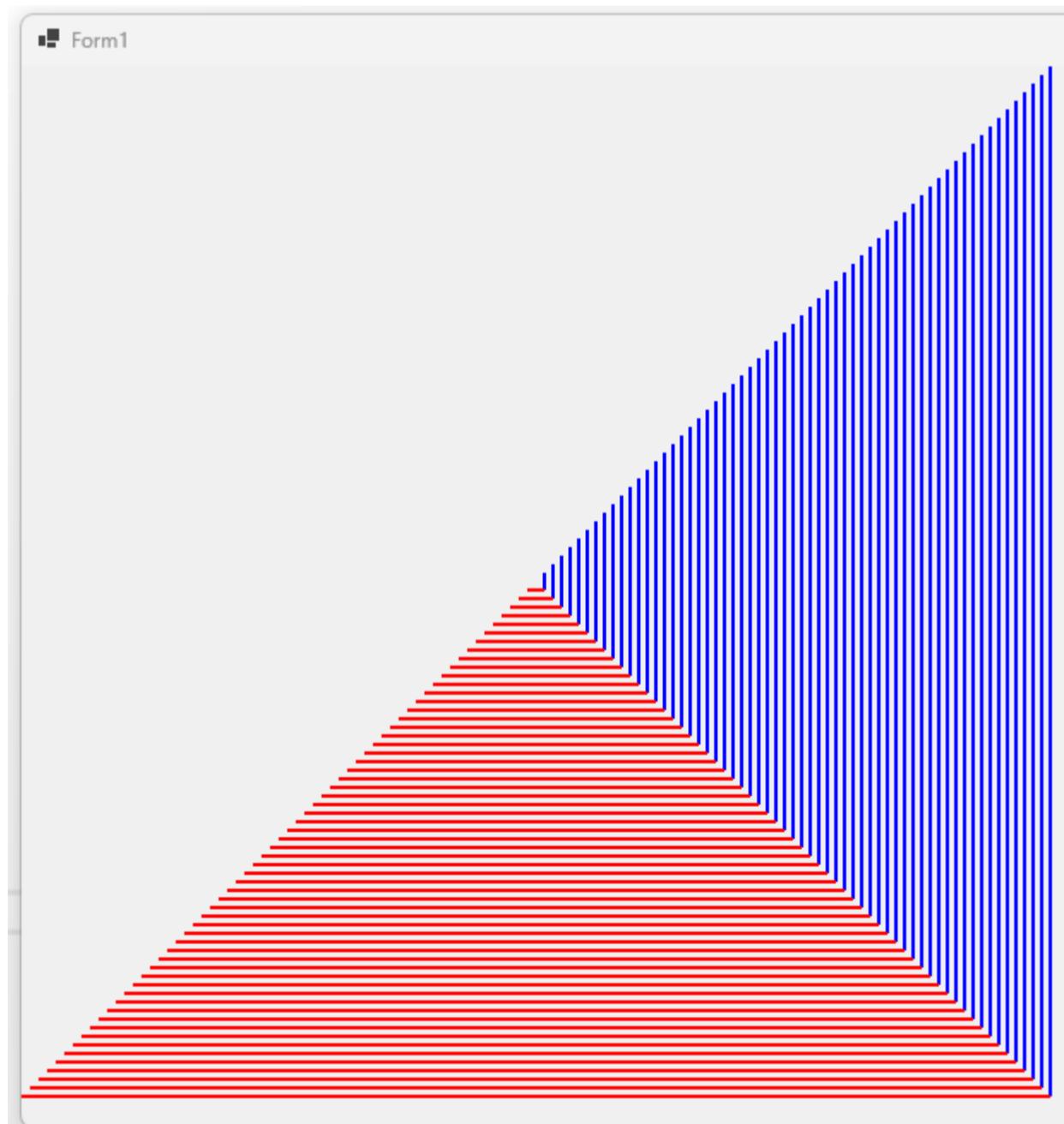


Ilustración 388: Líneas en esquina inferior derecha

```
namespace Graficos {
    public partial class Form1 : Form {
        public Form1() {
            InitializeComponent();
        }

        private void Form1_Paint(object sender, PaintEventArgs e) {
            Graphics grafico = e.Graphics;
            Pen lapiz = new Pen(Color.Blue, 2);
            Pen lapiz2 = new Pen(Color.Red, 2);
            int xval = 0;
            int yval = 300;

            do {
                grafico.DrawLine(lapiz, xval, 300 - yval, xval, 300 + yval);
                xval += 5;
                yval -= 5;
            }
            while (yval >= 0);

            xval = 300;
            yval = 600;
            do {
                grafico.DrawLine(lapiz2, 300 - xval, yval, 300 + xval, yval);
                xval -= 5;
                yval -= 5;
            }
            while (yval >= 300);
        }
    }
}
```

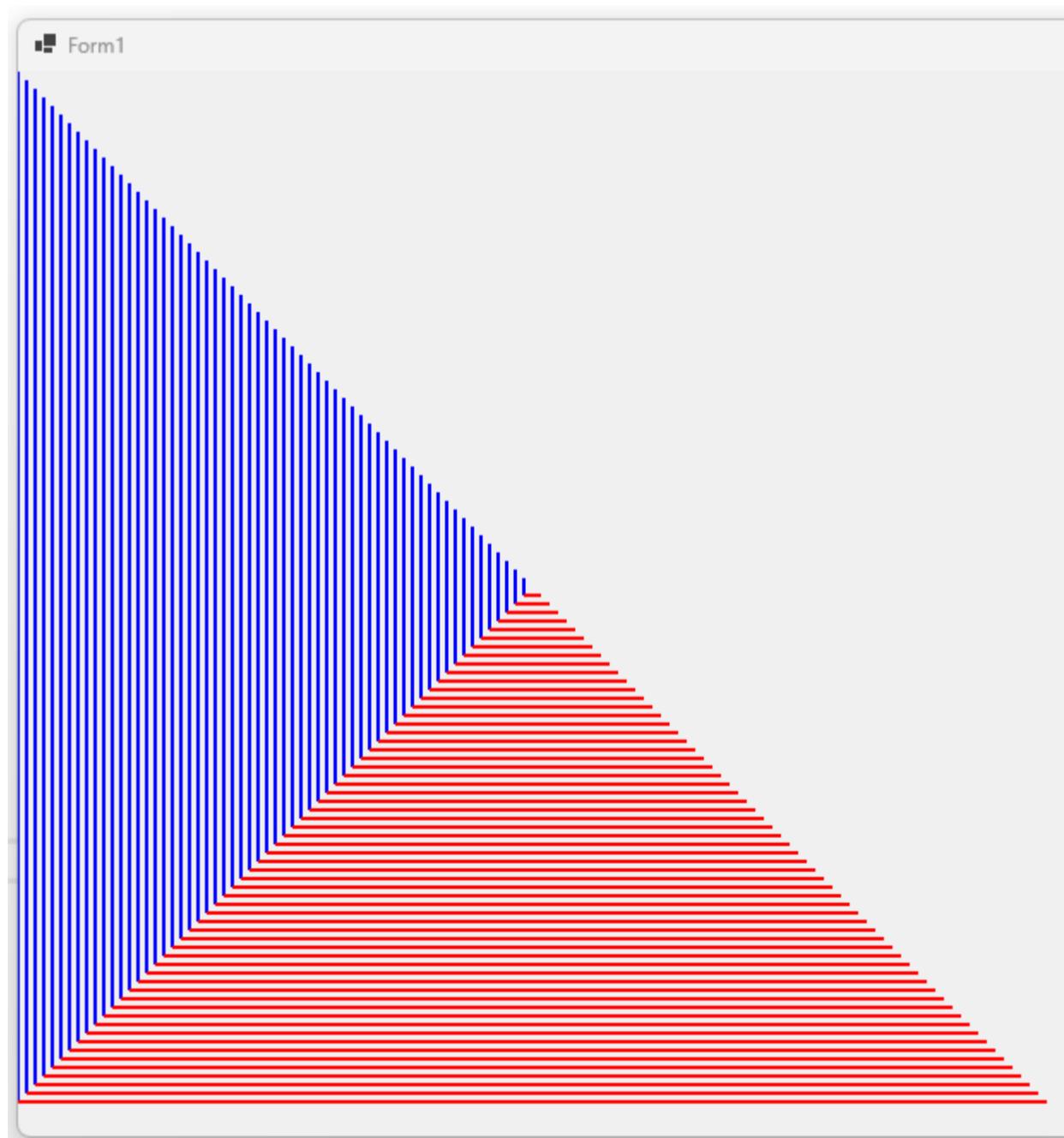


Ilustración 389: Líneas en esquina inferior izquierda

```
namespace Graficos {
    public partial class Form1 : Form {
        public Form1() {
            InitializeComponent();
        }

        private void Form1_Paint(object sender, PaintEventArgs e) {
            Graphics grafico = e.Graphics;
            Pen lapiz = new Pen(Color.Blue, 2);

            int pX1, pY1, pX2, pY2, pX3, pY3, constante;
            pX1 = 420;
            pY1 = 360;
            pX2 = 480;
            pY2 = 400;
            pX3 = 420;
            pY3 = 440;
            constante = 20;
            do {
                grafico.DrawLine(lapiz, pX1, pY1, pX2, pY2);
                grafico.DrawLine(lapiz, pX2, pY2, pX3, pY3);
                grafico.DrawLine(lapiz, pX1, pY1, pX3, pY3);
                pX1 -= constante;
                pY1 -= constante;
                pX2 += constante;
                pX3 -= constante;
                pY3 += constante;
            }
            while (pY1 >= 0);
        }
    }
}
```

