MATERIA PROGRAMACIÓN II
ESCUELA: ANÁLISIS DE SISTEMAS
ÚLTIMA ACTUALIZACIÓN: FEBRERO 2019

ELABORADO POR: ING. MARIA NEUS

OBJETIVOS GENERALES

Adquirir los conocimientos y competencias generales que le permitan desarrollar soluciones de software, utilizando la herramienta de programación Visual C#.

CONTENIDO

UNIDAD 1 NOCIONES BÁSICAS DE VISUAL C#
1.1 Conceptos básicos de codificar un programa
1.2 Creación de un proyecto
1.3 Tipos de errores
1.4 Declaración de Variables
1.5 Operadores
1.6 Tipos de datos
UNIDAD 2. ESTRUCTURAS DE PROGRAMACIÓN
2.1 Estructuras secuenciales
2.2 Estructuras condicionales
2.3 Estructuras Condicionales compuestas con operadores lógicos
UNIDAD 3. ESTRUCTURAS REPETITIVAS, CICLOS O BUCLES
3.1 Estructura repetitiva while
3.2 Estructura repetitiva for
3.3 Estructura repetitiva Do While
3.4 Estructura Repetitiva Foreach
UNIDAD 4. CLASES Y OBJETOS
4.1 Clases
4.2 Campos
4.3 Métodos
4.4 Parámetros
4.5 Constructores
4.6 Propiedades
4.7 Indexador
UNIDAD 5. ESTRUCTURAS DE DATOS
5.1 Estructura tipo vector
5.2 Estructura tipo Matríz
UNIDAD 6. ESTRUCTURAS DINÁMICAS
6.1 Generalidades
6.2 Listas
6.3 Listas tipo pila

Instituto Universitario de Tecnología para la Informática –IutepiInstituto Universitario de Tecnología para la Informática - Iutepi

6.4 Listas tipo cola
6.5 Árboles
UNIDAD 7. FORMULARIOS Y CONTROLES
7.1 Windows Forms
7.2 Controles
1.3 Propiedades de los controles
1.4 Manejo de eventos

UNIDAD 1. NOCIONES BÁSICAS DE VISUAL C#

1.1Conceptos básicos para codificar un programa.

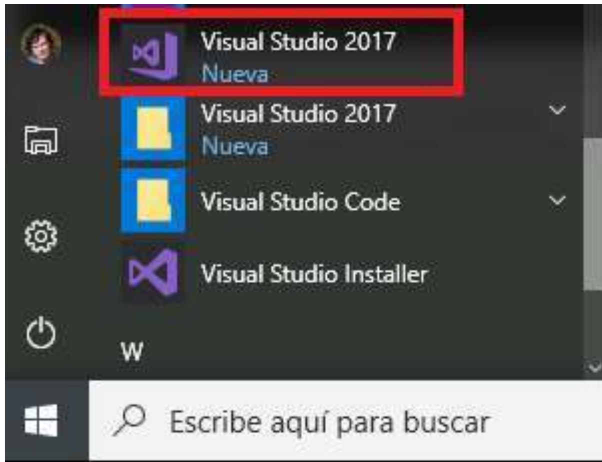
Variable: Es un depósito donde hay un valor. Consta de un nombre y pertenece a un tipo.
Para el ejemplo planteado la variable HorasTrabajadas almacena la cantidad de horas trabajadas por el operario. La variable ValorHora almacena el precio de una hora de trabajo. La variable Sueldo almacena el sueldo a abonar al operario.
En el ejemplo tenemos tres variables.
Tipos de variables:
Una variable puede almacenar:
- Valores Enteros (100, 260, etc.)
- Valores Reales (1.24, 2.90, 5.00, etc.)
- Cadenas de caracteres ("Juan", "Compras", "Listado", etc.)

Elección del nombre de una variable:
Debemos elegir nombres de variables representativas. En el ejemplo el nombre HorasTrabajadas es lo suficientemente claro para darnos una idea acabada sobre su contenido. Podemos darle otros buenos nombres. Otros no son tan representativos, por ejemplo HTr. Posiblemente cuando estemos resolviendo un problema dicho nombre nos recuerde que almacenamos las horas trabajadas por el operario pero cuando pase el tiempo y leamos el diagrama probablemente no recordemos ni entendamos qué significa HTr.

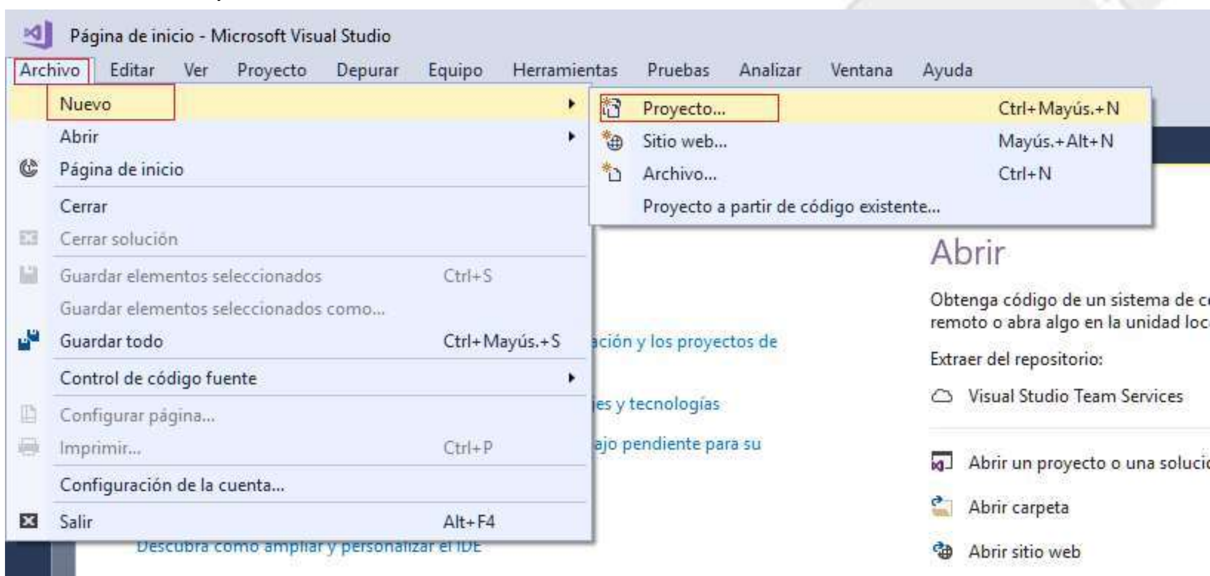
1.2 Creación de un proyecto

Hay que tener en cuenta que el entorno de programación "Microsoft Visual Studio" no ha sido desarrollado pensando en un principiante de la programación. Lo mismo ocurre con el propio lenguaje C#, es decir su origen no tiene como objetivo el aprendizaje de la programación. Debido a estos dos puntos veremos que a medida que avanzamos con el tutorial muchos conceptos que iremos dejando pendientes se irán aclarando.
Veremos los pasos para la creación de un proyecto en C#.
Pasos.
1 - Ingresemos al "Microsoft Visual Studio".

Instituto Universitario de Tecnología para la Informática –IutepiInstituto Universitario de Tecnología para la Informática - Iutepi

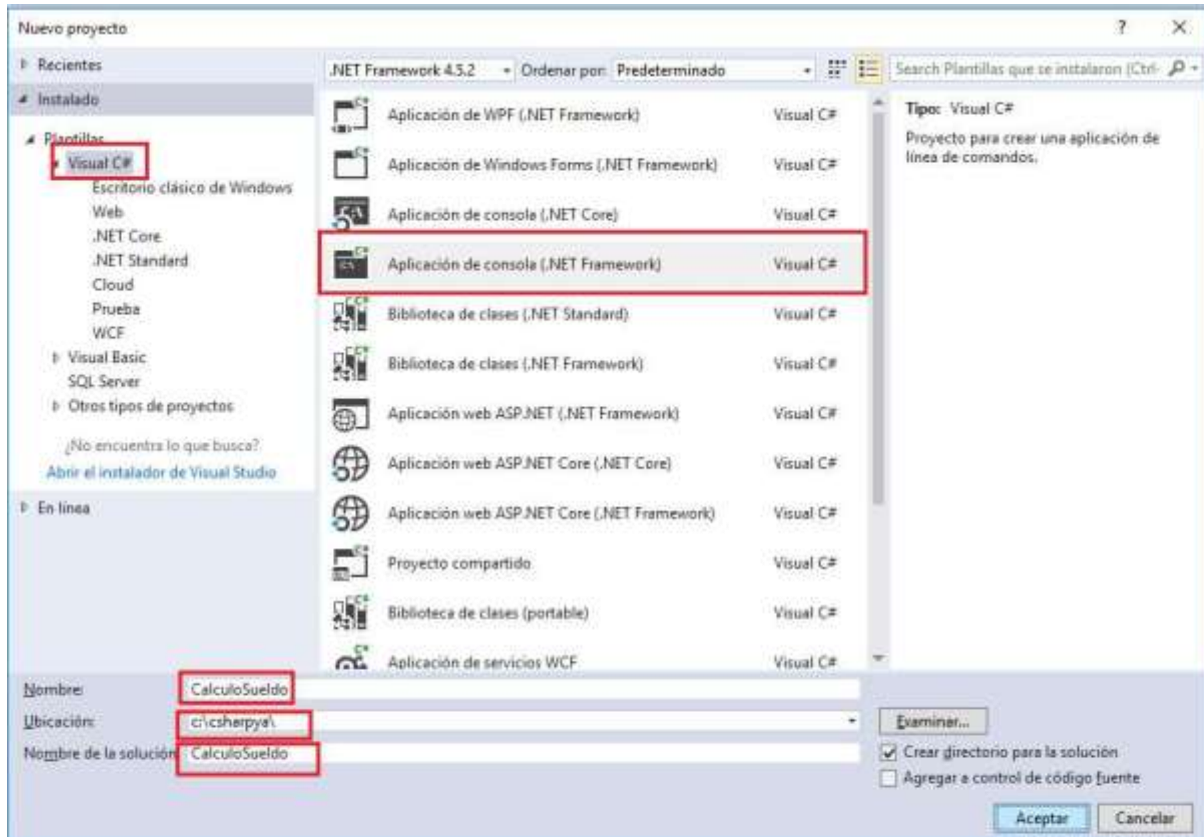


2 - Creación del proyecto. Para esto seleccionamos desde el menú la opción "Archivo" -> "Nuevo" -> "Proyecto..."

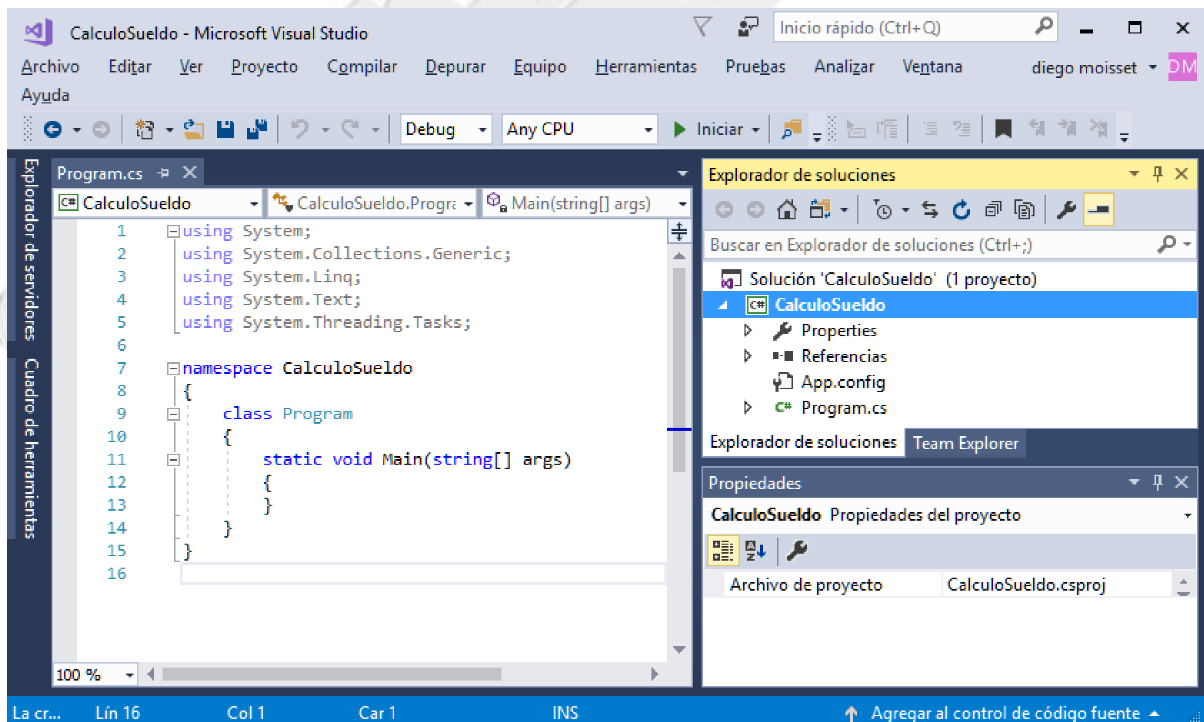


Aparece un diálogo donde debemos indicar del lado izquierdo que utilizaremos el lenguaje Visual C# y del lado de la derecha seleccionamos "Aplicación de consola (.Net Framework)" y en la parte inferior definimos el "nombre", "ubicación" y "nombre de la solución" (podemos usar el mismo texto para el "nombre de la solución" y "nombre"):