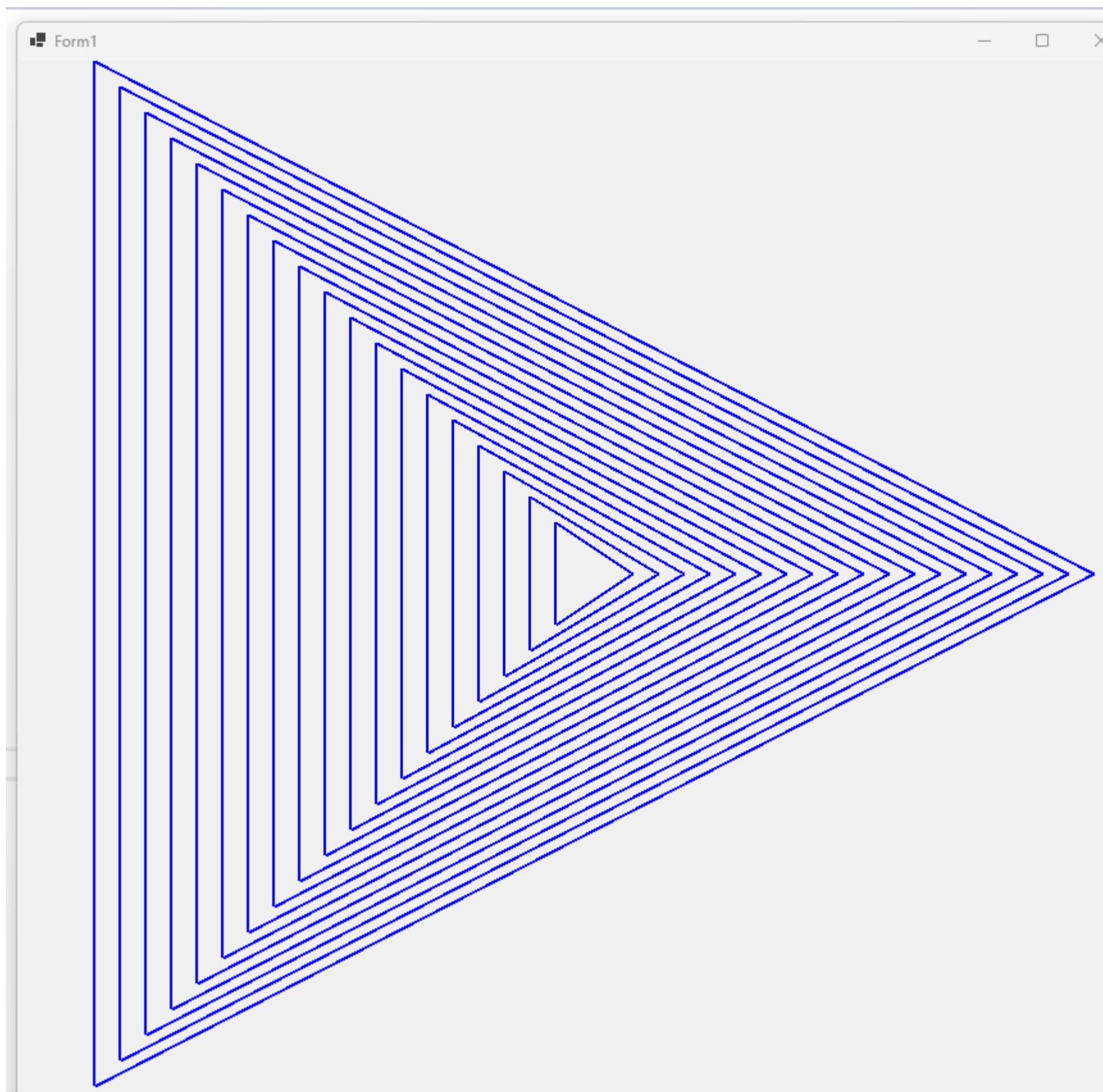


Ilustración 390: Triángulos apuntan a la derecha

```
namespace Graficos {
    public partial class Form1 : Form {
        public Form1() {
            InitializeComponent();
        }

        private void Form1_Paint(object sender, PaintEventArgs e) {
            Graphics grafico = e.Graphics;
            Pen lapiz = new Pen(Color.Blue, 2);

            int pX1, pY1, pX2, pY2, pX3, pY3, constante;
            pX1 = 420;
            pY1 = 360;
            pX2 = 360;
            pY2 = 400;
            pX3 = 420;
            pY3 = 440;
            constante = 20;
            do {
                grafico.DrawLine(lapiz, pX1, pY1, pX2, pY2);
                grafico.DrawLine(lapiz, pX2, pY2, pX3, pY3);
                grafico.DrawLine(lapiz, pX1, pY1, pX3, pY3);
                pX1 += constante;
                pY1 -= constante;
                pX2 -= constante;
                pX3 += constante;
                pY3 += constante;
            }
            while (pY1 >= 0);
        }
    }
}
```

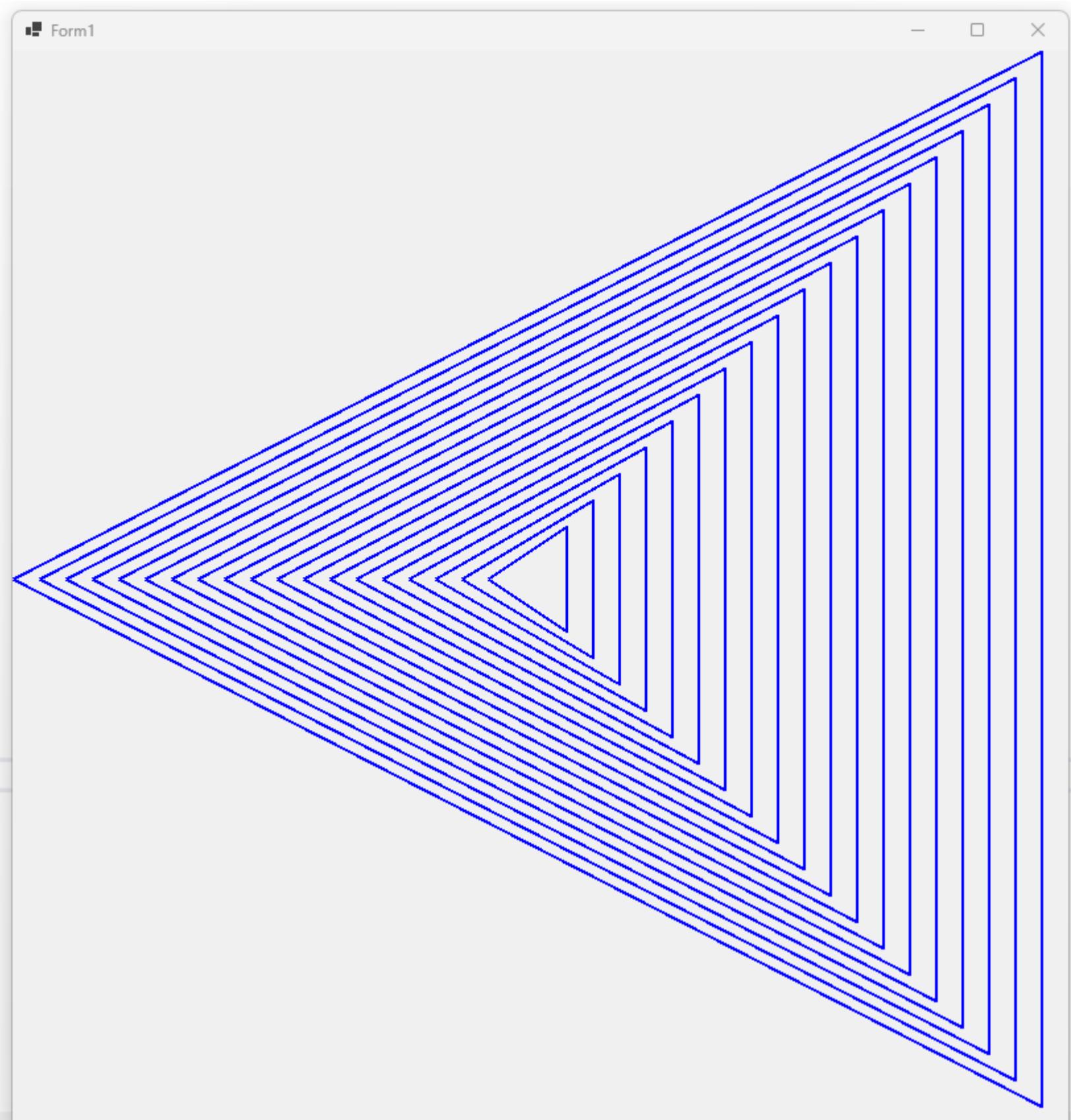


Ilustración 391: Triángulos apuntan a la izquierda

```
namespace Graficos {
    public partial class Form1 : Form {
        public Form1() {
            InitializeComponent();
        }

        private void Form1_Paint(object sender, PaintEventArgs e) {
            Graphics grafico = e.Graphics;
            Pen lapiz;
            Pen lapizRojo = new Pen(Color.Red, 2);
            Pen lapizAzul = new Pen(Color.Blue, 2);

            int pX1, pY1, pX2, pY2, pX3, pY3, constante, cambia;
            pX1 = 220;
            pY1 = 360;
            pX2 = 160;
            pY2 = 400;
            pX3 = 220;
            pY3 = 440;
            constante = 30;
            cambia = 1;
            do {
                if (cambia == 1) {
                    cambia = 2;
                    lapiz = lapizAzul;
                }
                else {
                    cambia = 1;
                    lapiz = lapizRojo;
                }

                grafico.DrawLine(lapiz, pX1, pY1, pX2, pY2);
                grafico.DrawLine(lapiz, pX2, pY2, pX3, pY3);
                grafico.DrawLine(lapiz, pX1, pY1, pX3, pY3);
                pX1 += constante;
                pY1 -= constante;
                pX2 += constante;
                pX3 += constante;
                pY3 += constante;
            }
            while (pY1 >= 0);
        }
    }
}
```

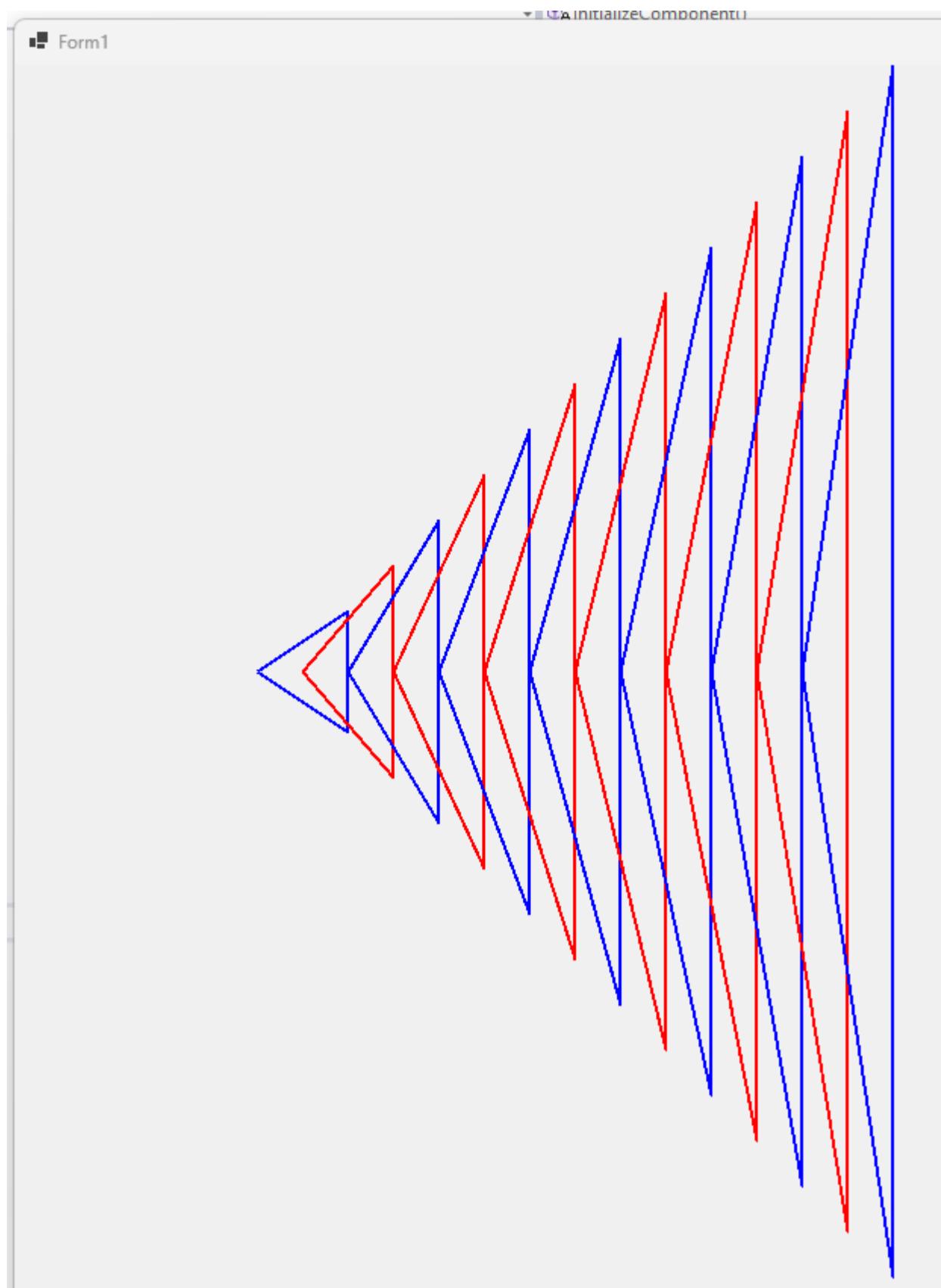


Ilustración 392: Triángulos apuntan a la izquierda alternando colores

```
namespace Graficos {
    public partial class Form1 : Form {
        public Form1() {
            InitializeComponent();
        }

        private void Form1_Paint(object sender, PaintEventArgs e) {
            Graphics grafico = e.Graphics;
            Pen lapizA = new Pen(Color.Blue, 2);
            Pen lapizB = new Pen(Color.Red, 2);
            Pen lapizC = new Pen(Color.Chocolate, 2);
            Pen lapizD = new Pen(Color.DarkGreen, 2);

            int variar, centro;
            variar = 0;
            centro = 380;

            for (var cont = 1; cont <= 30; cont += 1) {
                grafico.DrawLine(lapizA, centro - variar, centro - variar, centro - variar + variar * 2, centro - variar);
                grafico.DrawLine(lapizB, centro - variar, centro - variar, centro - variar, centro - variar, centro - variar + variar * 2);
                grafico.DrawLine(lapizC, centro - variar + variar * 2, centro - variar, centro - variar + variar * 2, centro - variar + variar * 2);
                grafico.DrawLine(lapizD, centro - variar, centro - variar + variar * 2, centro - variar + variar * 2, centro - variar + variar * 2);
                variar = variar + 10;
            }
        }
    }
}
```

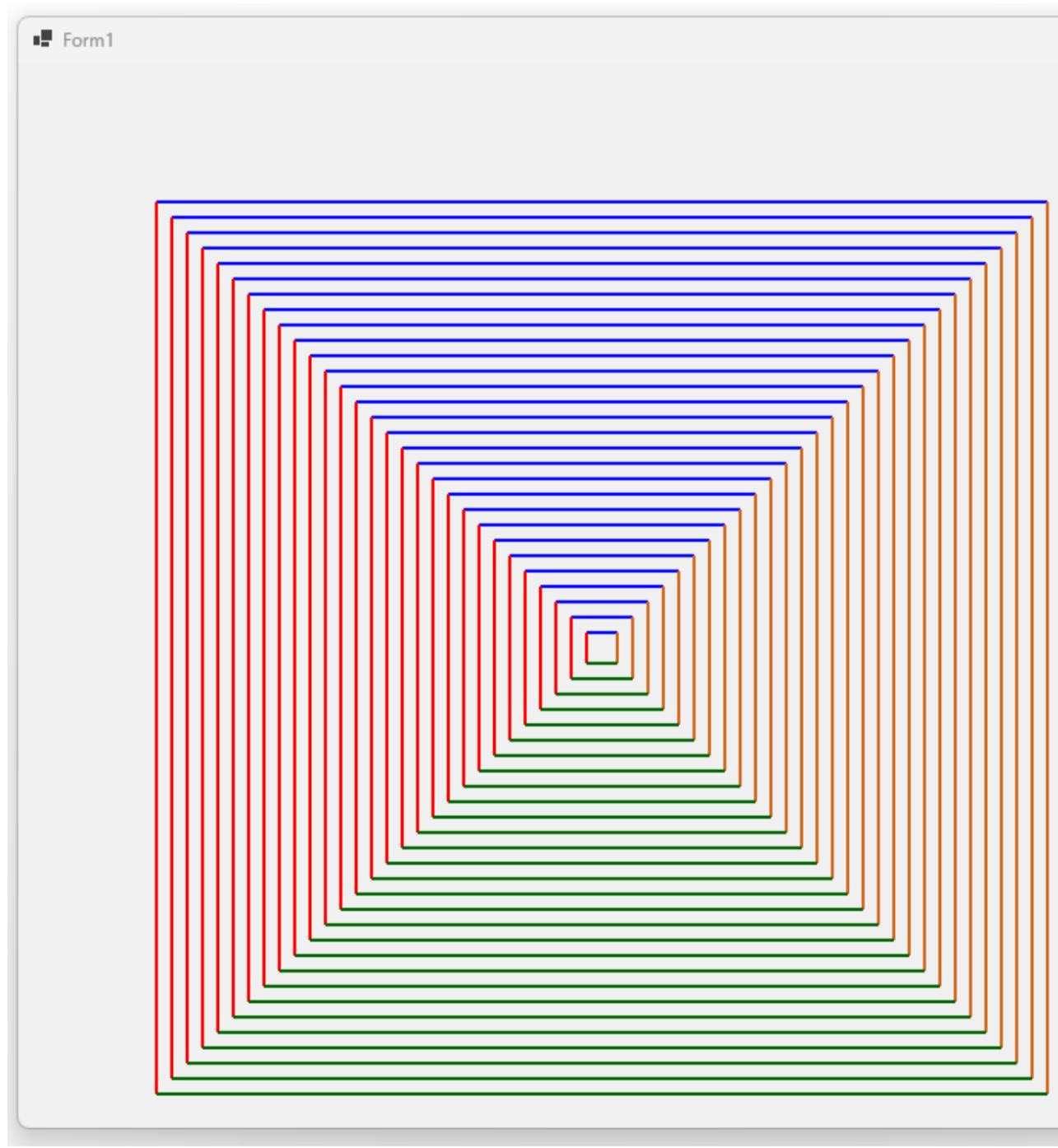


Ilustración 393: Rectángulos concéntricos

```
namespace Graficos {
    public partial class Form1 : Form {
        public Form1() {
            InitializeComponent();
        }

        private void Form1_Paint(object sender, PaintEventArgs e) {
            Graphics grafico = e.Graphics;
            Pen lapiz = new Pen(Color.Blue, 2);
            Pen lapiz2 = new Pen(Color.Red, 2);
            int variar = 0;

            for (var cont = 1; cont <= 18; cont += 1) {
                grafico.DrawLine(lapiz, 380 - variar, 380 - variar, 380 + variar, 380 - variar);
                grafico.DrawLine(lapiz, 380 - variar, 380 - variar, 380 - variar, 380 + variar);
                grafico.DrawLine(lapiz, 380 + variar, 380 - variar, 380 + variar, 380 + variar);
                variar += 10;

                grafico.DrawLine(lapiz2, 380 - variar, 380 + variar, 380 + variar, 380 + variar);
                grafico.DrawLine(lapiz2, 380 - variar, 380 - variar, 380 - variar, 380 + variar);
                grafico.DrawLine(lapiz2, 380 + variar, 380 - variar, 380 + variar, 380 + variar);
                variar += 10;
            }
        }
    }
}
```

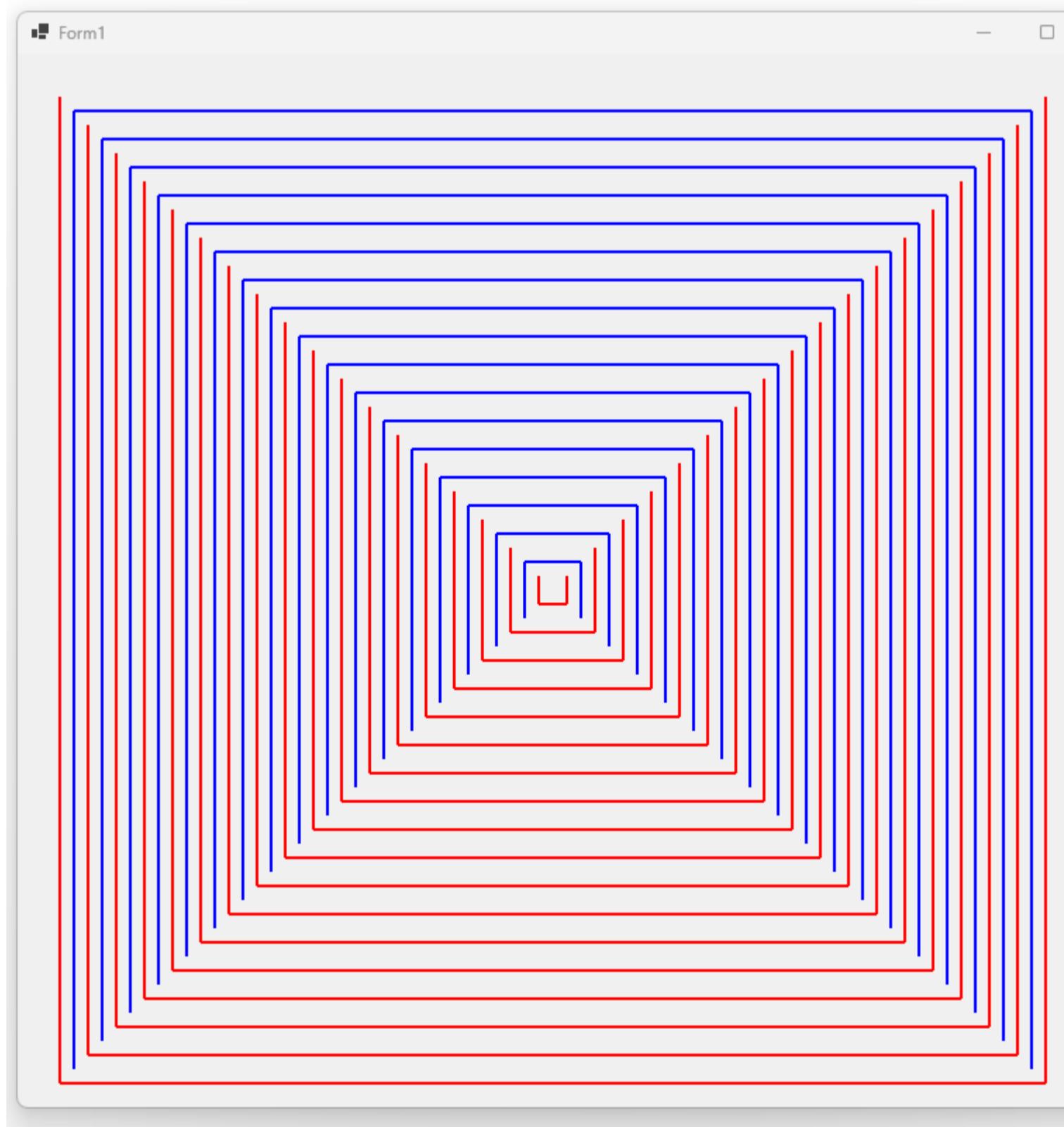


Ilustración 394: Dibuja formas que semejan cuadrados a las que les falta el techo o la base

```
namespace Graficos {
    public partial class Form1 : Form {
        public Form1() {
            InitializeComponent();
        }

        private void Form1_Paint(object sender, PaintEventArgs e) {
            Graphics grafico = e.Graphics;
            Pen lapiz = new Pen(Color.Blue, 2);
            Pen lapiz2 = new Pen(Color.Red, 2);
            int variar = 0;

            for (var cont = 1; cont <= 18; cont += 1) {
                grafico.DrawLine(lapiz, 380 - variar, 380 - variar, 380 + variar, 380 - variar);
                grafico.DrawLine(lapiz, 380 - variar, 380 - variar, 380 - variar, 380 + variar);
                variar += 10;

                grafico.DrawLine(lapiz2, 380 - variar, 380 + variar, 380 + variar, 380 + variar);
                grafico.DrawLine(lapiz2, 380 - variar, 380 - variar, 380 - variar, 380 + variar);
                variar += 10;
            }
        }
    }
}
```