Instituto Universitario de Tecnología para la Informática –IutepiInstituto Universitario de Tecnología para la Informática - Iutepi

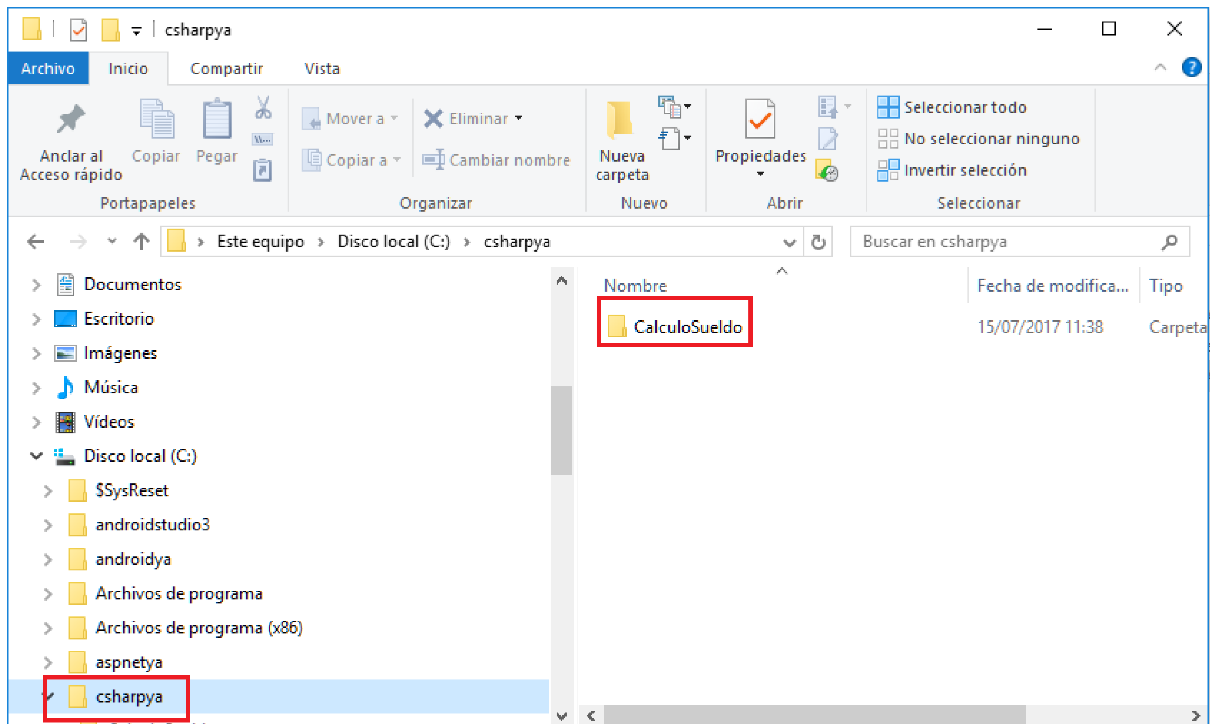


Podemos ver que el entorno nos generó automáticamente el esqueleto de nuestro programa:

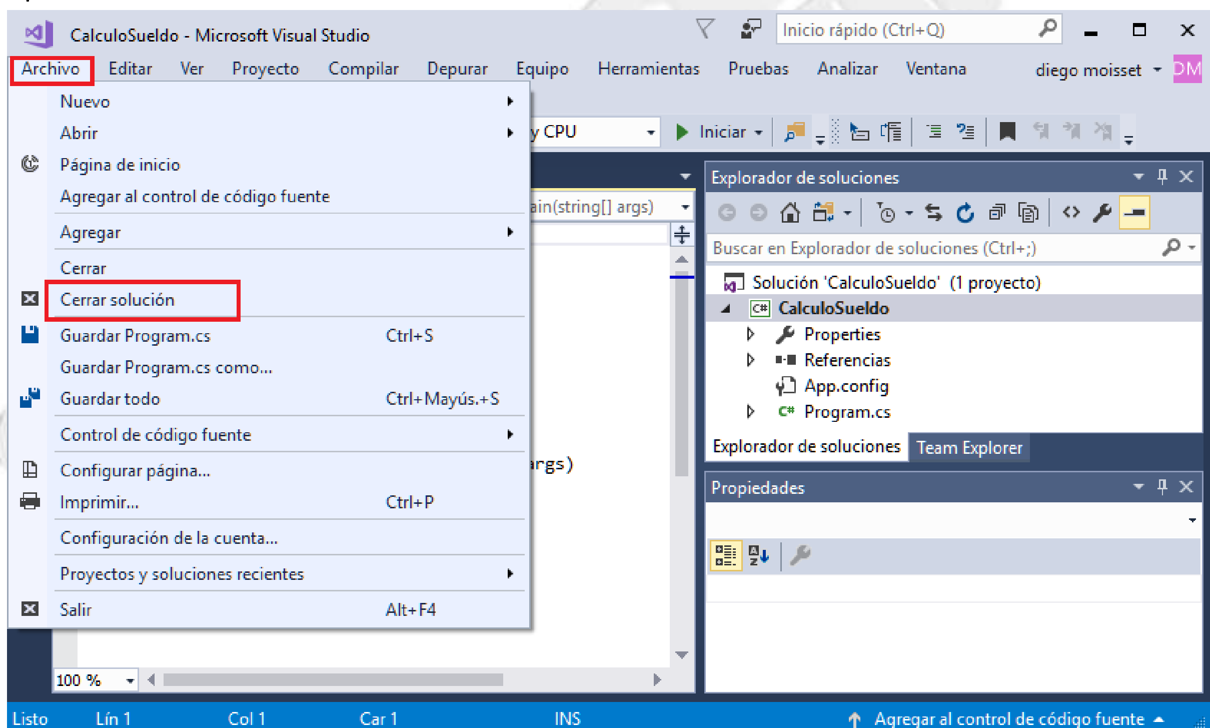


3 - Si vamos al Explorador de archivos de Windows podemos ver que tenemos creada la carpeta con nuestro programa:

Instituto Universitario de Tecnología para la Informática –IutepiInstituto Universitario de Tecnología para la Informática - Iutepi



4 - Una vez que finalizamos de trabajar con el proyecto lo cerramos desde el menú de opciones Archivo -> Cerrar solución:



1.3 Tipos de Errores

Sintácticos. Los errores de sintaxis, o sintácticos, ocurren cuando el programador escribe código que no va de acuerdo a las reglas de escritura del lenguaje de programación.

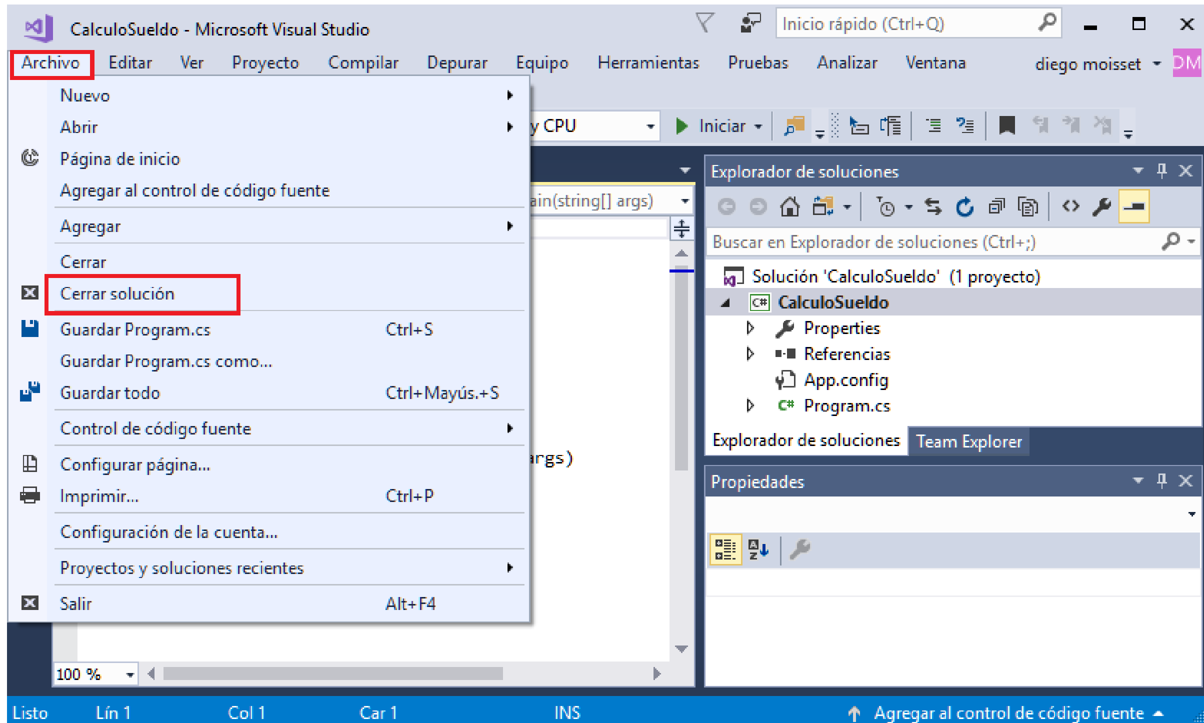
Instituto Universitario de Tecnología para la Informática –IutepiInstituto Universitario de Tecnología para la Informática - Iutepi

Lógicos. Los errores lógicos ocurren a causa de un mal diseño del programa. Puede ocurrir que una línea de código observe todas las reglas sintácticas del lenguaje, pero el código tenga una lógica equivocada.

Ejemplo Hallar la superficie de un cuadrado conociendo el valor de un lado.

Creemos un proyecto llamado SuperficieCuadrado.

Recordemos que si tenemos un proyecto abierto actualmente debemos cerrarlo desde Archivo -> Cerrar solución:



Codificamos el algoritmo en C# e introducimos dos errores sintáctico:

1 - Disponemos el nombre del objeto Console con minúsculas.

2 - Tratamos de imprimir el nombre de la variable superficie con el primer carácter en mayúsculas.

Instituto Universitario de Tecnología para la Informática –IutepiInstituto Universitario de Tecnología para la Informática - Iutepi

SuperficieCuadrado - Microsoft Visual Studio

Program.cs*

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace SuperficieCuadrado
{
    class Program
    {
        static void Main(string[] args)
        {
            int lado;
            int superficie;
            String linea;
            console.Write("Ingrese el valor del lado del cuadrado:");
            linea = Console.ReadLine();
            lado = int.Parse(linea);
            superficie = lado * lado;
            console.Write("La superficie del cuadrado es:");
            console.Write(Superficie);
            console.ReadKey();
        }
    }
}
```

Como podemos observar aparece subrayado la línea donde disponemos Console con minúsculas como en la línea que imprimimos la variable superficie con mayúsculas. Si modificamos y corregimos los errores sintácticos podremos ejecutar nuestro programa.

Programa correctamente codificado:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace SuperficieCuadrado
{
    class Program
    {
        static void Main(string[] args)
        {
            int lado;
            int superficie;
            String linea;
```

Instituto Universitario de Tecnología para la Informática –IutepiInstituto Universitario de Tecnología para la Informática - Iutepi

```
        Console.Write("Ingrese el valor del lado del cuadrado:");
        linea = Console.ReadLine();
        lado = int.Parse(linea);
        superficie = lado * lado;
        Console.Write("La superficie del cuadrado es:");
        Console.Write(superficie);
        Console.ReadKey();
      }
   }
}
```

Programa con un error lógico:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace SuperficieCuadrado
{
   class Program
   {
      static void Main(string[] args)
      {
         int lado;
         int superficie;
         String linea;
         Console.Write("Ingrese el valor del lado del cuadrado:");
         linea = Console.ReadLine();
         lado = int.Parse(linea);
         superficie = lado * lado * lado;
         Console.Write("La superficie del cuadrado es:");
         Console.Write(superficie);
         Console.ReadKey();
      }
   }
}
```

Como podemos observar si ejecutamos el programa no presenta ningún error de compilación. Pero luego de ingresar el valor del lado del cuadrado (por ejemplo el valor 10)

Instituto Universitario de Tecnología para la Informática –IutepiInstituto Universitario de Tecnología para la Informática - Iutepi

obtenemos como resultado un valor incorrecto (imprime el 1000), esto debido que definimos incorrectamente la fórmula para calcular la superficie del cuadrado:

superficie = lado * lado * lado;

Variables y Constantes

Ambas Representan un espacio de Memoria RAM que guarda un valor que servirá para algún proceso en particular, la Variable contiene un valor que puede ser modificado en cualquier momento, mientras que el valor de la constante es fijo y no cambia en la ejecución del programa.

1.4 Declaración de Variables

En C# existen 3 tipos de Variables:

Variables de instancia.

Se utilizan para definir los atributos de un objeto.

Variables de clase.

Son similares a las variables de instancia, con la excepción de que sus valores son los mismos para todas las instancias de la clase.

Variables locales.

Se declaran y se utilizan dentro de las definiciones de los métodos.

1.5Operadores

Operadores Numéricos.

Tipo de Operador	Operadores asociados
Cambio de signo	-,+
Aritméticos	+,-,*, /, %
Incremento y decremento	++, -

Instituto Universitario de Tecnología para la Informática –IutepiInstituto Universitario de Tecnología para la Informática - Iutepi

Operadores de Comparación

Operador	Significado
>	Mayor que
<	Menor que
==	Igual a
>=	Mayor o igual que
<=	Menor o igual que
!=	Distinto que

1.6 Tipos de datos

Nombre	Clase .NET	Tipo	Ancho	Intervalo (bits)
byte	Byte	Entero sin signo	8	0 a 255
sbyte	SByte	Entero con signo	8	-128 a 127
int	Int32	Entero con signo	32	-2.147.483.648 a 2.147.483.647
uint	UInt32	Entero sin signo	32	0 a 4294967295
short	Int16	Entero con signo	16	-32.768 a 32.767
ushort	UInt16	Entero sin signo	16	0 a 65535
long	Int64	Entero con signo	64	-922337203685477508 a 922337203685477507
ulong	UInt64	Entero sin signo	64	0 a 18446744073709551615
float	Single	Tipo de punto flotante de precisión simple	32	-3,402823e38 a 3,402823e38
Double	Double	Tipo de punto flotante de precisión doble	64	1,79769313486232e308 a 1,79769313486232e308

Instituto Universitario de Tecnología para la Informática –IutepiInstituto Universitario de Tecnología para la Informática - Iutepi

char	Char	Un carácter Unicode	16	Símbolos Unicode utilizados en el texto
bool	Boolean	Tipo Boolean lógico	8	True o false
object	Object	Tipo base de todos los otros tipos		
string	String	Una secuencia de caracteres		
decimal	Decimal	Tipo preciso fraccionario o integral, que puede representar números decimales con 29 dígitos significativos	128	±1.0 × 10e−28 a ±7.9 × 10e28

EVALUACION

1. Escribir un programa en el cual se ingresen cuatro números, calcular e informar la suma de los dos primeros y el producto del tercero y el cuarto.
2. Realizar un programa que lea cuatro valores numéricos e informar su suma y promedio.
3. Se debe desarrollar un programa que pida el ingreso del precio de un artículo y la cantidad que lleva el cliente. Mostrar lo que debe abonar el comprador.

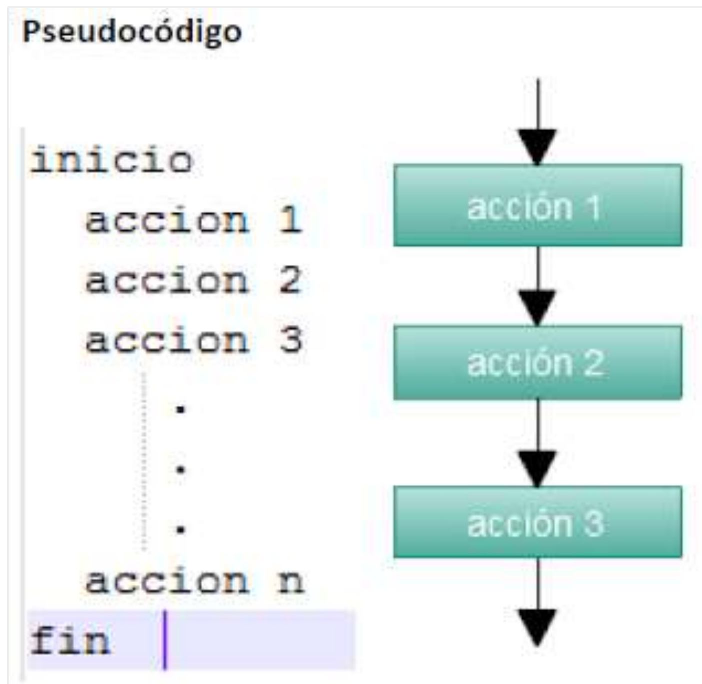
UNIDAD 2. ESTRUCTURAS DE PROGRAMACIÓN

2.1 Estructuras secuenciales

Cuando en un problema sólo participan operaciones, entradas y salidas se la denomina una estructura secuencial.
Los problemas diagramados y codificados previamente emplean solo estructuras secuenciales.

Instituto Universitario de Tecnología para la Informática –IutepiInstituto Universitario de Tecnología para la Informática - Iutepi

La programación requiere una práctica ininterrumpida de diagramación y codificación de problemas.



Ejercicios resueltos de Programación Secuencial en C#
Ejemplo 1

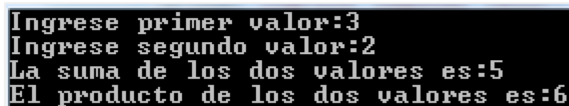
Realizar la carga de dos números enteros por teclado e imprimir su suma y su producto.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace SumaProductoNumeros
{
  class Program
       {
     static void Main(string[] args)
       {
        int num1, num2, suma, producto;
        string linea;
        Console.Write("Ingrese primer valor:");
         linea = Console.ReadLine();
         num1 = int.Parse(linea);
```

Instituto Universitario de Tecnología para la Informática –IutepiInstituto Universitario de Tecnología para la Informática - Iutepi

```
        Console.Write("Ingrese segundo valor:");
         linea = Console.ReadLine();
         num2 = int.Parse(linea);
         suma = num1 + num2;
         producto = num1 * num2;
        Console.Write("La suma de los dos valores es:");
        Console.WriteLine(suma);
        Console.Write("El producto de los dos valores es:");
        Console.WriteLine(producto);
        Console.ReadKey();
         }
         }
}
```

Al ejecutar el código muestra el siguiente resultado



Recordemos que tenemos que seguir todos los pasos vistos para la creación de un proyecto. Algunas cosas nuevas que podemos notar:

Podemos definir varias variables en la misma línea: int num1, num2, suma, producto;
Si llamamos a la función WriteLine en lugar de Write, la impresión siguiente se efectuará en la próxima línea: Console.WriteLine(suma);

2.2 Estructuras Condicionales

No todos los problemas pueden resolverse empleando estructuras secuenciales. Cuando hay que tomar una decisión aparecen las estructuras condicionales.
En nuestra vida diaria se nos presentan situaciones donde debemos decidir.
¿Elijo la carrera A o la carrera B?
¿Me pongo este pantalón?
Para ir al trabajo, ¿elijo el camino A o el camino B?
Al cursar una carrera, ¿elijo el turno mañana, tarde o noche?
Por supuesto que en un problema se combinan estructuras secuenciales y condicionales.
Ejemplo

```
using System;
using System.Collections.Generic;
```

Instituto Universitario de Tecnología para la Informática –IutepiInstituto Universitario de Tecnología para la Informática - Iutepi

```
using System.Linq;
using System.Text;

namespace EstructuraCondicionalSimple1
{
    class Program
    {
        static void Main(string[] args)
        {
                float sueldo;
            string linea;
                Console.Write("Ingrese el sueldo:");
            linea=Console.ReadLine();
                sueldo=float.Parse(linea);
                if (sueldo>3000)
            {
                    Console.Write("Esta persona debe abonar impuestos");
                }
            Console.ReadKey();
        }
    }
}
```

La palabra clave "if" indica que estamos en presencia de una estructura condicional; seguidamente disponemos la condición entre paréntesis. Por último encerrada entre llaves las instrucciones de la rama del verdadero.

Es necesario que las instrucciones a ejecutar en caso que la condición sea verdadera estén encerradas entre llaves { }, con ellas marcamos el comienzo y el fin del bloque del verdadero.

Ejecutando el programa e ingresamos un sueldo superior a 3000 pesos. Podemos observar como aparece en pantalla el mensaje "Esta persona debe abonar impuestos", ya que la condición del if es verdadera.

Volvamos a ejecutar el programa y carguemos un sueldo menor o igual a 3000 pesos. No debe aparecer mensaje en pantalla.

Estructura condicional switch

La estructura condicional switch remplaza en algunos casos un conjunto de if.

La estructura del switch:

```
switch(variable) {
    case valor1:
         Instrucciones
         break;
```

Instituto Universitario de Tecnología para la Informática –IutepiInstituto Universitario de Tecnología para la Informática - Iutepi

```
  case valor2:
      Instrucciones
      break;
  case valor3:
      Instrucciones
      break;
 .
 .
 .
 default:
       Instrucciones
       break;
}
```

Luego de la palabra clave switch entre paréntesis indicamos una variable, luego con una serie de case verificamos si dicha variable almacena un valor igual a [valor1, valor2, valor3 etc.] en el caso de ser igual se ejecutan las instrucciones contenidas en dicho case.
Si todos los case son falsos, luego se ejecutan las instrucciones contenidas después de la palabra default

Problema 1:

Ingresar un valor entero entre 1 y 5. Luego mostrar en castellano el valor ingresado. Si se ingresa un valor fuera de dicho rango mostrar un mensaje indicando tal situación

Programa:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Estructuraswitch1
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.Write("Ingrese un valor entre 1 y 5:");
            int valor = int.Parse(Console.ReadLine());
            switch (valor)
            {
                case 1: Console.Write("uno");
                        break;
                case 2: Console.Write("dos");
```

Instituto Universitario de Tecnología para la Informática –IutepiInstituto Universitario de Tecnología para la Informática - Iutepi

```
                break;
            case 3: Console.Write("tres");
                break;
            case 4: Console.Write("cuatro");
                break;
            case 5: Console.Write("cinco");
                break;
            default:
                Console.Write("Se ingreso un valor fuera de rango");
                break;
        }
        Console.ReadKey();
    }
  }
}
```

Es obligatorio que esté entre paréntesis la variable luego de la palabra clave switch.
Luego de cada case debemos indicar el valor con el que se comparará la variable (siempre debe ser un valor constante y no podemos disponer una variable luego de la palabra case.
Es necesario la palabra break luego de cada bloque de instrucciones por cada case.

2.3 Estructuras condicionales compuestas con operadores lógicos

Estos dos operadores se emplean fundamentalmente en las estructuras condicionales para agrupar varias condiciones simples.

Operador &&: Traducido se lo lee como “Y”. Si la Condición 1 es verdadera Y la condición 2 es verdadera luego ejecutar la rama del verdadero.

Cuando vinculamos dos o más condiciones con el operador “&&”, las dos condiciones deben ser verdaderas para que el resultado de la condición compuesta de Verdadero y continúe por la rama del verdadero de la estructura condicional.
La utilización de operadores lógicos permiten en muchos casos plantear algoritmos más cortos y comprensibles.

Problema:
Confeccionar un programa que lea por teclado tres números distintos y nos muestre el mayor.

La primera estructura condicional es una ESTRUCTURA CONDICIONAL COMPUESTA con una CONDICION COMPUESTA.

Instituto Universitario de Tecnología para la Informática –IutepiInstituto Universitario de Tecnología para la Informática - Iutepi

Podemos leerla de la siguiente forma:
Si el contenido de la variable num1 es mayor al contenido de la variable num2 Y si el contenido de la variable num1 es mayor al contenido de la variable num3 entonces la CONDICION COMPUESTA resulta Verdadera.
Si una de las condiciones simples da falso la CONDICION COMPUESTA da Falso y continua por la rama del falso.
Es decir que se mostrará el contenido de num1 si y sólo si num1>num2 y num1>num3. En caso de ser Falsa la condición, analizamos el contenido de num2 y num3 para ver cual tiene un valor mayor.
En esta segunda estructura condicional no se requieren operadores lógicos al haber una condición simple.

Programa:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace CondicionCompuesta1
{
    class Program
    {
        static void Main(string[] args)
        {
            int num1,num2,num3;
            string linea;
            Console.Write("Ingrese primer valor:");
            linea = Console.ReadLine();
            num1=int.Parse(linea);
            Console.Write("Ingrese segundo valor:");
            linea = Console.ReadLine();
            num2 = int.Parse(linea);
            Console.Write("Ingrese tercer valor:");
            linea = Console.ReadLine();
            num3 = int.Parse(linea);
            if (num1>num2 && num1>num3)
            {
                Console.Write(num1);
            }
            else
            {
```

Instituto Universitario de Tecnología para la Informática –IutepiInstituto Universitario de Tecnología para la Informática - Iutepi

```
            if (num2>num3)
            {
             Console.Write(num2);
            }
            else
            {
             Console.Write(num3);
            }
        }
        Console.ReadKey();
    }
  }
}
```

Operador ||

Operador or

Traducido se lo lee como “O”. Si la condición 1 es Verdadera O la condición 2 es Verdadera, luego ejecutar la rama del Verdadero.