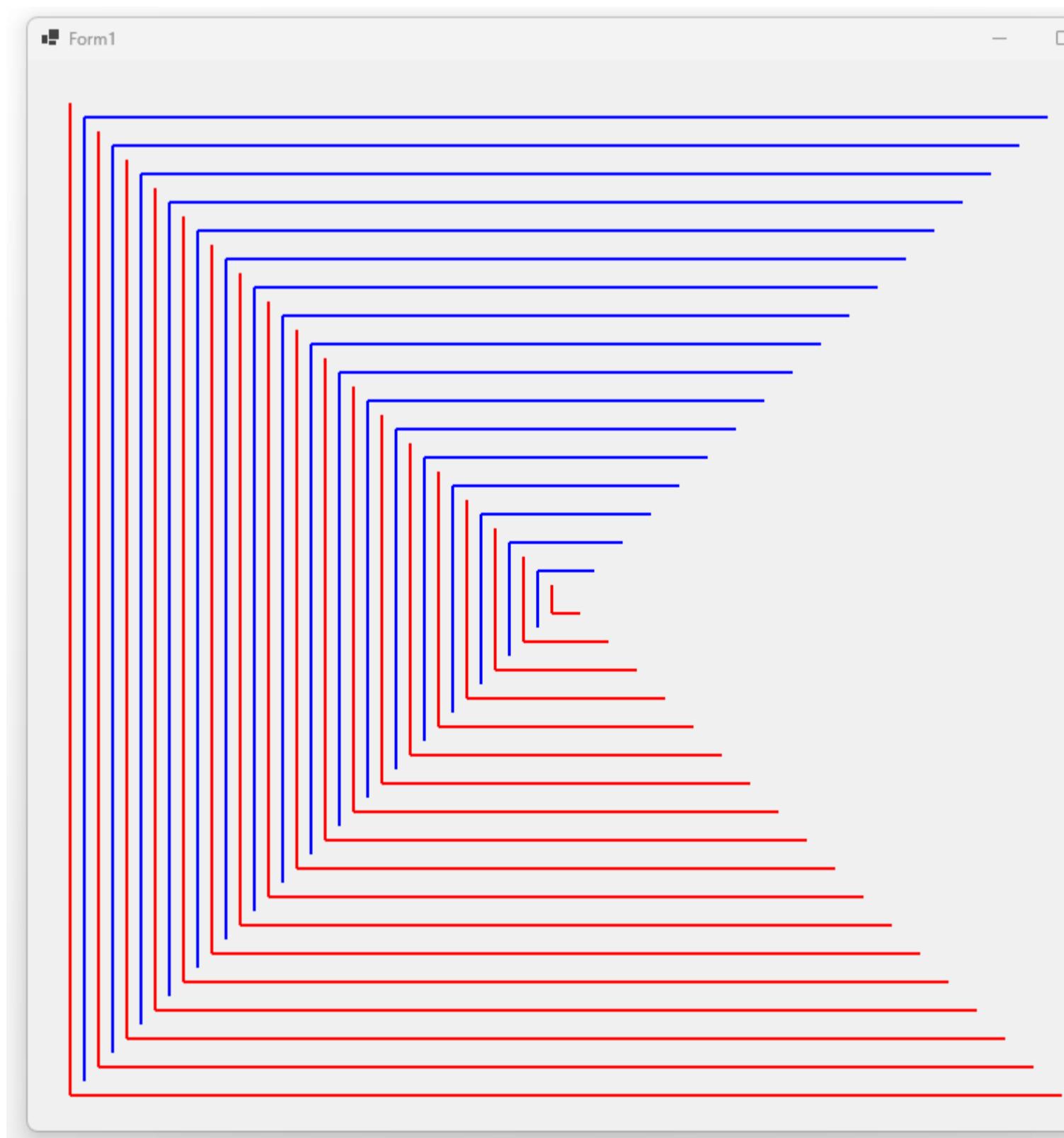


Ilustración 395: Dibuja ángulos rectos

```
namespace Graficos {
    public partial class Form1 : Form {
        public Form1() {
            InitializeComponent();
        }

        private void Form1_Paint(object sender, PaintEventArgs e) {
            Graphics grafico = e.Graphics;
            Pen lapiz = new Pen(Color.Blue, 2);
            Pen lapiz2 = new Pen(Color.Red, 2);
            Pen lapiz3 = new Pen(Color.Green, 2);
            Pen lapiz4 = new Pen(Color.Violet, 2);
            int cont = 0;

            do {
                grafico.DrawLine(lapiz, cont, 600, 600, cont);
                grafico.DrawLine(lapiz2, 1200 - cont, 600, 600, cont);
                grafico.DrawLine(lapiz3, 1200 - cont, 0, 600, 600 - cont);
                grafico.DrawLine(lapiz4, cont, 0, 600, 600 - cont);
                cont += 20;
            }
            while (cont <= 600);
        }
    }
}
```

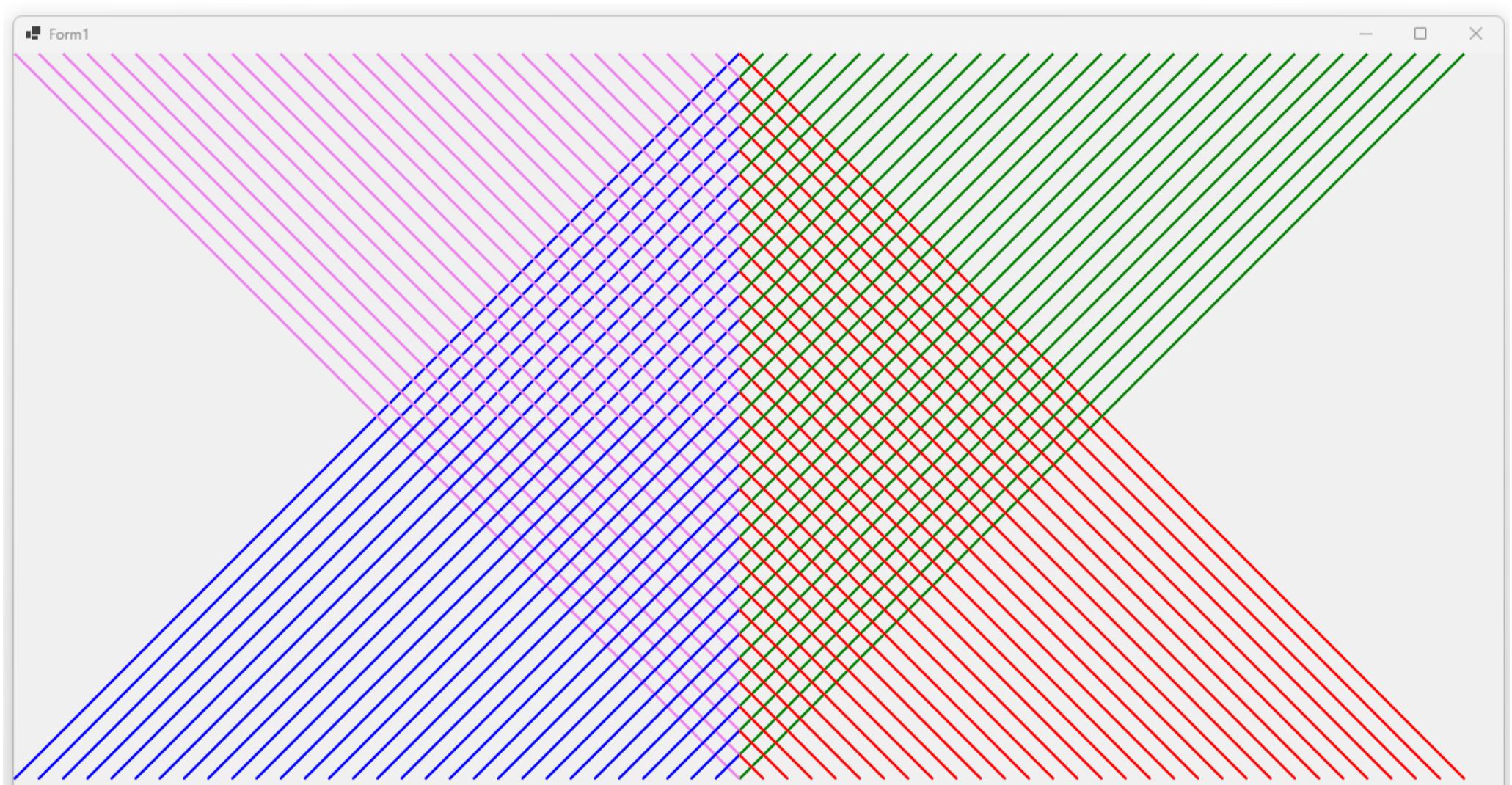


Ilustración 396: Líneas diagonales cruzadas

```
namespace Graficos {
    public partial class Form1 : Form {
        public Form1() {
            InitializeComponent();
        }

        private void Form1_Paint(object sender, PaintEventArgs e) {
            // Necesario para los gráficos GDI+
            Graphics grafico = e.Graphics;

            // Lápiz para el color de la línea y con un ancho de 1
            Pen lapiz = new Pen(Color.Blue, 1);

            for (int Fila = 0; Fila <= 10; Fila++) {
                for (int Columna = 0; Columna <= 10; Columna++)
                    grafico.DrawRectangle(lapiz, Fila * 35 + 80, Columna * 35 + 40, 30, 30);
            }
        }
}
```

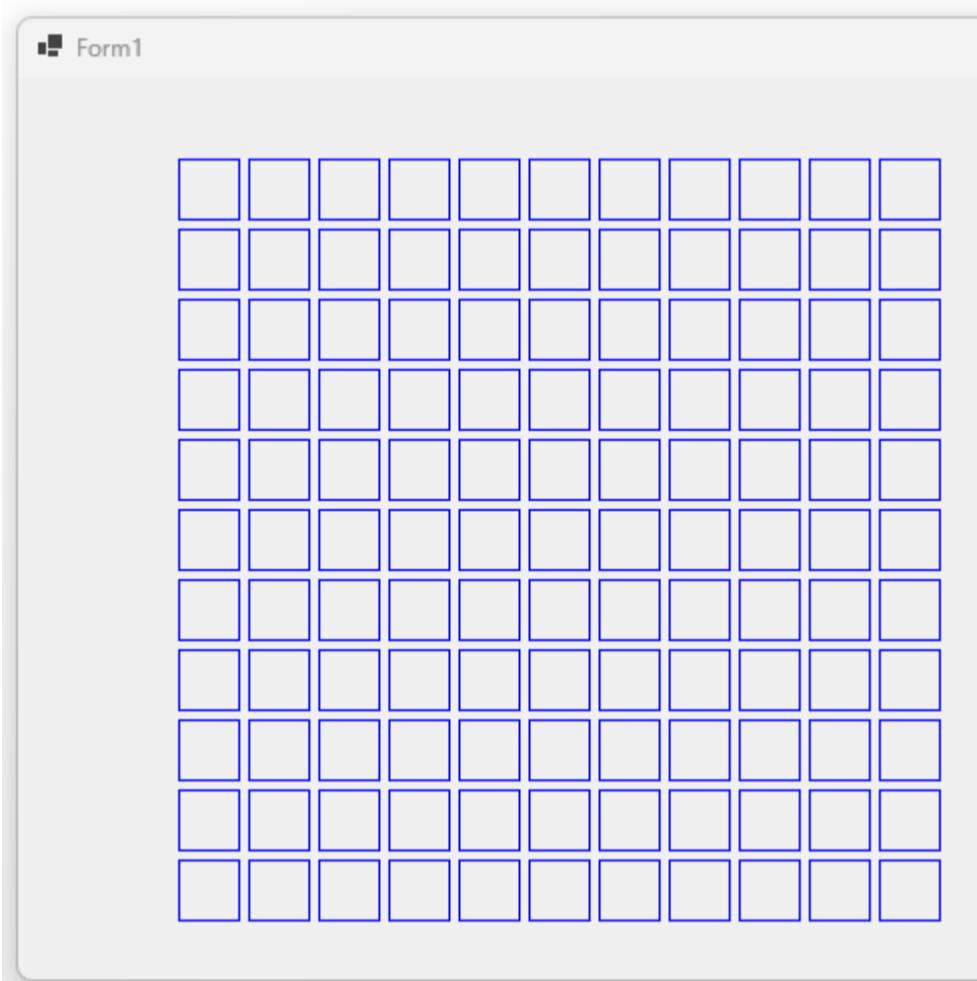


Ilustración 397: Celdas

```
namespace Graficos {
    public partial class Form1 : Form {
        public Form1() {
            InitializeComponent();
        }

        private void Form1_Paint(object sender, PaintEventArgs e) {
            Graphics grafico = e.Graphics;

            // Lápiz para el color de la línea y con un ancho de 1
            Pen lapiz = new Pen(Color.Blue, 1);

            // Parte superior
            for (int Posicion = 0; Posicion <= 200; Posicion += 20) {
                // Línea requiere: Lapiz, X1, Y1, X2, Y2
                grafico.DrawLine(lapiz, 200, Posicion, Posicion + 200, 200);
                grafico.DrawLine(lapiz, 200, Posicion, 200 - Posicion, 200);
            }

            //Parte inferior
            for (int Posicion = 400; Posicion >= 200; Posicion += -20) {
                // Línea requiere: Lapiz, X1, Y1, X2, Y2
                grafico.DrawLine(lapiz, 200, Posicion, 600 - Posicion, 200);
                grafico.DrawLine(lapiz, 200, Posicion, Posicion - 200, 200);
            }
        }
    }
}
```