Cuando vinculamos dos o más condiciones con el operador “Or", con que una de las dos condiciones sea Verdadera alcanza para que el resultado de la condición compuesta sea Verdadero.

Problema:

Se carga una fecha (día, mes y año) por teclado. Mostrar un mensaje si corresponde al primer trimestre del año (enero, febrero o marzo) Cargar por teclado el valor numérico del día, mes y año.

Ejemplo: dia:10 mes:1 año:2010.

La carga de una fecha se hace por partes, ingresamos las variables dia, mes y año.

Mostramos el mensaje "Corresponde al primer trimestre" en caso que el mes ingresado por teclado sea igual a 1, 2 ó 3.

En la condición no participan las variables dia y año.

Programa:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace CondicionCompuesta2
{
  class Program
```


Instituto Universitario de Tecnología para la Informática –IutepiInstituto Universitario de Tecnología para la Informática - Iutepi

```
{
    static void Main(string[] args)
    {
        int dia,mes,año;
        string linea;
        Console.Write("Ingrese nro de día:");
        linea = Console.ReadLine();
        dia = int.Parse(linea); ;
        Console.Write("Ingrese nro de mes:");
        linea = Console.ReadLine();
        mes=int.Parse(linea);
        Console.Write("Ingrese nro de año:");
        linea = Console.ReadLine();
        año=int.Parse(linea);
        if (mes==1 || mes==2 || mes==3)
        {
            Console.Write("Corresponde al primer trimestre");
        }
        Console.ReadLine();
    }
}
}
```

EVALUACION

Realice los siguientes ejercicios

1) Se ingresan tres valores por teclado, si todos son iguales se imprime la suma del primero con el segundo y a este resultado se lo multiplica por el tercero.
2) Se ingresan por teclado tres números, si todos los valores ingresados son menores a 10, imprimir en pantalla la leyenda "Todos los números son menores a diez".
3) Se ingresan por teclado tres números, si al menos uno de los valores ingresados es menor a 10, imprimir en pantalla la leyenda "Alguno de los números es menor a diez".
4) Escribir un programa que pida ingresar la coordenada de un punto en el plano, es decir dos valores enteros x e y (distintos a cero). Posteriormente imprimir en pantalla en que cuadrante se ubica dicho punto. (1º Cuadrante si x > 0 Y y > 0 , 2º Cuadrante: x < 0 Y y > 0, etc.)

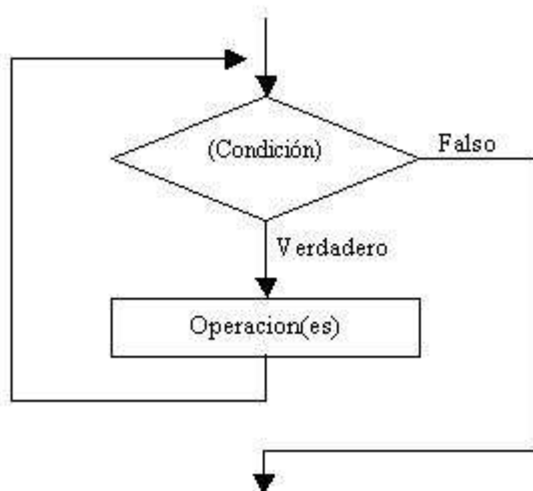
Instituto Universitario de Tecnología para la Informática –IutepiInstituto Universitario de Tecnología para la Informática - Iutepi

UNIDAD 3. ESTRUCTURAS REPETITIVAS, CICLOS O BUCLES

Una estructura repetitiva permite ejecutar una instrucción o un conjunto de instrucciones varias veces.
Una ejecución repetitiva de sentencias se caracteriza por:
- La o las sentencias que se repiten.
- El test o prueba de condición antes de cada repetición, que motivará que se repitan o no las sentencias.

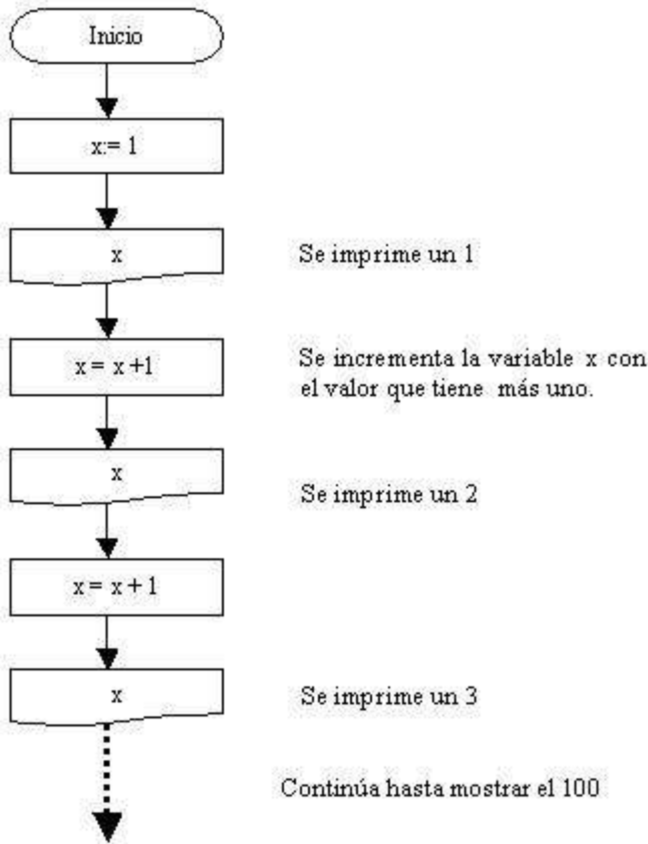
3.1 Estructura repetitiva while.

Representación gráfica de la estructura while:



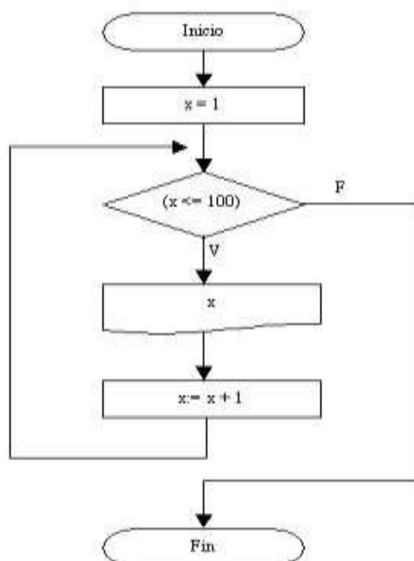
No debemos confundir la representación gráfica de la estructura repetitiva while (Mientras) con la estructura condicional if (Si)
Funcionamiento: En primer lugar se verifica la condición, si la misma resulta verdadera se ejecutan las operaciones que indicamos por la rama del Verdadero.
A la rama del verdadero la graficamos en la parte inferior de la condición. Una línea al final del bloque de repetición la conecta con la parte superior de la estructura repetitiva.
En caso que la condición sea Falsa continúa por la rama del Falso y sale de la estructura repetitiva para continuar con la ejecución del algoritmo.
El bloque se repite MIENTRAS la condición sea Verdadera.
Importante: Si la condición siempre retorna verdadero estamos en presencia de un ciclo repetitivo infinito. Dicha situación es un error de programación, nunca finalizará el programa.
Problema 1:
Realizar un programa que imprima en pantalla los números del 1 al 100.
Sin conocer las estructuras repetitivas podemos resolver el problema empleando una estructura secuencial. Inicializamos una variable con el valor 1, luego imprimimos la variable, incrementamos nuevamente la variable y así sucesivamente.
Diagrama de flujo:

Instituto Universitario de Tecnología para la Informática –IutepiInstituto Universitario de Tecnología para la Informática - Iutepi



Si continuamos con el diagrama no nos alcanzarían las próximas 5 páginas para finalizarlo. Emplear una estructura secuencial para resolver este problema produce un diagrama de flujo y un programa en C# muy largo.

Ahora veamos la solución empleando una estructura repetitiva while:



Es muy importante analizar este diagrama:

Instituto Universitario de Tecnología para la Informática –IutepiInstituto Universitario de Tecnología para la Informática - Iutepi

La primera operación inicializa la variable x en 1, seguidamente comienza la estructura repetitiva while y disponemos la siguiente condición (x <= 100), se lee MIENTRAS la variable x sea menor o igual a 100.
Al ejecutarse la condición retorna VERDADERO porque el contenido de x (1) es menor o igual a 100. Al ser la condición verdadera se ejecuta el bloque de instrucciones que contiene la estructura while. El bloque de instrucciones contiene una salida y una operación.
Se imprime el contenido de x, y seguidamente se incrementa la variable x en uno.
La operación x=x + 1 se lee como "en la variable x se guarda el contenido de x más 1". Es decir, si x contiene 1 luego de ejecutarse esta operación se almacenará en x un 2.
Al finalizar el bloque de instrucciones que contiene la estructura repetitiva se verifica nuevamente la condición de la estructura repetitiva y se repite el proceso explicado anteriormente.
Mientras la condición retorne verdadero se ejecuta el bloque de instrucciones; al retornar falso la verificación de la condición se sale de la estructura repetitiva y continua el algoritmo, en este caso finaliza el programa.
Lo más difícil es la definición de la condición de la estructura while y qué bloque de instrucciones se van a repetir. Observar que si, por ejemplo, disponemos la condición x >=100 (si x es mayor o igual a 100) no provoca ningún error sintáctico pero estamos en presencia de un error lógico porque al evaluarse por primera vez la condición retorna falso y no se ejecuta el bloque de instrucciones que queríamos repetir 100 veces.
No existe una RECETA para definir una condición de una estructura repetitiva, sino que se logra con una práctica continua solucionando problemas.
Una vez planteado el diagrama debemos verificar si el mismo es una solución válida al problema (en este caso se debe imprimir los números del 1 al 100 en pantalla), para ello podemos hacer un seguimiento del flujo del diagrama y los valores que toman las variables a lo largo de la ejecución:

x
1
2
3
4
.
.
100
101 Cuando x vale 101 la condición de la estructura repetitiva retorna falso, en este caso finaliza el diagrama.

Importante: Podemos observar que el bloque repetitivo puede no ejecutarse ninguna vez si la condición retorna falso la primera vez.
La variable x debe estar inicializada con algún valor antes que se ejecute la operación x=x + 1 en caso de no estar inicializada aparece un error de compilación.

Instituto Universitario de Tecnología para la Informática –IutepiInstituto Universitario de Tecnología para la Informática - Iutepi

Programa:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace EstructuraRepetitivaWhile1
{
    class Program
    {
        static void Main(string[] args)
        {
            int x;
            x = 1;
            while (x <= 100)
            {
                Console.Write(x);
                Console.Write(" - ");
                x = x + 1;
            }
            Console.ReadKey();
        }
    }
}
```

Problema 2:

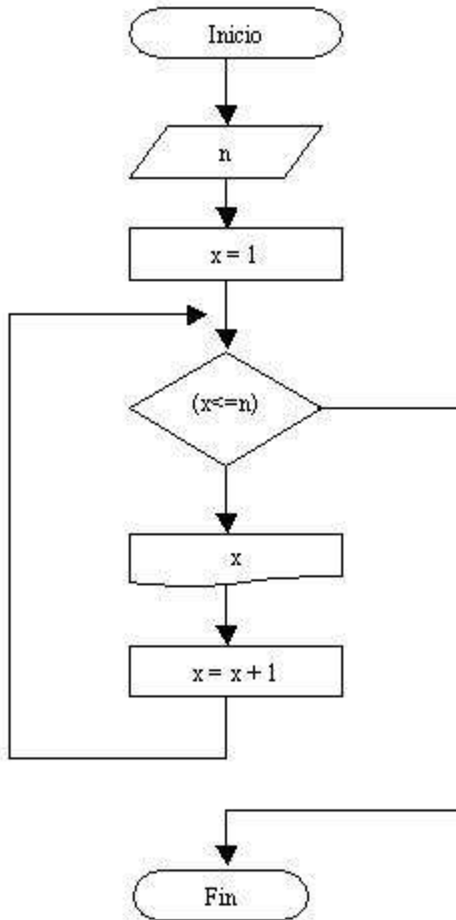
Escribir un programa que solicite la carga de un valor positivo y nos muestre desde 1 hasta el valor ingresado de uno en uno.

Ejemplo: Si ingresamos 30 se debe mostrar en pantalla los números del 1 al 30.

Es de FUNDAMENTAL importancia analizar los diagramas de flujo y la posterior codificación en C# de los siguientes problemas, en varios problemas se presentan otras situaciones no vistas en el ejercicio anterior.

Diagrama de flujo:

Instituto Universitario de Tecnología para la Informática –IutepiInstituto Universitario de Tecnología para la Informática - Iutepi



Podemos observar que se ingresa por teclado la variable n. El operador puede cargar cualquier valor.

Si el operador carga 10 el bloque repetitivo se ejecutará 10 veces, ya que la condición es "Mientras x<=n ", es decir "mientras x sea menor o igual a 10"; pues x comienza en uno y se incrementa en uno cada vez que se ejecuta el bloque repetitivo.

A la prueba del diagrama la podemos realizar dándole valores a las variables; por ejemplo, si ingresamos 5 el seguimiento es el siguiente:

n x
5 1 (Se imprime el contenido de x)
2 " "
3 " "
4 " "
5 " "
6 (Sale del while porque 6 no es menor o igual a 5)

Programa:

```
using System;
using System.Collections.Generic;
using System.Linq;
```


Instituto Universitario de Tecnología para la Informática –IutepiInstituto Universitario de Tecnología para la Informática - Iutepi

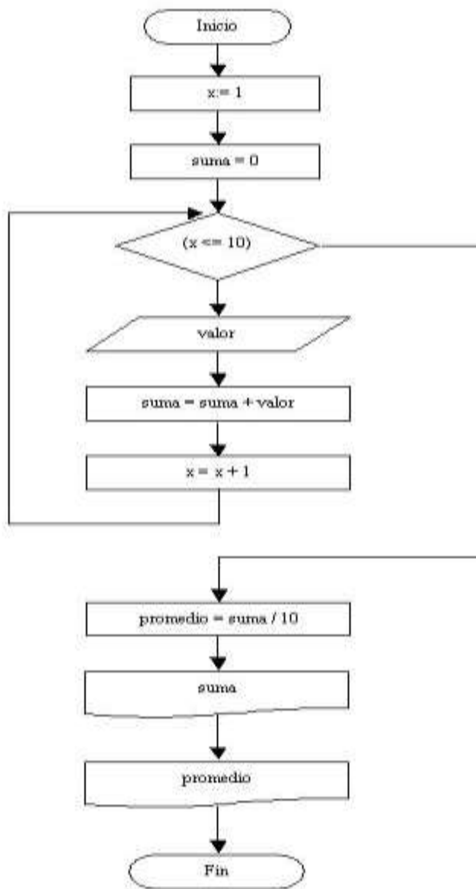
```
using System.Text;

namespace EstructuraRepetitivaWhile2
{
    class Program
    {
        static void Main(string[] args)
        {
            int n,x;
            string linea;
            Console.Write("Ingrese el valor final:");
            linea=Console.ReadLine();
            n=int.Parse(linea);
            x=1;
            while (x<=n)
            {
                Console.Write(x);
                Console.Write(" - ");
                x = x + 1;
            }
            Console.ReadKey();
        }
    }
}
```

Los nombres de las variables n y x pueden ser palabras o letras (como en este caso)
La variable x recibe el nombre de CONTADOR. Un contador es un tipo especial de variable que se incrementa o decrementa con valores constantes durante la ejecución del programa.
El contador x nos indica en cada momento la cantidad de valores impresos en pantalla.

Problema 3:

Desarrollar un programa que permita la carga de 10 valores por teclado y nos muestre posteriormente la suma de los valores ingresados y su promedio.
Diagrama de flujo:

Instituto Universitario de Tecnología para la Informática –IutepiInstituto Universitario de Tecnología para la Informática - Iutepi



En este problema, a semejanza de los anteriores, llevamos un CONTADOR llamado x que nos sirve para contar las vueltas que debe repetir el while.

También aparece el concepto de ACUMULADOR (un acumulador es un tipo especial de variable que se incrementa o decrementa con valores variables durante la ejecución del programa)

Hemos dado el nombre de suma a nuestro acumulador. Cada ciclo que se repita la estructura repetitiva, la variable suma se incrementa con el contenido ingresado en la variable valor.

Programa:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace EstructuraRepetitivaWhile3
{
    class Program
    {
        static void Main(string[] args)
        {
```

Instituto Universitario de Tecnología para la Informática –IutepiInstituto Universitario de Tecnología para la Informática - Iutepi

```
            int x,suma,valor,promedio;
            string linea;
            x=1;
            suma=0;
            while (x<=10)
            {
                Console.Write("Ingrese un valor:");
                linea = Console.ReadLine();
                valor=int.Parse(linea);
                suma=suma+valor;
                x=x+1;
            }
            promedio=suma/10;
            Console.Write("La suma de los 10 valores es:");
            Console.WriteLine(suma);
            Console.Write("El promedio es:");
            Console.Write(promedio);
            Console.ReadKey();
        }
    }
}
```

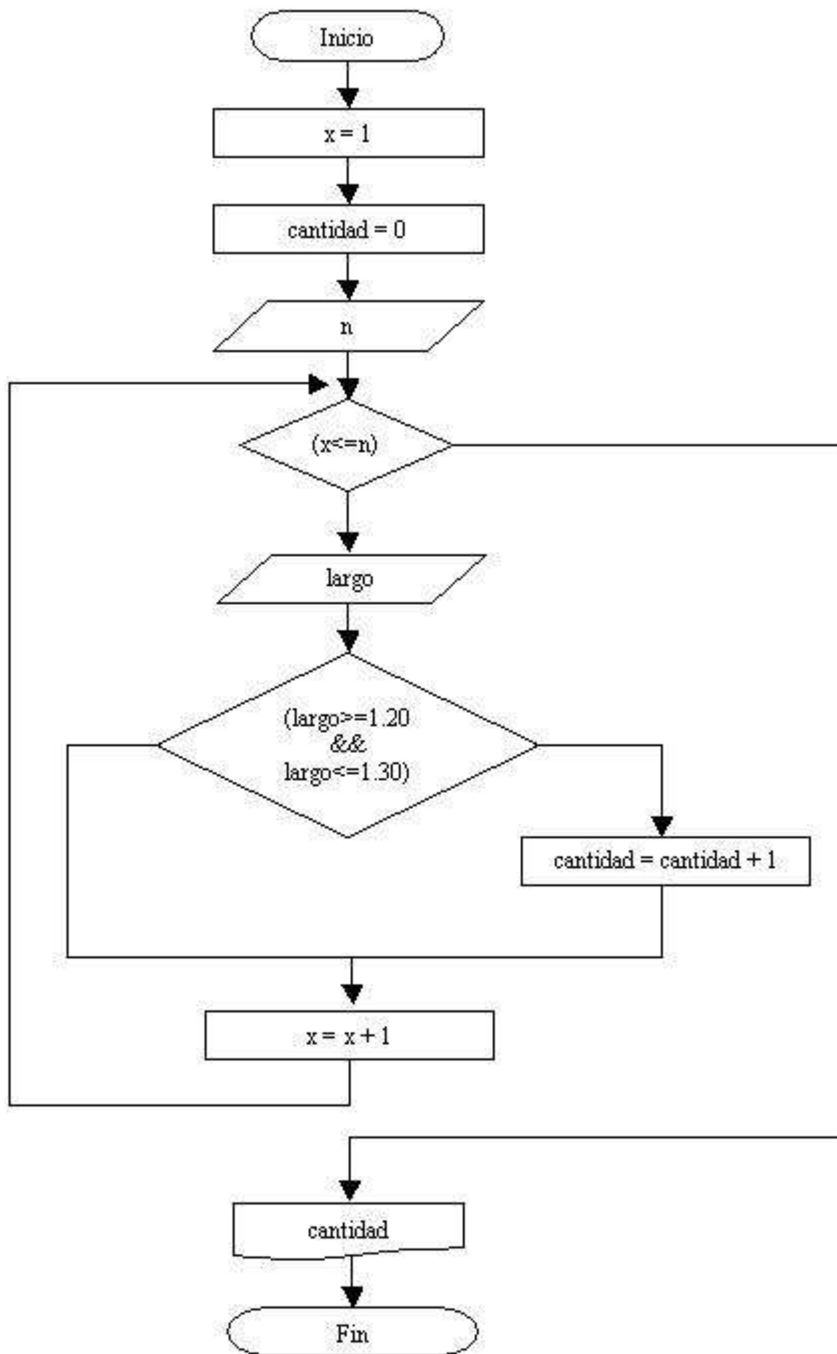
Problema 4:

Una planta que fabrica perfiles de hierro posee un lote de n piezas.

Confeccionar un programa que pida ingresar por teclado la cantidad de piezas a procesar y luego ingrese la longitud de cada perfil; sabiendo que la pieza cuya longitud esté comprendida en el rango de 1,20 y 1,30 son aptas. Imprimir por pantalla la cantidad de piezas aptas que hay en el lote.

Diagrama de flujo:

Instituto Universitario de Tecnología para la Informática –IutepiInstituto Universitario de Tecnología para la Informática - Iutepi



Podemos observar que dentro de una estructura repetitiva puede haber estructuras condicionales (inclusive puede haber otras estructuras repetitivas que veremos más adelante)

En este problema hay que cargar inicialmente la cantidad de piezas a ingresar (n), seguidamente se cargan n valores de largos de piezas.

Cada vez que ingresamos un largo de pieza (largo) verificamos si es una medida correcta (debe estar entre 1.20 y 1.30 el largo para que sea correcta), en caso de ser correcta la CONTAMOS (incrementamos la variable cantidad en 1)

Instituto Universitario de Tecnología para la Informática –IutepiInstituto Universitario de Tecnología para la Informática - Iutepi

Al contador cantidad lo inicializamos en cero porque inicialmente no se ha cargado ningún largo de medida.
Cuando salimos de la estructura repetitiva porque se han cargado n largos de piezas mostramos por pantalla el contador cantidad (que representa la cantidad de piezas aptas)
En este problema tenemos dos CONTADORES:
x (Cuenta la cantidad de piezas cargadas hasta el momento)
cantidad (Cuenta los perfiles de hierro aptos)

Programa:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace EstructuraRepetitivaWhile4
{
    class Program
    {
        static void Main(string[] args)
        {
            int x,cantidad,n;
            float largo;
            string linea;
            x=1;
            cantidad=0;
            Console.Write("Cuantas piezar procesará:");
            linea = Console.ReadLine();
            n=int.Parse(linea);
            while (x<=n)
            {
                Console.Write("Ingrese la medida de la pieza:");
                linea = Console.ReadLine();
                largo=float.Parse(linea);
                if (largo>=1.20 && largo<=1.30)
                {
                    cantidad = cantidad +1;
                }
                x=x + 1;
            }
            Console.Write("La cantidad de piezas aptas son:");
            Console.Write(cantidad);
```


Instituto Universitario de Tecnología para la Informática –IutepiInstituto Universitario de Tecnología para la Informática - Iutepi

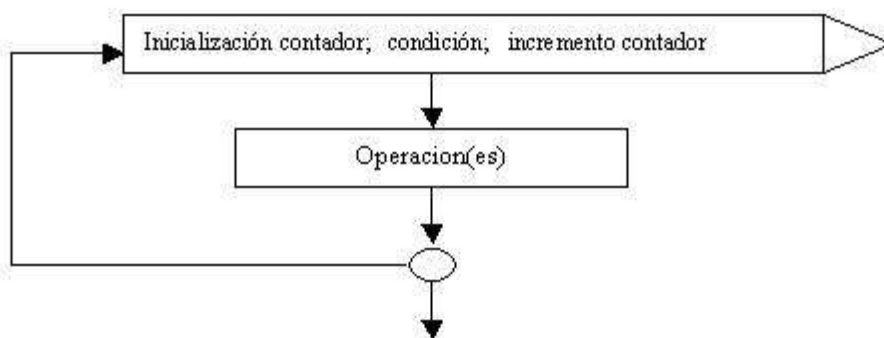
```
        Console.ReadKey();
      }
    }
}
```

3.2 Estructura repetitiva for

Cualquier problema que requiera una estructura repetitiva se puede resolver empleando la estructura while. Pero hay otra estructura repetitiva cuyo planteo es más sencillo en ciertas situaciones.