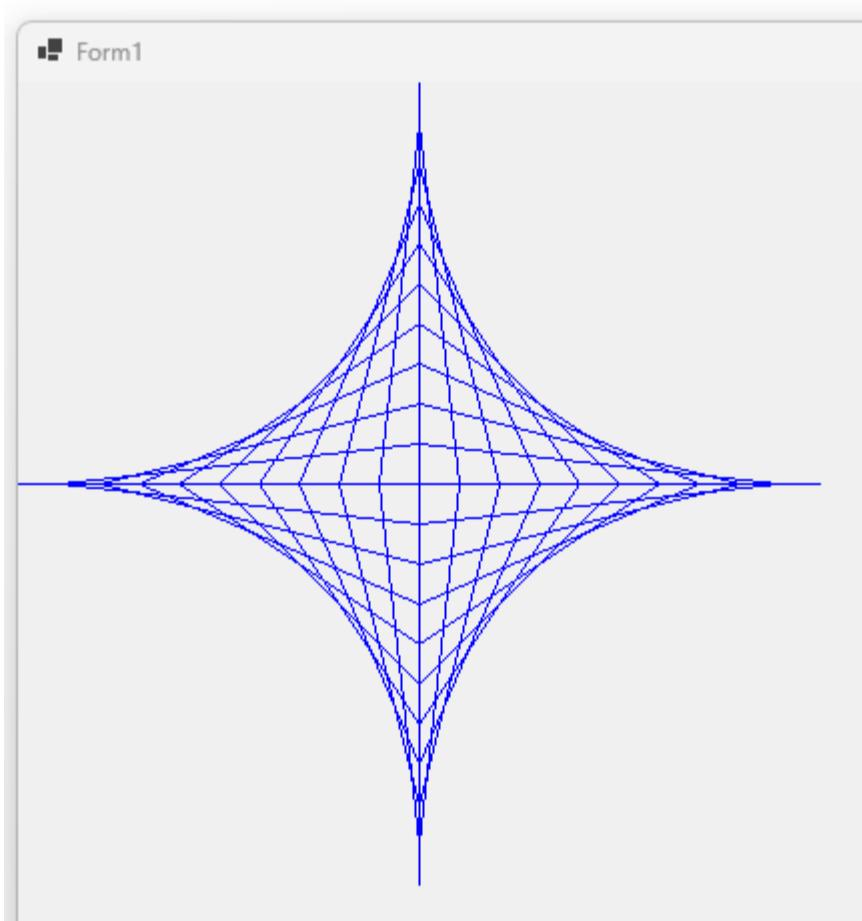


Ilustración 398: Líneas rectas simulan curvas

Algoritmo de Bresenham para dibujar líneas

Este algoritmo dibuja líneas rectas dadas las coordenadas inicial y final. Lo interesante es que no hace operaciones de punto flotante (que sería lento) sino operaciones de enteros (más rápido). Ese es el algoritmo de la función DrawLine

L/033.cs

```
//Algoritmo de Bresenham, para dibujar líneas rectas rápidamente
namespace Graficos {
    public partial class Form1 : Form {

        public Form1() {
            InitializeComponent();
        }

        public void DibujarLineaRecta(Graphics lienzo, int iniX, int iniY, int finX, int finY) {
            int Contador, Distancia;
            int Xerror=0, Yerror=0, CambioX, CambioY, incrementoX, incrementoY;

            CambioX = finX - iniX;
            CambioY = finY - iniY;

            if (CambioX > 0) incrementoX = 1;
            else if (CambioX == 0) incrementoX = 0;
            else incrementoX = -1;

            if (CambioY > 0) incrementoY = 1;
            else if (CambioY == 0) incrementoY = 0;
            else incrementoY = -1;

            if (CambioX < 0) CambioX *= -1;
            if (CambioY < 0) CambioY *= -1;

            if (CambioX > CambioY)
                Distancia = CambioX;
            else
                Distancia = CambioY;

            for (Contador = 0; Contador <= Distancia + 1; Contador++) {
                lienzo.FillRectangle(Brushes.Black, iniX, iniY, 1, 1);
                Xerror += CambioX;
                Yerror += CambioY;
                if (Xerror > Distancia) {
                    Xerror -= Distancia;
                    iniX += incrementoX;
                }

                if (Yerror > Distancia) {
                    Yerror -= Distancia;
                    iniY += incrementoY;
                }
            }
        }

        private void Form1_Paint(object sender, PaintEventArgs e) {
            //Ejemplo
            DibujarLineaRecta(e.Graphics, 16, 83, 197, 206);
        }
    }
}
```



Ilustración 399: Algoritmo de Bresenham para dibujar líneas

Traslado de figuras en un plano

Mover la figura en el eje X y en el eje Y, es simplemente sumar o restar a los valores de las coordenadas:

NuevaPosicionX = PosicionOriginalX + MueveX

NuevaPosicionY = PosicionOriginalY + MueveY

L/034.cs

```
//Traslado de figuras en un plano
namespace Graficos {
    public partial class Form1 : Form {

        public Form1() {
            InitializeComponent();
        }

        private void Form1_Paint(object sender, PaintEventArgs e) {
            //Traslado horizontal
            int mueveX = 0;

            //Traslado vertical
            int mueveY = 0;

            //Dibuja un rectángulo con cuatro líneas
            e.Graphics.DrawLine(Pens.Black, 10 + mueveX, 10 + mueveY, 400 + mueveX, 10 + mueveY);
            e.Graphics.DrawLine(Pens.Black, 10 + mueveX, 250 + mueveY, 400 + mueveX, 250 + mueveY);
            e.Graphics.DrawLine(Pens.Black, 400 + mueveX, 10 + mueveY, 400 + mueveX, 250 + mueveY);
            e.Graphics.DrawLine(Pens.Black, 10 + mueveX, 10 + mueveY, 10 + mueveX, 250 + mueveY);
        }
    }
}
```

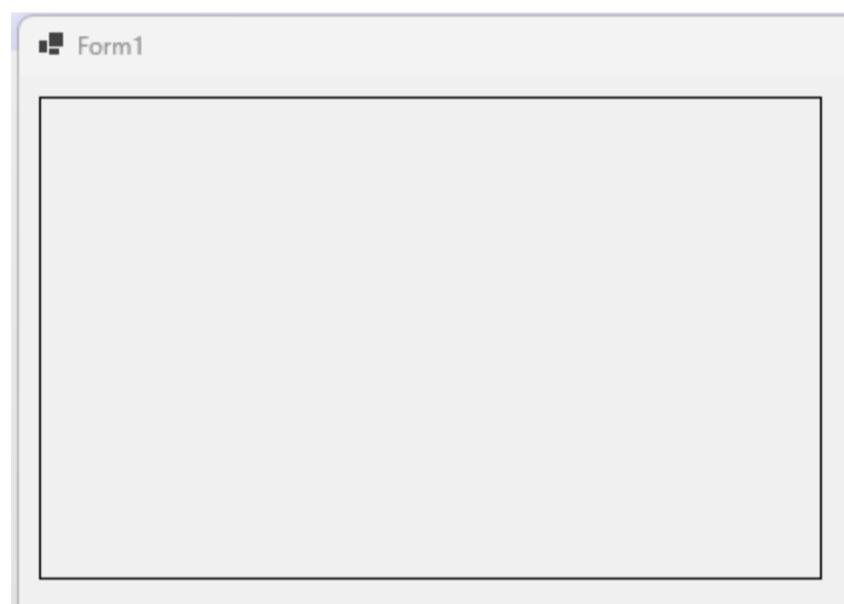


Ilustración 400: Traslado de figuras en un plano

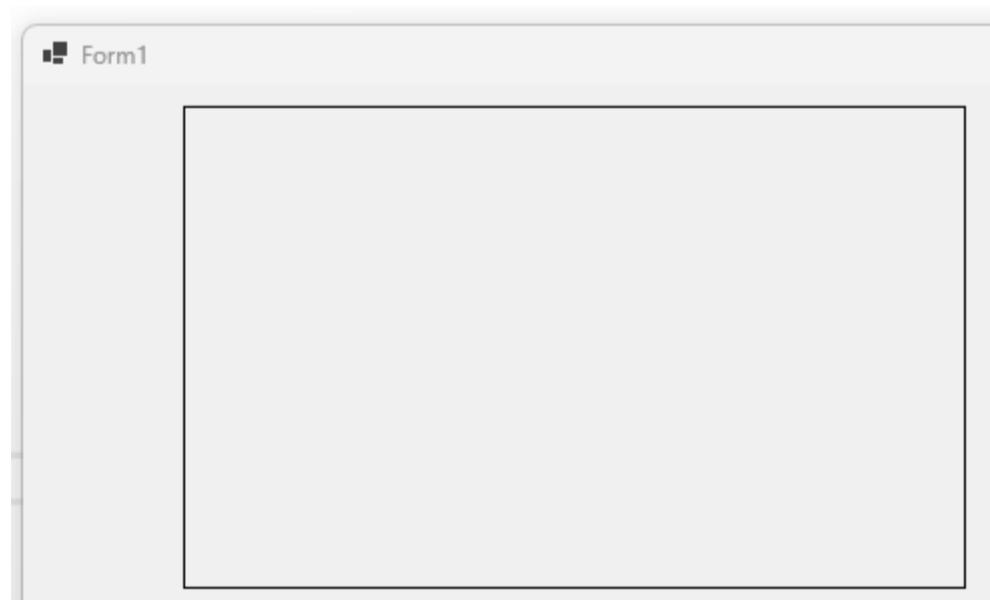


Ilustración 401: Traslado de figuras en un plano mueveX = 70



Ilustración 402: Traslado de figuras en un plano mueveY = 70



Ilustración 403: Traslado de figuras en un plano mueveX = 70 y mueveY = 70

Giro de figuras en un plano

Para el giro en determinado ángulo, se usa la siguiente fórmula para una coordenada plana:

$$\text{PosicionNuevaX} = \text{PosicionOriginalX} * \cos(\text{Angulo}) - \text{PosicionOriginalY} * \operatorname{sen}(\text{Angulo})$$

$$\text{PosicionNuevaY} = \text{PosicionOriginalX} * \operatorname{sen}(\text{Angulo}) + \text{PosicionOriginalY} * \cos(\text{Angulo})$$