En general, la estructura for se usa en aquellas situaciones en las cuales CONOCEMOS la cantidad de veces que queremos que se ejecute el bloque de instrucciones. Ejemplo: cargar 10 números, ingresar 5 notas de alumnos, etc. Conocemos de antemano la cantidad de veces que queremos que el bloque se repita. Veremos, sin embargo, que en el lenguaje C# la estructura for puede usarse en cualquier situación repetitiva, porque en última instancia no es otra cosa que una estructura while generalizada.

Representación gráfica:

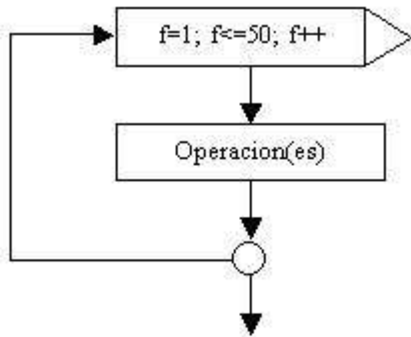


En su forma más típica y básica, esta estructura requiere una variable entera que cumple la función de un CONTADOR de vueltas. En la sección indicada como "inicialización contador", se suele colocar el nombre de la variable que hará de contador, asignándole a dicha variable un valor inicial. En la sección de "condición" se coloca la condición que deberá ser verdadera para que el ciclo continúe (en caso de un falso, el ciclo se detendrá). Y finalmente, en la sección de "incremento contador" se coloca una instrucción que permite modificar el valor de la variable que hace de contador (para permitir que alguna vez la condición sea falsa)

Cuando el ciclo comienza, antes de dar la primera vuelta, la variable del for toma el valor indicado en la sección de "inicialización contador". Inmediatamente se verifica, en forma automática, si la condición es verdadera. Seguidamente, se vuelve a controlar el valor de la condición, y así prosigue hasta que dicha condición entregue un falso.

Si conocemos la cantidad de veces que se repite el bloque es muy sencillo emplear un for, por ejemplo si queremos que se repita 50 veces el bloque de instrucciones puede hacerse así:

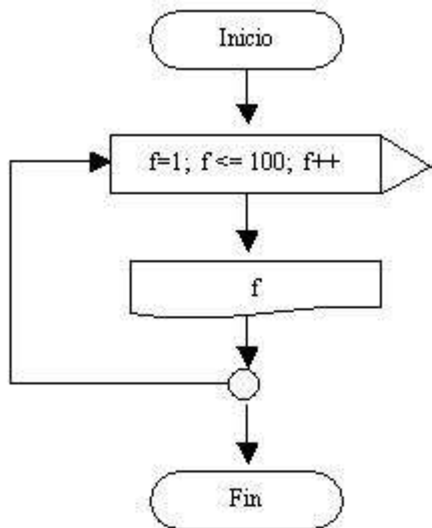
Instituto Universitario de Tecnología para la Informática –IutepiInstituto Universitario de Tecnología para la Informática - Iutepi



La variable del for puede tener cualquier nombre. En este ejemplo se la ha definido con el nombre f.
Analicemos el ejemplo:
- La variable f toma inicialmente el valor 1.
- Se controla automáticamente el valor de la condición: como f vale 1 y esto es menor que 50, la condición da verdadero.
- Como la condición fue verdadera, se ejecutan la/s operación/es.
- Al finalizar de ejecutarlas, se retorna a la instrucción f++, por lo que la variable f se incrementa en uno.
- Se vuelve a controlar (automáticamente) si f es menor o igual a 50.
Como ahora su valor es 2, se ejecuta nuevamente el bloque de instrucciones e incrementa nuevamente la variable del for al terminar el mismo.
- El proceso se repetirá hasta que la variable f sea incrementada al valor 51.
En este momento la condición será falsa, y el ciclo se detendrá.

La variable f PUEDE ser modificada dentro del bloque de operaciones del for, aunque esto podría causar problemas de lógica si el programador es inexperto.
La variable f puede ser inicializada en cualquier valor y finalizar en cualquier valor. Además, no es obligatorio que la instrucción de modificación sea un incremento del tipo contador (f++).
Cualquier instrucción que modifique el valor de la variable es válida. Si por ejemplo se escribe f=f+2 en lugar de f++, el valor de f será incrementado de a 2 en cada vuelta, y no de a 1. En este caso, esto significará que el ciclo no efectuará las 50 vueltas sino sólo 25.
Problema 1:
Realizar un programa que imprima en pantalla los números del 1 al 100.
Diagrama de flujo:

Instituto Universitario de Tecnología para la Informática –IutepiInstituto Universitario de Tecnología para la Informática - Iutepi



Podemos observar y comparar con el problema realizado con el while. Con la estructura while el CONTADOR x sirve para contar las vueltas. Con el for el CONTADOR f cumple dicha función.

Inicialmente f vale 1 y como no es superior a 100 se ejecuta el bloque, imprimimos el contenido de f, al finalizar el bloque repetitivo se incrementa la variable f en 1, como 2 no es superior a 100 se repite el bloque de instrucciones.

Cuando la variable del for llega a 101 sale de la estructura repetitiva y continúa la ejecución del algoritmo que se indica después del círculo.

La variable f (o como sea que se decida llamarla) debe estar definida como una variable más.

Programa:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace EstructuraRepetitivaFor1
{
    class Program
    {
        static void Main(string[] args)
        {
            int f;
            for(f=1;f<=100;f++)
            {
                Console.Write(f);
                Console.Write("-");
            }
            Console.ReadKey();
        }
```

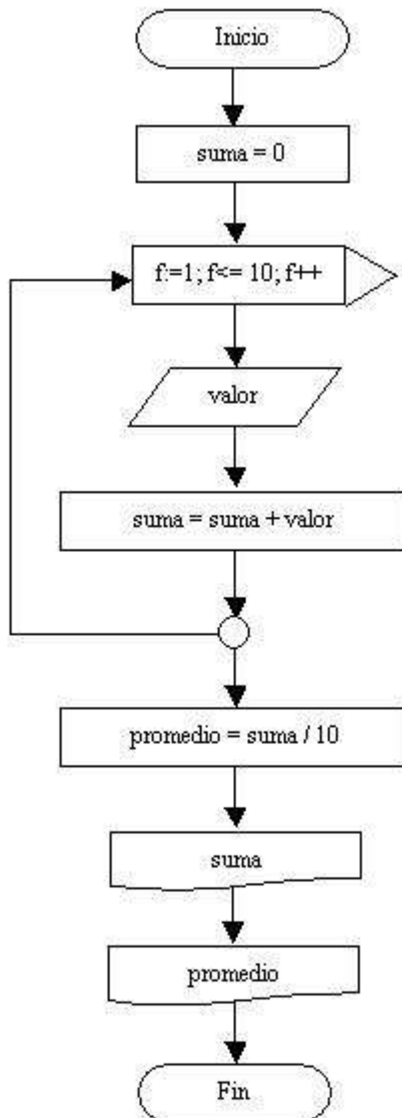
Instituto Universitario de Tecnología para la Informática –IutepiInstituto Universitario de Tecnología para la Informática - Iutepi

```
    }
}
```

Problema 2:

: Desarrollar un programa que permita la carga de 10 valores por teclado y nos muestre posteriormente la suma de los valores ingresados y su promedio. Este problema ya lo desarrollamos, lo resolveremos empleando la estructura for.

Diagrama de flujo:



En este caso, a la variable del for (f) sólo se la requiere para que se repita el bloque de instrucciones 10 veces.

Programa:

```
using System;
using System.Collections.Generic;
using System.Linq;
```


Instituto Universitario de Tecnología para la Informática –IutepiInstituto Universitario de Tecnología para la Informática - Iutepi

```
using System.Text;

namespace EstructuraRepetitivaFor2
{
    class Program
    {
        static void Main(string[] args)
        {
            int suma,f,valor,promedio;
            string linea;
            suma=0;
            for(f=1;f<=10;f++)
            {
                Console.Write("Ingrese valor:");
                linea=Console.ReadLine();
                valor=int.Parse(linea);
                suma=suma+valor;
            }
            Console.Write("La suma es:");
            Console.WriteLine(suma);
            promedio=suma/10;
            Console.Write("El promedio es:");
            Console.Write(promedio);
            Console.ReadKey();
        }
    }
}
```

El problema requiere que se carguen 10 valores y se sumen los mismos.
Tener en cuenta encerrar entre llaves bloque de instrucciones a repetir dentro del for.
El promedio se calcula fuera del for luego de haber cargado los 10 valores.

3.3 Estructura repetitiva DO WHILE

La estructura do while es otra estructura repetitiva, la cual ejecuta al menos una vez su bloque repetitivo, a diferencia del while o del for que podían no ejecutar el bloque. Esta estructura repetitiva se utiliza cuando conocemos de antemano que por lo menos una vez se ejecutará el bloque repetitivo.
La condición de la estructura está abajo del bloque a repetir, a diferencia del while o del for que está en la parte superior.

Instituto Universitario de Tecnología para la Informática –IutepiInstituto Universitario de Tecnología para la Informática - Iutepi

Problema 1:

Escribir un programa que solicite la carga de un número entre 0 y 999, y nos muestre un mensaje de cuántos dígitos tiene el mismo. Finalizar el programa cuando se cargue el valor 0.

Programa:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace EstructuraRepetitivaDoWhile1
{
    class Program
    {
        static void Main(string[] args)
        {
            int valor;
            string linea;
            do {
                Console.Write("Ingrese un valor entre 0 y 999 (0 finaliza):");
                linea = Console.ReadLine();
                valor=int.Parse(linea);
                if (valor>=100)
                {
                    Console.WriteLine("Tiene 3 dígitos.");
                }
                else
                {
                    if (valor>=10)
                    {
                        Console.WriteLine("Tiene 2 dígitos.");
                    }
                    else
                    {
                        Console.WriteLine("Tiene 1 dígito.");
                    }
                }
            } while (valor!=0);
```


Instituto Universitario de Tecnología para la Informática –IutepiInstituto Universitario de Tecnología para la Informática - Iutepi

```
      }
   }
}
```

Problema 2:

Escribir un programa que solicite la carga de números por teclado, obtener su promedio. Finalizar la carga de valores cuando se cargue el valor 0.

Cuando la finalización depende de algún valor ingresado por el operador conviene el empleo de la estructura do while, por lo menos se cargará un valor (en el caso más extremo se carga 0, que indica la finalización de la carga de valores)

Definimos un contador cant que cuenta la cantidad de valores ingresados por el operador (no lo incrementa si ingresamos 0)
El valor 0 no es parte de la serie de valores que se deben sumar.
Definimos el acumulador suma que almacena todos los valores ingresados por teclado.
La estructura repetitiva do while se repite hasta que ingresamos el valor 0. Con dicho valor la condición del ciclo retorna falso y continúa con el flujo del diagrama.
Disponemos por último una estructura condicional para el caso que el operador cargue únicamente un 0 y por lo tanto no podemos calcular el promedio ya que no existe la división por 0.
En caso que el contador cant tenga un valor distinto a 0 el promedio se obtiene dividiendo el acumulador suma por el contador cant que tiene la cantidad de valores ingresados antes de introducir el 0.

Programa:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace EstructuraRepetitivaDoWhile2
{
   class Program
   {
      static void Main(string[] args)
      {
         int suma,cant,valor,promedio;
         string linea;
         suma=0;
         cant=0;
```


Instituto Universitario de Tecnología para la Informática –IutepiInstituto Universitario de Tecnología para la Informática - Iutepi

```
        do {
            Console.Write("Ingrese un valor (0 para finalizar):");
            linea = Console.ReadLine();
            valor=int.Parse(linea);
            if (valor!=0) {
                suma=suma+valor;
                cant++;
            }
        } while (valor!=0);
        if (cant!=0) {
            promedio=suma/cant;
            Console.Write("El promedio de los valores ingresados es:");
            Console.Write(promedio);
        } else {
            Console.Write("No se ingresaron valores.");
        }
        Console.ReadLine();
    }
  }
}
```

El contador cant DEBE inicializarse antes del ciclo, lo mismo que el acumulador suma. El promedio se calcula siempre y cuando el contador cant sea distinto a 0.

3.4 Estructura Repetitiva Foreach

La estructura repetitiva foreach es utilizada para recorrer colecciones de datos (por ejemplo vectores), si bien podemos utilizar de forma tradicional las otras estructuras repetitivas el empleo del foreach hacen más natural el acceso a los elementos.

Problema 1:

Almacenar los sueldos de 5 operarios en un vector, imprimir los elementos recorriendo el vector con la estructura repetitiva foreach.

Programa:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Estructuraforeach1
{
  class SueldoEmpleados
```

Instituto Universitario de Tecnología para la Informática –IutepiInstituto Universitario de Tecnología para la Informática - Iutepi

```
{
    private int[] sueldos;

    public void Cargar()
    {
        sueldos = new int[5];
        for (int f = 0; f < 5; f++)
        {
            Console.Write("Ingrese valor de la componente:");
            String linea;
            linea = Console.ReadLine();
            sueldos[f] = int.Parse(linea);
        }
    }

    public void Imprimir()
    {
        foreach (int s in sueldos)
        {
            Console.WriteLine(s);
        }
        Console.ReadKey();
    }

    static void Main(string[] args)
    {
        SueldoEmpleados pv = new SueldoEmpleados();
        pv.Cargar();
        pv.Imprimir();
    }
  }
}
```

El funcionamiento del foreach:

```
foreach (int s in sueldos)
{
    Console.WriteLine(s);
}
```

La variable s almacena la primera vez el primer elemento del vector sueldos, seguidamente se ejecuta el bloque del foreach (en este caso imprimimos el contenido de la variable s)

Instituto Universitario de Tecnología para la Informática –IutepiInstituto Universitario de Tecnología para la Informática - Iutepi

Es decir que s almacena en cada vuelta del foreach un elemento del vector.
Con la estructura foreach recorremos en forma completa el vector y en cada iteración tenemos acceso a un elemento del vector que se copia en una variable auxiliar.
Podemos utilizar la palabra clave var para definir en forma implícita la variable que almacena sucesivamente los elementos del vector:

```
public void Imprimir()
{
   foreach (var s in sueldos)
   {
      Console.WriteLine(s);
   }
   Console.ReadKey();
}
```

EVALUACIÓN

Ejercicio 1: Escribir un programa que lea 10 notas de alumnos y nos informe cuántos tienen notas mayores o iguales a 7 y cuántos menores.
Para resolver este problema se requieren tres contadores:
aprobados (Cuenta la cantidad de alumnos aprobados)
reprobados (Cuenta la cantidad de reprobados)
f (es el contador del for)
Ejercicio 2
En un banco se procesan datos de las cuentas corrientes de sus clientes. De cada cuenta corriente se conoce: número de cuenta y saldo actual. El ingreso de datos debe finalizar al ingresar un valor negativo en el número de cuenta.
Se pide confeccionar un programa que lea los datos de las cuentas corrientes e informe:
a) De cada cuenta: número de cuenta y estado de la cuenta según su saldo, sabiendo que:

Estado de la cuenta 'Acreedor' si el saldo es >0.
'Deudor' si el saldo es <0.
'Nulo' si el saldo es =0.

b) La suma total de los saldos acreedores.

UNIDAD 4. CLASES Y OBJETOS

4.1 Clases

Las clases son los tipos más fundamentales de C#. Una clase es una estructura de datos que combina estados (campos) y acciones (métodos y otros miembros de función) en una sola

Instituto Universitario de Tecnología para la Informática –IutepiInstituto Universitario de Tecnología para la Informática - Iutepi

unidad. Una clase proporciona una definición para instancias creadas dinámicamente de la clase, también conocidas como objetos. Las clases admiten herencia y polimorfismo, mecanismos por los que las clases derivadas pueden extender y especializar clases base.
Las clases nuevas se crean mediante declaraciones de clase. Una declaración de clase se inicia con un encabezado que especifica los atributos y modificadores de la clase, el nombre de la clase, la clase base (si se indica) y las interfaces implementadas por la clase. Al encabezado le sigue el cuerpo de la clase, que consta de una lista de declaraciones de miembros escritas entre los delimitadores { y }.
La siguiente es una declaración de una clase simple denominada Point:

```
public class Point
{
   public int x, y;
   public Point(int x, int y)
   {
      this.x = x;
      this.y = y;
   }
}
```

Las instancias de clases se crean mediante el operador new, que asigna memoria para una nueva instancia, invoca un constructor para inicializar la instancia y devuelve una referencia a la instancia.Las instrucciones siguientes crean dos objetos Point y almacenan las referencias en esos objetos en dos variables:

```
Point p1 = new Point(0, 0);
Point p2 = new Point(10, 20);
```

Clase Base

Una declaración de clase puede especificar una clase base colocando después del nombre de clase y los parámetros de tipo dos puntos seguidos del nombre de la clase base. Omitir una especificación de la clase base es igual que derivarla del tipo object. En el ejemplo siguiente, la clase base de Point3D es Point y la clase base de Point es object:

```
public class Point
{
   public int x, y;
   public Point(int x, int y)
   {
      this.x = x;
      this.y = y;
   }
}
public class Point3D: Point
{
```

Instituto Universitario de Tecnología para la Informática –IutepiInstituto Universitario de Tecnología para la Informática - Iutepi

```
    public int z;
    public Point3D(int x, int y, int z) :
        base(x, y)
    {
        this.z = z;
    }
}
```

Una clase hereda a los miembros de su clase base. La herencia significa que una clase contiene implícitamente todos los miembros de su clase base, excepto la instancia y los constructores estáticos, y los finalizadores de la clase base. Una clase derivada puede agregar nuevos miembros a aquellos de los que hereda, pero no puede quitar la definición de un miembro heredado. En el ejemplo anterior, Point3D hereda los campos x y y de Point y cada instancia de Point3Dcontiene tres campos: x, y y z.

Existe una conversión implícita de un tipo de clase a cualquiera de sus tipos de clase base. Por lo tanto, una variable de un tipo de clase puede hacer referencia a una instancia de esa clase o a una instancia de cualquier clase derivada. Por ejemplo, dadas las declaraciones de clase anteriores, una variable de tipo Point puede hacer referencia a una instancia de Point o Point3D:

```
Point a = new Point(10, 20);
Point b = new Point3D(10, 20, 30);
```

4.2 Campos

Un campo es una variable que está asociada con una clase o a una instancia de una clase.

Un campo declarado con el modificador "static" define un campo estático. Un campo estático identifica exactamente una ubicación de almacenamiento. Independientemente del número de instancias de una clase que se creen, siempre solo hay una copia de un campo estático.

Un campo declarado sin el modificador "static" define un campo de instancia. Cada instancia de una clase contiene una copia independiente de todos los campos de instancia de esa clase.

En el ejemplo siguiente, cada instancia de la clase Color tiene una copia independiente de los campos de instancia r, g y b, pero solo hay una copia de los campos estáticos Black, White, Red, Green y Blue:

```
public class Color
{
    public static readonly Color Black = new Color(0, 0, 0);
    public static readonly Color White = new Color(255, 255, 255);
    public static readonly Color Red = new Color(255, 0, 0);
```

Instituto Universitario de Tecnología para la Informática –IutepiInstituto Universitario de Tecnología para la Informática - Iutepi

```
    public static readonly Color Green = new Color(0, 255, 0);
    public static readonly Color Blue = new Color(0, 0, 255);
    private byte r, g, b;
    public Color(byte r, byte g, byte b)
    {
        this.r = r;
        this.g = g;
        this.b = b;
    }
}
```

4.3 Métodos

Un método es un miembro que implementa un cálculo o una acción que puede realizar un objeto o una clase. A los métodos estáticos se accede a través de la clase. A los métodos de instancia se accede a través de instancias de la clase.

Los métodos pueden tener una lista de parámetros, que representan valores o referencias a variables que se pasan al método, y un tipo de valor devuelto, que especifica el tipo del valor calculado y devuelto por el método. El tipo de valor devuelto de un método es void si no se devuelve un valor.

Al igual que los tipos, los métodos también pueden tener un conjunto de parámetros de tipo, para lo cuales se deben especificar argumentos de tipo cuando se llama al método. A diferencia de los tipos, los argumentos de tipo a menudo se pueden deducir de los argumentos de una llamada al método y no es necesario proporcionarlos explícitamente.

La signatura de un método debe ser única en la clase en la que se declara el método. La signatura de un método se compone del nombre del método, el número de parámetros de tipo y el número, los modificadores y los tipos de sus parámetros. La signatura de un método no incluye el tipo de valor devuelto.

4.4 Parámetros

Los parámetros se usan para pasar valores o referencias a variables a métodos. Los parámetros de un método obtienen sus valores reales de los argumentos que se especifican cuando se invoca el método. Hay cuatro tipos de parámetros: parámetros de valor, parámetros de referencia, parámetros de salida y matrices de parámetros.

Un parámetro de valor se usa para pasar argumentos de entrada. Un parámetro de valor corresponde a una variable local que obtiene su valor inicial del argumento que se ha pasado para el parámetro. Las modificaciones en un parámetro de valor no afectan el argumento que se pasa para el parámetro.

Los parámetros de valor pueden ser opcionales; se especifica un valor predeterminado para que se puedan omitir los argumentos correspondientes.

Instituto Universitario de Tecnología para la Informática –IutepiInstituto Universitario de Tecnología para la Informática - Iutepi

Un parámetro de referencia se usa para pasar argumentos mediante una referencia. El argumento pasado para un parámetro de referencia debe ser una variable con un valor definitivo, y durante la ejecución del método, el parámetro de referencia representa la misma ubicación de almacenamiento que la variable del argumento. Un parámetro de referencia se declara con el modificador ref. En el ejemplo siguiente se muestra el uso de parámetros ref.

```
using System;
class RefExample
{
   static void Swap(ref int x, ref int y)
   {
      int temp = x;
      x = y;
      y = temp;
   }
   public static void SwapExample()
   {
      int i = 1, j = 2;
      Swap(ref i, ref j);
      Console.WriteLine($"{i} {j}");    // Outputs "2 1"
   }
}
```

Un parámetro de salida se usa para pasar argumentos mediante una referencia. Es similar a un parámetro de referencia, excepto que no necesita que asigne un valor explícitamente al argumento proporcionado por el autor de la llamada. Un parámetro de salida se declara con el modificador out. En el siguiente ejemplo se muestra el uso de los parámetros out con la sintaxis que se ha presentado en C# 7.

```
using System;
   class OutExample
   {
      static void Divide(int x, int y, out int result, out int remainder)
      {
         result = x / y;
         remainder = x % y;
      }
      public static void OutUsage()
      {
         Divide(10, 3, out int res, out int rem);
         Console.WriteLine("{0} {1}", res, rem);           // Outputs "3 1"
      }
   }
```

Instituto Universitario de Tecnología para la Informática –IutepiInstituto Universitario de Tecnología para la Informática - Iutepi

}

4.5 Constructores

C# admite constructores de instancia y estáticos. Un constructor de instancia es un miembro que implementa las acciones necesarias para inicializar una instancia de una clase. Un constructor estático es un miembro que implementa las acciones necesarias para inicializar una clase en sí misma cuando se carga por primera vez.
Un constructor se declara como un método sin ningún tipo de valor devuelto y el mismo nombre que la clase contenedora. Si una declaración de constructor incluye un modificador "static", declara un constructor estático. De lo contrario, declara un constructor de instancia.
Los constructores de instancias se pueden sobrecargar y pueden tener parámetros opcionales. Por ejemplo, la clase List<T> declara dos constructores de instancia, una sin parámetros y otra que toma un parámetro int. Los constructores de instancia se invocan mediante el operador new. Las siguientes instrucciones asignan dos instancias List<string> mediante el constructor de la clase List con y sin el argumento opcional.

```
List<string> list1 = new List<string>();
List<string> list2 = new List<string>(10);
```

A diferencia de otros miembros, los constructores de instancia no se heredan y una clase no tiene ningún constructor de instancia que no sea el que se declara realmente en la clase. Si no se proporciona ningún constructor de instancia para una clase, se proporciona automáticamente uno vacío sin ningún parámetro.