L/035.cs

```
//Giro de figuras en un plano
using System;
using System.Drawing;
using System.Windows.Forms;

namespace Graficos {
    public partial class Form1 : Form {

        public Form1() {
            InitializeComponent();
        }

        private void Form1_Paint(object sender, PaintEventArgs e) {
            //Coordenadas de la figura
            int posXA, posYA, posXb, posYb, posXc, posYc;

            posXA = 80;
            posYA = 80;
            posXb = 400;
            posYb = 300;
            posXc = 500;
            posYc = 200;

            //Dibuja un triángulo con tres líneas
            e.Graphics.DrawLine(Pens.Black, posXA, posYA, posXb, posYb);
            e.Graphics.DrawLine(Pens.Black, posXb, posYb, posXc, posYc);
            e.Graphics.DrawLine(Pens.Black, posXc, posYc, posXA, posYA);

            //Ángulo de giro
            int AnguloGiro = 15;
            double AnguloRadianes = AnguloGiro * Math.PI / 180;

            //Cálcula el giro
            int posXga = Convert.ToInt32(posXA * Math.Cos(AnguloRadianes) - posYA * Math.Sin(AnguloRadianes));
            int posYga = Convert.ToInt32(posXA * Math.Sin(AnguloRadianes) + posYA * Math.Cos(AnguloRadianes));

            int posXgb = Convert.ToInt32(posXb * Math.Cos(AnguloRadianes) - posYb * Math.Sin(AnguloRadianes));
            int posYgb = Convert.ToInt32(posXb * Math.Sin(AnguloRadianes) + posYb * Math.Cos(AnguloRadianes));

            int posXgc = Convert.ToInt32(posXc * Math.Cos(AnguloRadianes) - posYc * Math.Sin(AnguloRadianes));
            int posYgc = Convert.ToInt32(posXc * Math.Sin(AnguloRadianes) + posYc * Math.Cos(AnguloRadianes));

            //Dibuja el triángulo con el giro
            e.Graphics.DrawLine(Pens.Red, posXga, posYga, posXgb, posYgb);
            e.Graphics.DrawLine(Pens.Red, posXgb, posYgb, posXgc, posYgc);
            e.Graphics.DrawLine(Pens.Red, posXgc, posYgc, posXga, posYga);
        }
    }
}
```

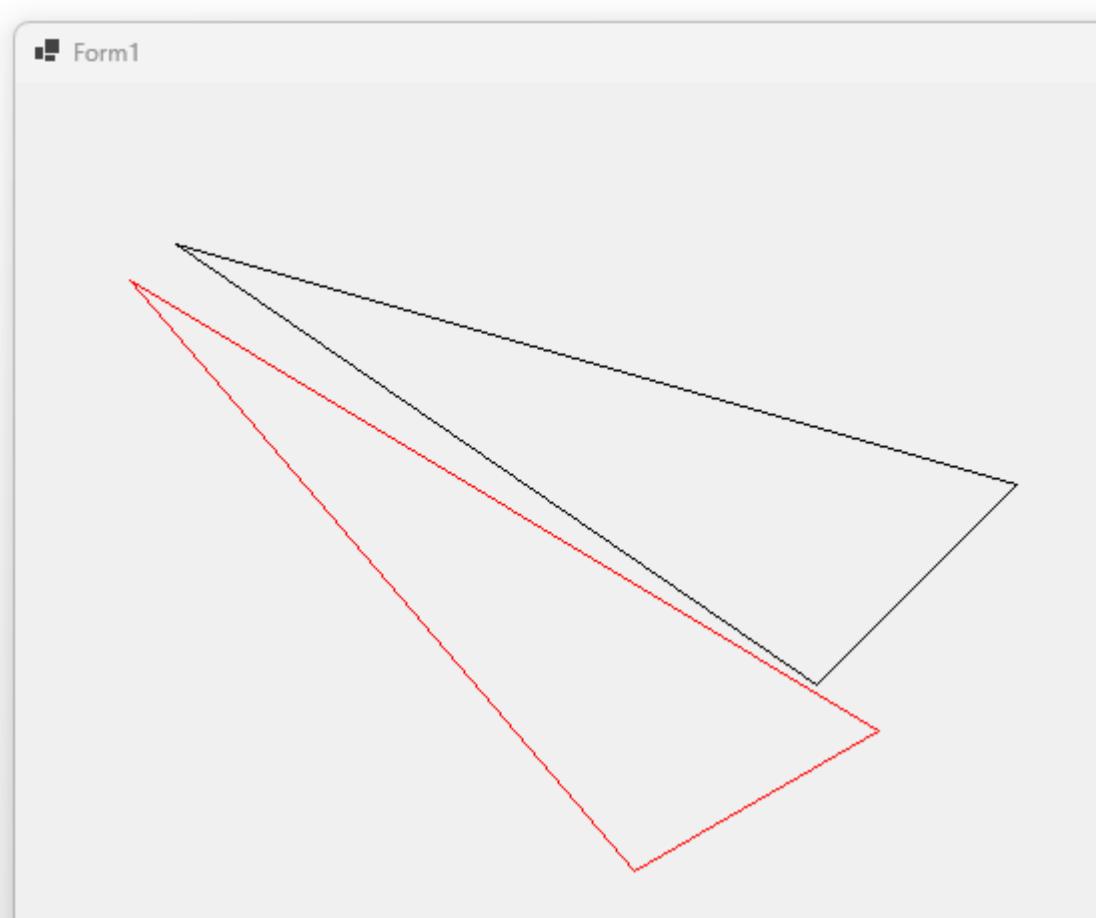


Ilustración 404: Giro de figuras en un plano

En el ejemplo, se usó un giro de 5 grados (que se deben convertir en radianes antes de hacer uso de las funciones de seno y coseno). El ángulo se aplica desde la posición 0,0 de la ventana.

Giro de figuras en un plano calculando el centroide

Se requiere calcular el centroide de la figura (fácil si es un rectángulo), se aplica el giro y se compara cuánto se desplazó en X y Y con respecto a la posición original. Una vez obtenidos esos datos, se restaura para toda la figura y se obtiene el giro sobre sí misma.

L/036.cs

```
//Giro de figuras en un plano calculando el centroide
using System;
using System.Drawing;
using System.Windows.Forms;

namespace Graficos {
    public partial class Form1 : Form {

        public Form1() {
            InitializeComponent();
        }

        private void Form1_Paint(object sender, PaintEventArgs e) {
            //Datos del rectángulo
            int posXa, posYa, Largo, Alto;
            posXa = 80;
            posYa = 80;
            Largo = 400;
            Alto = 180;

            //Deduce las otras tres coordenadas del rectángulo
            int posXb, posYb, posXc, posYc, posXd, posYd;
            posXb = posXa + Largo;
            posYb = posYa;
            posXc = posXa + Largo;
            posYc = posYa + Alto;
            posXd = posXa;
            posYd = posYa + Alto;

            //Dibuja un rectángulo con cuatro líneas
            e.Graphics.DrawLine(Pens.Black, posXa, posYa, posXb, posYb);
            e.Graphics.DrawLine(Pens.Black, posXb, posYb, posXc, posYc);
            e.Graphics.DrawLine(Pens.Black, posXc, posYc, posXd, posYd);
            e.Graphics.DrawLine(Pens.Black, posXd, posYd, posXa, posYa);

            //Ángulo de giro
            int AnguloGiro = 15;
            double AnguloRadianes = AnguloGiro * Math.PI / 180;

            //Centroide
            int posXcentro = posXa + Largo / 2;
            int posYcentro = posYa + Alto / 2;

            //Cálcula el giro
            int posXga = Convert.ToInt32(posXa * Math.Cos(AnguloRadianes) - posYa * Math.Sin(AnguloRadianes));
            int posYga = Convert.ToInt32(posXa * Math.Sin(AnguloRadianes) + posYa * Math.Cos(AnguloRadianes));

            int posXgb = Convert.ToInt32(posXb * Math.Cos(AnguloRadianes) - posYb * Math.Sin(AnguloRadianes));
            int posYgb = Convert.ToInt32(posXb * Math.Sin(AnguloRadianes) + posYb * Math.Cos(AnguloRadianes));

            int posXgc = Convert.ToInt32(posXc * Math.Cos(AnguloRadianes) - posYc * Math.Sin(AnguloRadianes));
            int posYgc = Convert.ToInt32(posXc * Math.Sin(AnguloRadianes) + posYc * Math.Cos(AnguloRadianes));

            int posXgd = Convert.ToInt32(posXd * Math.Cos(AnguloRadianes) - posYd * Math.Sin(AnguloRadianes));
            int posYgd = Convert.ToInt32(posXd * Math.Sin(AnguloRadianes) + posYd * Math.Cos(AnguloRadianes));

            //Giro del centroide
            int posXgcentro = Convert.ToInt32(posXcentro * Math.Cos(AnguloRadianes) - posYcentro * Math.Sin(AnguloRadianes));
            int posYgcentro = Convert.ToInt32(posXcentro * Math.Sin(AnguloRadianes) + posYcentro * Math.Cos(AnguloRadianes));

            //¿Cuánto se desplazó el centroide?
            int desplazaX = posXgcentro - posXcentro;
            int desplazaY = posYgcentro - posYcentro;
        }
    }
}
```

```

    //Dibuja el triángulo con el giro
    e.Graphics.DrawLine(Pens.Red, posXga - desplazaX, posYga - desplazaY, posXgb - desplazaX,
posYgb - desplazaY);
    e.Graphics.DrawLine(Pens.Red, posXgb - desplazaX, posYgb - desplazaY, posXgc - desplazaX,
posYgc - desplazaY);
    e.Graphics.DrawLine(Pens.Red, posXgc - desplazaX, posYgc - desplazaY, posXgd - desplazaX,
posYgd - desplazaY);
    e.Graphics.DrawLine(Pens.Red, posXga - desplazaX, posYga - desplazaY, posXgd - desplazaX,
posYgd - desplazaY);
}
}
}

```

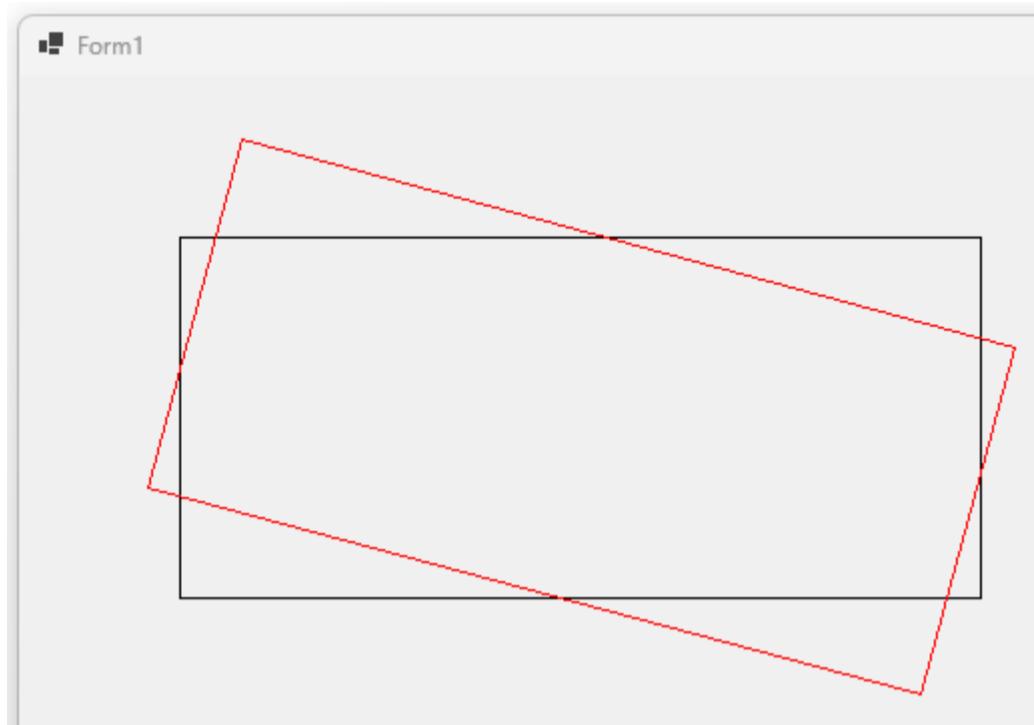


Ilustración 405: Giro de figuras en un plano calculando el centroide

Gráfico matemático en 2D

Dada una ecuación del tipo $Y=F(X)$, por ejemplo:

$$Y = 0.1X^6 + 0.6X^5 - 0.7X^4 - 5.7X^3 + 2X^2 + 2X + 1$$

Para graficarla, se deben hacer estos pasos:

1. Saber el valor de inicio de X y el valor final de X. Se llamarán X_{ini} y X_{fin} respectivamente.
2. Saber cuántos puntos se van a calcular. Se llamará $numPuntos$.
3. Con esos datos, se hace uso de un ciclo que calcule los valores de Y. Se almacena el par X, Y
4. Debido a que el eje Y aumenta hacia abajo en una pantalla o ventana, habrá que darles la vuelta a los valores de Y calculados, es decir, multiplicarlos por -1. Luego realmente se almacena $X, -Y$
5. Se requieren cuatro datos:
 - a. El mínimo valor de X. Es el mismo X_{ini} .
 - b. El máximo valor de X. Se llamará $maximoXreal$ que muy probablemente difiera de X_{fin} .
 - c. El mínimo valor de Y obtenido. Se llamará Y_{min} .
 - d. El máximo valor de Y obtenido. Se llamará Y_{max} .
6. También se requieren estos dos datos para poder ajustar el gráfico matemático a un área rectangular definida en la ventana.
 - a. Coordenada superior izquierda en pantalla. Serán las coordenadas enteras positivas $X_{pantallaIni}, Y_{pantallaIni}$
 - b. Coordenada inferior derecha en pantalla. Serán las coordenadas enteras positivas $X_{pantallaFin}, Y_{pantallaFin}$
7. Se calculan unas constantes de conversión con estas fórmulas:
 - a. $convertirX = (X_{pantallaFin} - X_{pantallaIni}) / (maximoXreal - X_{ini})$
 - b. $convertirY = (Y_{pantallaFin} - Y_{pantallaIni}) / (Y_{max} - Y_{min})$
8. Tomar cada coordenada calculada de la ecuación (valor, valorY) y hacerle la conversión a pantalla plana con la siguiente fórmula:
 - a. $pantallaX = convertirX * (valorX - X_{ini}) + X_{pantallaIni}$
 - b. $pantallaY = convertirY * (valorY - Y_{min}) + Y_{pantallaIni}$
9. Se grafican esos puntos y se unen con líneas.

A continuación, el código que se encuentra en un archivo .zip porque es un proyecto escrito en Microsoft Visual Studio 2022. La clase Puntos es para almacenar los valores reales de la ecuación (valor, valorY) y los valores convertidos para que cuadren en pantalla (pantallaX, pantallaY)

L/037.cs

```
//Gráfico matemático en 2D
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Windows.Forms;

namespace Graficos {
    internal class Puntos {
        //Valor X, Y reales de la ecuación
        public double valorX, valorY;

        //Puntos convertidos a coordenadas enteras de pantalla
        public int pantallaX, pantallaY;

        public Puntos(double valorX, double valorY) {
            this.valorX = valorX;
            this.valorY = valorY;
        }
    }

    public partial class Form1 : Form {
        List<Puntos> puntos;
        int XpantallaIni, YpantallaIni, XpantallaFin, YpantallaFin;

        public Form1() {
            InitializeComponent();
            puntos = new List<Puntos>();

            //Área donde se dibujará el gráfico matemático
            XpantallaIni = 20;
            YpantallaIni = 20;
            XpantallaFin = 600;
            YpantallaFin = 400;

            //Algoritmo que calcula los puntos del gráfico
            double minX = -5;
            double maxX = 3;
        }
    }
}
```

```

        int numPuntos = 80;
        Logica(minX, maxX, numPuntos);
    }

    public void Logica(double Xini, double Xfin, int numPuntos) {
        //Calcula los puntos de la ecuación a graficar
        double pasoX = (Xfin - Xini) / numPuntos;
        double Ymin = double.MaxValue; //El mínimo valor de Y obtenido
        double Ymax = double.MinValue; //El máximo valor de Y obtenido
        double maximoXreal = double.MinValue; //El máximo valor de X (difiere de Xfin)

        puntos.Clear();
        for (double X = Xini; X <= Xfin; X += pasoX) {
            double valY = -1*Ecuacion(X); //Se invierte el valor porque el eje Y aumenta hacia
abajo
            if (valY > Ymax) Ymax = valY;
            if (valY < Ymin) Ymin = valY;
            if (X > maximoXreal) maximoXreal = X;
            puntos.Add(new Puntos(X, valY));
        }
        //¡OJO! X puede que no llegue a ser Xfin, por lo que la variable maximoXreal almacena el
valor máximo de X

        //Calcula los puntos a poner en la pantalla
        double convierteX = (XpantallaFin - XpantallaIni) / (maximoXreal - Xini);
        double convierteY = (YpantallaFin - YpantallaIni) / (Ymax - Ymin);

        for (int cont = 0; cont < puntos.Count; cont++) {
            puntos[cont].pantallaX = Convert.ToInt32(convierteX * (puntos[cont].valorX - Xini) +
XpantallaIni);
            puntos[cont].pantallaY = Convert.ToInt32(convierteY * (puntos[cont].valorY - Ymin) +
YpantallaIni);
        }
    }

    //Aquí está la ecuación que se desee graficar con variable independiente X
    public double Ecuacion(double X) {
        return 0.1*Math.Pow(X, 6)+0.6*Math.Pow(X, 5)-0.7*Math.Pow(X, 4)-5.7*Math.Pow(X, 3)+2*X*X+2*X+1;
    }

    //Pinta la ecuación
    private void Form1_Paint(object sender, PaintEventArgs e) {
        Graphics lienzo = e.Graphics;
        Pen lapiz = new Pen(Color.Blue, 3);

        //Un recuadro para ver el área del gráfico
        lienzo.FillRectangle(Brushes.Black, XpantallaIni, YpantallaIni, XpantallaFin-XpantallaIni,
YpantallaFin-YpantallaIni);

        //Dibuja el gráfico matemático
        for (int cont = 0; cont < puntos.Count - 1; cont++) {
            lienzo.DrawLine(lapiz, puntos[cont].pantallaX, puntos[cont].pantallaY, puntos[cont +
1].pantallaX, puntos[cont + 1].pantallaY);
        }
    }
}

```

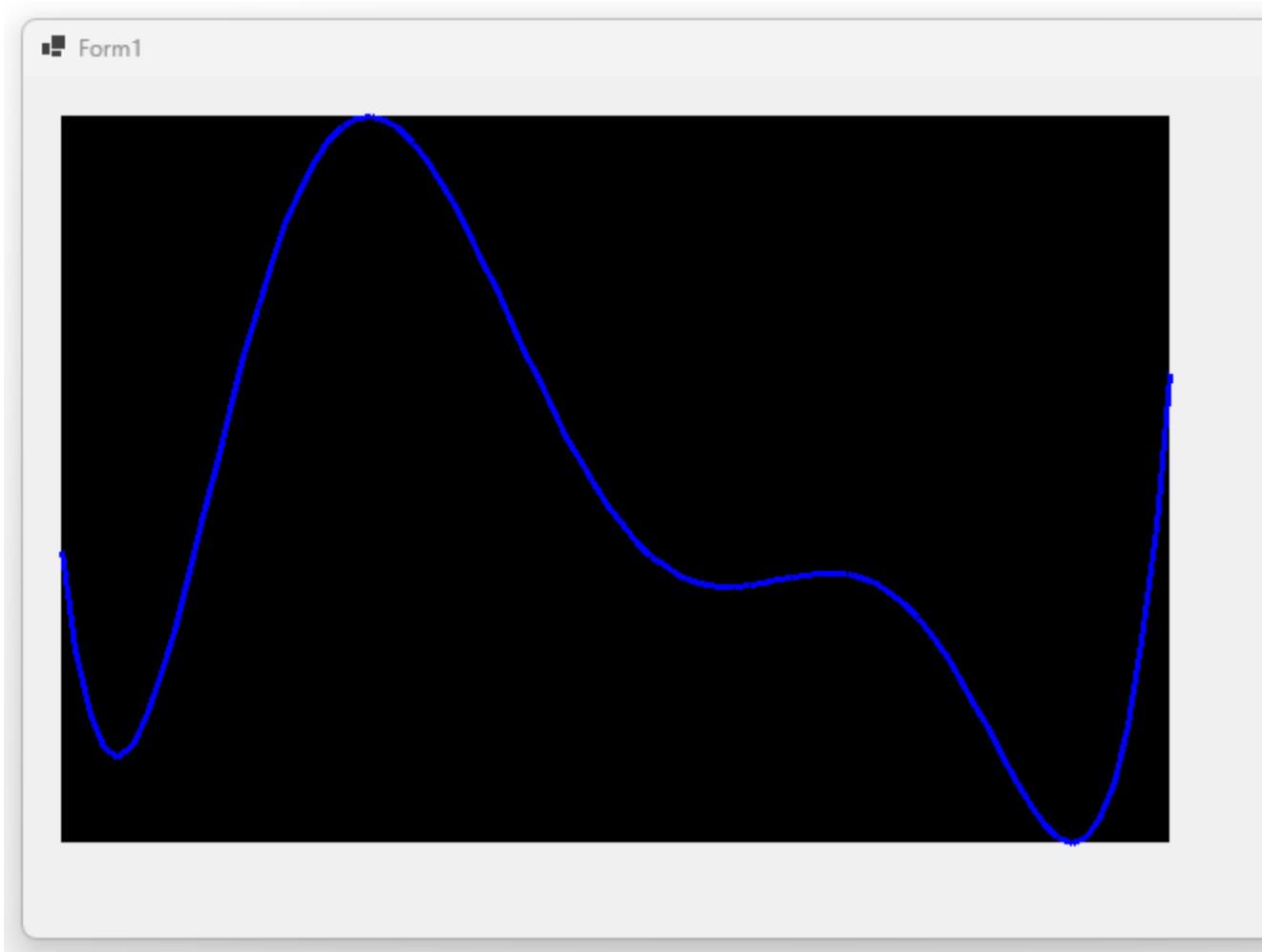


Ilustración 406: Gráfico matemático en 2D

Gráfico polar

Dada una ecuación del tipo $r=F(\Theta)$, donde Θ es el ángulo y r el radio, por ejemplo:

$$r = 2 * \operatorname{seno}(4 * \theta * \pi / 180)$$

Se procede a graficarlo de esta forma:

1. Saber el valor de inicio de Θ y el valor final de Θ . Se llamarán minTheta y maxTheta respectivamente.
2. Saber cuántos puntos se van a calcular. Se llamará numPuntos.
3. Con esos datos, se hace uso de un ciclo que calcule los valores de r . Luego se hace esta conversión:

```
r = Ecuacion(theta);
X = r * Math.Cos (theta * Math.PI / 180);
Y = r * Math.Sin (theta * Math.PI / 180);
```

4. Debido a que el eje Y aumenta hacia abajo en una pantalla o ventana, habrá que darles la vuelta a los valores de Y calculados, es decir, multiplicarlos por -1. Luego realmente se almacena X, -Y
5. Se requieren cuatro datos:
 - a. El mínimo valor de X. Es el mismo Xinia.
 - b. El máximo valor de X. Se llamará maximoXreal que muy probablemente difiera de Xfin.
 - c. El mínimo valor de Y obtenido. Se llamará Ymin.
 - d. El máximo valor de Y obtenido. Se llamará Ymax.
6. También se requieren estos dos datos para poder ajustar el gráfico matemático a un área rectangular definida en la ventana.
 - a. Coordenada superior izquierda en pantalla. Serán las coordenadas enteras positivas XpantallaIni, YpantallaIni
 - b. Coordenada inferior derecha en pantalla. Serán las coordenadas enteras positivas XpantallaFin, YpantallaFin
7. Se calculan unas constantes de conversión con estas fórmulas:
 - a. convierteX = (XpantallaFin - XpantallaIni) / (maximoXreal - Xinia)
 - b. convierteY = (YpantallaFin - YpantallaIni) / (Ymax - Ymin)
8. Tomar cada coordenada calculada de la ecuación (valor, valorY) y hacerle la conversión a pantalla plana con la siguiente fórmula:
 - a. pantallaX = convierteX * (valorX - Xinia) + XpantallaIni
 - b. pantallaY = convierteY * (valorY - Ymin) + YpantallaIni
9. Se grafican esos puntos y se unen con líneas.

A continuación, el código que se encuentra en un archivo .zip porque es un proyecto escrito en Microsoft Visual Studio 2022. La clase Puntos es para almacenar los valores reales de la ecuación (valor, valorY) y los valores convertidos para que cuadren en pantalla (pantallaX, pantallaY)

L/038.cs

```
//Gráfico polar
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Windows.Forms;

namespace Graficos {
    internal class Puntos {
        //Valor X, Y reales de la ecuación
        public double valorX, valorY;

        //Puntos convertidos a coordenadas enteras de pantalla
        public int pantallaX, pantallaY;

        public Puntos(double valorX, double valorY) {
            this.valorX = valorX;
            this.valorY = valorY;
        }
    }

    public partial class Form1 : Form {
        List<Puntos> puntos;
        int XpantallaIni, YpantallaIni, XpantallaFin, YpantallaFin;

        public Form1() {
            InitializeComponent();
            puntos = new List<Puntos>();

            //Área donde se dibujará el gráfico matemático
            XpantallaIni = 20;
```

```

YpantallaIni = 20;
XpantallaFin = 600;
YpantallaFin = 400;

//Algoritmo que calcula los puntos del gráfico
double minTheta = 0;
double maxTheta = 360;
int numPuntos = 200;
Logica(minTheta, maxTheta, numPuntos);
}

public void Logica(double thetaIni, double thetaFin, int numPuntos) {
    //Calcula los puntos de la ecuación a graficar
    double pasoTheta = (thetaFin - thetaIni) / numPuntos;
    double Ymin = double.MaxValue; //El mínimo valor de Y
    double Ymax = double.MinValue; //El máximo valor de Y
    double Xmin = double.MaxValue; //El máximo valor de X
    double Xmax = double.MinValue; //El máximo valor de X

    puntos.Clear();
    for (double theta = thetaIni; theta <= thetaFin; theta += pasoTheta) {
        double valorR = Ecuacion(theta);
        double X = valorR * Math.Cos(theta * Math.PI / 180);
        double Y = -1 * valorR * Math.Sin(theta * Math.PI / 180);

        if (Y > Ymax) Ymax = Y;
        if (Y < Ymin) Ymin = Y;
        if (X > Xmax) Xmax = X;
        if (X < Xmin) Xmin = X;
        puntos.Add(new Puntos(X, Y));
    }

    //Calcula los puntos a poner en la pantalla
    double convierteX = (XpantallaFin - XpantallaIni) / (Xmax - Xmin);
    double convierteY = (YpantallaFin - YpantallaIni) / (Ymax - Ymin);

    for (int cont = 0; cont < puntos.Count; cont++) {
        puntos[cont].pantallaX = Convert.ToInt32(convierteX * (puntos[cont].valorX - Xmin) +
XpantallaIni);
        puntos[cont].pantallaY = Convert.ToInt32(convierteY * (puntos[cont].valorY - Ymin) +
YpantallaIni);
    }
}

//Aquí está la ecuación polar que se desee graficar con variable Theta
public double Ecuacion(double Theta) {
    return 2 * Math.Sin(4 * (Theta * Math.PI / 180));
}

//Pinta la ecuación
private void Form1_Paint(object sender, PaintEventArgs e) {
    Graphics lienzo = e.Graphics;
    Pen lapiz = new Pen(Color.Blue, 3);

    //Un recuadro para ver el área del gráfico
    lienzo.FillRectangle(Brushes.Black, XpantallaIni, YpantallaIni, XpantallaFin - XpantallaIni,
YpantallaFin - YpantallaIni);

    //Dibuja el gráfico matemático
    for (int cont = 0; cont < puntos.Count - 1; cont++) {
        lienzo.DrawLine(lapiz, puntos[cont].pantallaX, puntos[cont].pantallaY, puntos[cont + 1].pantallaX, puntos[cont + 1].pantallaY);
    }
}
}

```

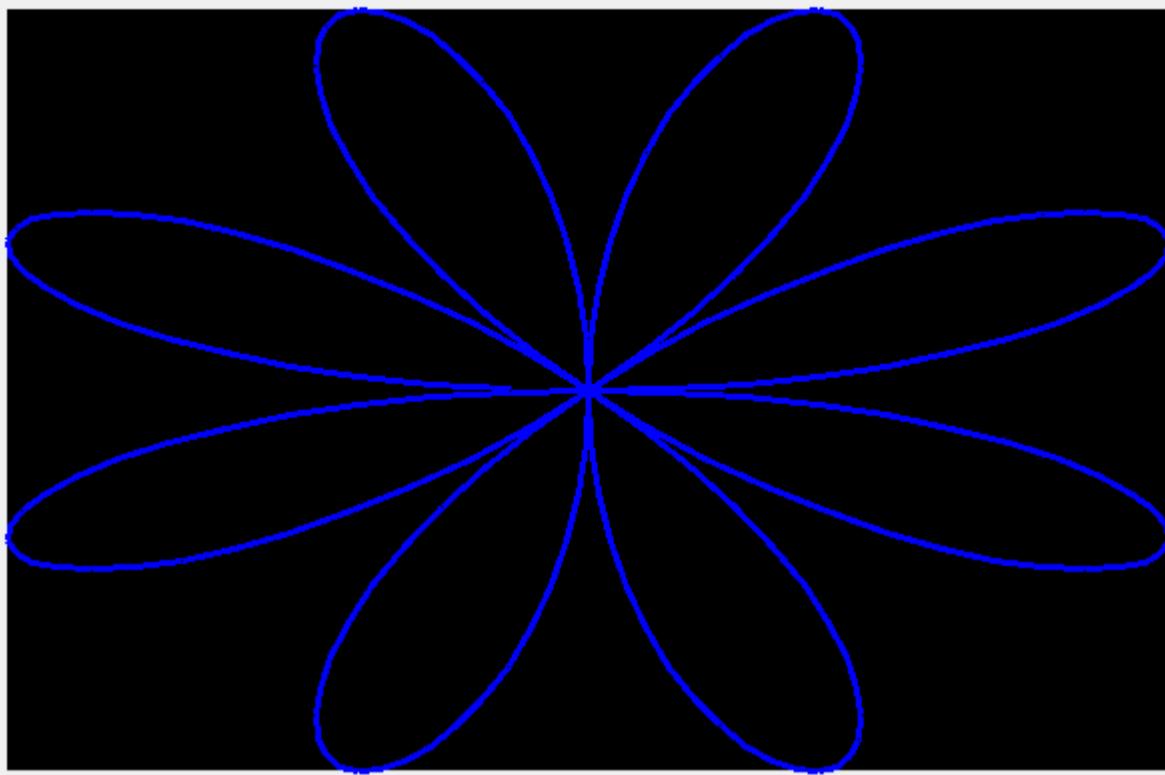


Ilustración 407: Gráfico polar

Bibliografía

- [1] Microsoft Learn, «El sistema de tipos de C#,» febrero 2023. [En línea]. Available: <https://learn.microsoft.com/es-es/dotnet/csharp/fundamentals/types/>. [Último acceso: enero 2024].
- [2] Microsoft, «Visual Studio 2022,» 2024. [En línea]. Available: <https://visualstudio.microsoft.com/es/vs/>. [Último acceso: enero 2024].
- [3] JetBrains, «JetBrains Rider,» 2024. [En línea]. Available: <https://www.jetbrains.com/rider/>. [Último acceso: enero 2024].
- [4] Microsoft, «Create a Windows Forms app in Visual Studio with C#,» enero 2023. [En línea]. Available: <https://learn.microsoft.com/en-us/visualstudio/ide/create-csharp-winform-visual-studio?view=vs-2022>. [Último acceso: febrero 2024].
- [5] Microsoft, «Build beautiful web apps with Blazor,» 2024. [En línea]. Available: <https://dotnet.microsoft.com/en-us/apps/aspnet/web-apps/blazor>. [Último acceso: febrero 2024].
- [6] Godot, «Godot Engine,» 2024. [En línea]. Available: <https://godotengine.org/download/3.x/windows/>. [Último acceso: enero 2024].
- [7] Unity, «CODING IN C# IN UNITY FOR BEGINNERS,» [En línea]. Available: <https://unity.com/how-to/learning-c-sharp-unity-beginners>.
- [8] Microsoft, «Introducción a la compilación de aplicaciones multiplataforma,» 2023. [En línea]. Available: <https://learn.microsoft.com/es-es/xamarin/cross-platform/app-fundamentals/building-cross-platform-applications/overview>. [Último acceso: enero 2024].
- [9] Microsoft, «Interfaz de usuario de aplicaciones multiplataforma de .NET,» 2024. [En línea]. Available: <https://dotnet.microsoft.com/es-es/apps/maui>. [Último acceso: enero 2024].
- [10] Microsoft, «Native AOT deployment,» 2023. [En línea]. Available: <https://learn.microsoft.com/en-us/dotnet/core/deploying/native-aot/?tabs=net7%2Cwindows>. [Último acceso: enero 2024].
- [11] Microsoft, «Opciones del compilador de C# para controlar la generación de código,» 2023. [En línea]. Available: <https://learn.microsoft.com/es-es/dotnet/csharp/language-reference/compiler-options/code-generation>. [Último acceso: enero 2024].
- [12] NDepend, «.NET Native AOT Explained,» diciembre 2023. [En línea]. Available: <https://blog.ndepend.com/net-native-aot-explained/>. [Último acceso: enero 2024].
- [13] Microsoft, «Automatic Memory Management,» abril 2023. [En línea]. Available: <https://learn.microsoft.com/es-es/dotnet/standard/automatic-memory-management>. [Último acceso: febrero 2024].
- [14] Geeks for Geeks, «Garbage Collection in C# | .NET Framework,» 2023. [En línea]. Available: <https://www.geeksforgeeks.org/garbage-collection-in-c-sharp-dot-net-framework/>. [Último acceso: enero 2024].
- [15] Microsoft, «.NET is Free,» 2024. [En línea]. Available: <https://dotnet.microsoft.com/en-us/platform/free>. [Último acceso: enero 2024].
- [16] Microsoft, «ECMA-334. C# language specification,» diciembre 2023. [En línea]. Available: <https://ecma-international.org/publications-and-standards/standards/ecma-334/>. [Último acceso: enero 2024].