4.6 Propiedades

Las propiedades son una extensión natural de los campos. Ambos son miembros con nombre con tipos asociados y la sintaxis para acceder a los campos y las propiedades es la misma. Sin embargo, a diferencia de los campos, las propiedades no denotan ubicaciones de almacenamiento. Las propiedades tienen descriptores de acceso que especifican las instrucciones que se ejecutan cuando se leen o escriben sus valores.
Una propiedad se declara como un campo, salvo que la declaración finaliza con un descriptor de acceso get y un descriptor de acceso set escrito entre los delimitadores { y } en lugar de finalizar en un punto y coma. Una propiedad que tiene un descriptor de acceso get y un descriptor de acceso set es una propiedad de lectura y escritura, una propiedad que tiene solo un descriptor de acceso get es una propiedad de solo lectura y una propiedad que tiene solo un descriptor de acceso set es una propiedad de solo escritura.
Un descriptor de acceso get corresponde a un método sin parámetros con un valor devuelto del tipo de propiedad. Excepto como destino de una asignación, cuando se hace referencia a

Instituto Universitario de Tecnología para la Informática –IutepiInstituto Universitario de Tecnología para la Informática - Iutepi

una propiedad en una expresión, el descriptor de acceso get de la propiedad se invoca para calcular el valor de la propiedad.
Un descriptor de acceso set corresponde a un método con un solo parámetro denominado value y ningún tipo de valor devuelto. Cuando se hace referencia a una propiedad como el destino de una asignación o como el operando de ++ o--, el descriptor de acceso set se invoca con un argumento que proporciona el nuevo valor.
La clase List<T> declara dos propiedades, Count y Capacity, que son de solo lectura y de lectura y escritura, respectivamente. El siguiente es un ejemplo de uso de estas propiedades.

```
List<string> names = new List<string>();
names.Capacity = 100;   // Invokes set accessor
int i = names.Count;    // Invokes get accessor
int j = names.Capacity; // Invokes get accessor
```

4.7 Indexador

Un indexador es un miembro que permite indexar de la misma manera que una matriz. Un indexador se declara como una propiedad, excepto por el hecho que el nombre del miembro va seguido por una lista de parámetros que se escriben entre los delimitadores [y]. Los parámetros están disponibles en los descriptores de acceso del indexador. De forma similar a las propiedades, los indexadores pueden ser lectura y escritura, de solo lectura y de solo escritura, y los descriptores de acceso de un indexador pueden ser virtuales.
La clase List declara un único indexador de lectura y escritura que toma un parámetro int. El indexador permite indexar instancias de List con valores int. Por ejemplo:

```
List<string> names = new List<string>();
names.Add("Liz");
names.Add("Martha");
names.Add("Beth");
for (int i = 0; i < names.Count; i++)
{
   string s = names[i];
   names[i] = s.ToUpper();
}
```

Los indexadores se pueden sobrecargar, lo que significa que una clase puede declarar varios indexadores siempre y cuando el número o los tipos de sus parámetros sean diferentes.

EVALUACIÓN

Definir una clase estática llamada Operaciones. Implementar cuatro métodos que permitan sumar,restar,multiplicar y dividir dos enteros

Instituto Universitario de Tecnología para la Informática –IutepiInstituto Universitario de Tecnología para la Informática - Iutepi

UNIDAD 5 Estructuras de datos

5.1 Estructura tipo vector

Un vector es una estructura de datos que permite almacenar un CONJUNTO de datos del MISMO tipo.
Con un único nombre se define un vector y por medio de un subíndice hacemos referencia a cada elemento del mismo (componente)

Problema 1:

Se desea guardar los sueldos de 5 operarios.
Según lo conocido deberíamos definir 5 variables si queremos tener en un cierto momento los 5 sueldos almacenados en memoria.
Empleando un vector solo se requiere definir un único nombre y accedemos a cada elemento por medio del subíndice.

sueldos

1200	750	820	550	490
sueldos[0]	sueldos[1]	sueldos[2]	sueldos[3]	sueldos[4]

Programa:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace PruebaVector1
{
   class PruebaVector1
   {
      private int[] sueldos;

      public void Cargar()
      {
         sueldos = new int[5];
         for (int f = 0; f < 5; f++)
         {
            Console.Write("Ingrese valor de la componente:");
            String linea;
            linea = Console.ReadLine();
            sueldos[f] = int.Parse(linea);
```

Instituto Universitario de Tecnología para la Informática –IutepiInstituto Universitario de Tecnología para la Informática - Iutepi

```
        }
    }

    public void Imprimir()
    {
        for(int f = 0; f < 5; f++)
        {
            Console.WriteLine(sueldos[f]);
        }
        Console.ReadKey();
    }

    static void Main(string[] args)
    {
        PruebaVector1 pv = new PruebaVector1();
        pv.Cargar();
        pv.Imprimir();
    }
  }
}
```

Para la declaración de un vector le antecedemos al nombre los corchetes abiertos y cerrados:

```
private int[] sueldos;
```

Lo definimos como atributo de la clase ya que lo utilizaremos en los dos métodos.
En el método de Cargar lo primero que hacemos es crear el vector (en C# los vectores son objetos por lo que es necesario proceder a su creación mediante el operador new):

```
sueldos = new int[5];
```

Cuando creamos el vector indicamos entre corchetes la cantidad de elementos que se pueden almacenar posteriormente en el mismo.
Para cargar cada componente debemos indicar entre corchetes que elemento del vector estamos accediendo:

```
for (int f = 0; f < 5; f++)
{
    Console.Write("Ingrese valor de la componente:");
    String linea;
    linea = Console.ReadLine();
    sueldos[f] = int.Parse(linea);
```


Instituto Universitario de Tecnología para la Informática –IutepiInstituto Universitario de Tecnología para la Informática - Iutepi

```
}
```

La estructura de programación que más se adapta para cargar en forma completa las componentes de un vector es un for, ya que sabemos de antemano la cantidad de valores a cargar.
Cuando f vale cero estamos accediendo a la primer componente del vector (en nuestro caso sería):

```
sueldos[f] = int.Parse(linea);
```

Lo mas común es utilizar una estructura repetitiva for para recorrer cada componente del vector.
Utilizar el for nos reduce la cantidad de código, si no utilizo un for debería en forma secuencial implementar el siguiente código:

```
string linea;
Console.Write("Ingrese valor de la componente:");
linea=Console.ReadLine();
sueldos[0]=int.Parse(linea);
Console.Write("Ingrese valor de la componente:");
linea=Console.ReadLine();
sueldos[1]=int.Parse(linea);
Console.Write("Ingrese valor de la componente:");
linea=Console.ReadLine();
sueldos[2]=int.Parse(linea);
Console.Write("Ingrese valor de la componente:");
linea=Console.ReadLine();
sueldos[3]=int.Parse(linea);
Console.Write("Ingrese valor de la componente:");
linea=Console.ReadLine();
sueldos[4]=int.Parse(linea);
```

La impresión de las componentes del vector lo hacemos en el otro método:

```
public void Imprimir()
{
  for(int f = 0; f < 5; f++)
  {
    Console.WriteLine(sueldos[f]);
  }
  Console.ReadKey();
}
```


Instituto Universitario de Tecnología para la Informática –IutepiInstituto Universitario de Tecnología para la Informática - Iutepi

Siempre que queremos acceder a una componente del vector debemos indicar entre corchetes la componente, dicho valor comienza a numerarse en cero y continua hasta un número menos del tamaño del vector, en nuestro caso creamos el vector con 5 elementos:

```
sueldos = new int[5];
```

Por último en este programa creamos un objeto en la Main y llamamos a lo métodos de Cargar e Imprimir el vector:

```
static void Main(string[] args)
{
    PruebaVector1 pv = new PruebaVector1();
    pv.Cargar();
    pv.Imprimir();
}
```

5.2 Estructura tipo matriz

Una matriz es una estructura de datos que contiene un número de variables a las que se accede mediante índices calculados. Las variables contenidas en una matriz, denominadas también elementos de la matriz, son todas del mismo tipo y este tipo se conoce como tipo de elemento de la matriz.

Los tipos de matriz son tipos de referencia, y la declaración de una variable de matriz simplemente establece un espacio reservado para una referencia a una instancia de matriz. Las instancias de matriz reales se crean dinámicamente en tiempo de ejecución mediante el nuevo operador. La nueva operación especifica la longitud de la nueva instancia de matriz, que luego se fija para la vigencia de la instancia. Los índices de los elementos de una matriz van de 0 a Length - 1. El operador newinicializa automáticamente los elementos de una matriz a su valor predeterminado, que, por ejemplo, es cero para todos los tipos numéricos y null para todos los tipos de referencias.

En el ejemplo siguiente se crea una matriz de elementos int, se inicializa la matriz y se imprime el contenido de la matriz.

```
using System;
class ArrayExample
{
  static void Main()
  {
    int[] a = new int[10];
    for (int i = 0; i < a.Length; i++)
    {
      a[i] = i * i;
    }
```


Instituto Universitario de Tecnología para la Informática –IutepiInstituto Universitario de Tecnología para la Informática - Iutepi

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

Este ejemplo se crea y se pone en funcionamiento en una matriz unidimensional. C# también admite matrices multidimensionales. El número de dimensiones de un tipo de matriz, conocido también como rango del tipo de matriz, es una más el número de comas escritas entre los corchetes del tipo de matriz. En el ejemplo siguiente se asignan una matriz unidimensional, multidimensional y tridimensional, respectivamente.

```
int[] a1 = new int[10];
int[,] a2 = new int[10, 5];
int[,,] a3 = new int[10, 5, 2];
```

La matriz a1 contiene 10 elementos, la matriz a2 50 (10 × 5) elementos y la matriz a3 100 (10 × 5 × 2) elementos. El tipo de elemento de una matriz puede ser cualquiera, incluido un tipo de matriz.Una matriz con elementos de un tipo de matriz a veces se conoce como matriz escalonada porque las longitudes de las matrices de elementos no tienen que ser iguales. En el ejemplo siguiente se asigna una matriz de matrices de int:

```
int[][] a = new int[3][];
a[0] = new int[10];
a[1] = new int[5];
a[2] = new int[20];
```

La primera línea crea una matriz con tres elementos, cada uno de tipo int[] y cada uno con un valor inicial de null. Las líneas posteriores inicializan entonces los tres elementos con referencias a instancias de matriz individuales de longitud variable.

El nuevo operador permite especificar los valores iniciales de los elementos de matriz mediante un inicializador de matriz, que es una lista de las expresiones escritas entre los delimitadores { y }.En el ejemplo siguiente se asigna e inicializa un tipo int[] con tres elementos.

```
int[] a = new int[] {1, 2, 3};
```

Tenga en cuenta que la longitud de la matriz se deduce del número de expresiones entre { y }. Las declaraciones de variable local y campo se pueden acortar más para que así no sea necesario reformular el tipo de matriz.

```
int[] a = {1, 2, 3};
```

Los dos ejemplos anteriores son equivalentes a lo siguiente:

```
int[] t = new int[3];
t[0] = 1;
t[1] = 2;
t[2] = 3;
int[] a = t;
```

Instituto Universitario de Tecnología para la Informática –IutepiInstituto Universitario de Tecnología para la Informática - Iutepi

EVALUACIÓN

1) Definir un vector de 5 componentes de tipo float que representen las alturas de 5 personas.
Obtener el promedio de las mismas. Contar cuántas personas son más altas que el promedio y cuántas más bajas.
2) Cargar un vector de 10 elementos y verificar posteriormente si el mismo está ordenado de menor a mayor.

UNIDAD 6 ESTRUCTURAS DINÁMICAS

6.1 Generalidades

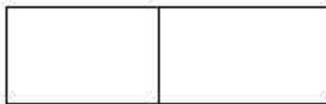
Conocemos algunas estructuras de datos como son los vectores y matrices. No son las únicas. Hay muchas situaciones donde utilizar alguna de estas estructuras nos proporcionará una solución muy ineficiente (cantidad de espacio que ocupa en memoria, velocidad de acceso a la información, etc.)
Ejemplo 1. Imaginemos que debemos realizar un procesador de texto, debemos elegir la estructura de datos para almacenar en memoria las distintas líneas que el operador irá tipeando. Una solución factible es utilizar una matriz de caracteres. Pero como sabemos debemos especificar la cantidad de filas y columnas que ocupará de antemano. Podría ser por ejemplo 2000 filas y 200 columnas. Con esta definición estamos reservando de antemano 800000 bytes de la memoria, no importa si el operador después carga una línea con 20 caracteres, igualmente ya se ha reservado una cantidad de espacio que permanecerá ociosa.
Tiene que existir alguna estructura de datos que pueda hacer más eficiente la solución del problema anterior.
Ejemplo 2. ¿Cómo estarán codificadas las planillas de cálculo? ¿Reservarán espacio para cada casilla de la planilla al principio? Si no la lleno, ¿lo mismo se habrá reservado espacio?
Utilizar una matriz para almacenar todas las casillas de una planilla de cálculo seguro será ineficiente.
Bien, todos estos problemas y muchos más podrán ser resueltos en forma eficiente cuando conozcamos estas nuevas estructuras de datos (Listas, árboles)

6.2 Listas

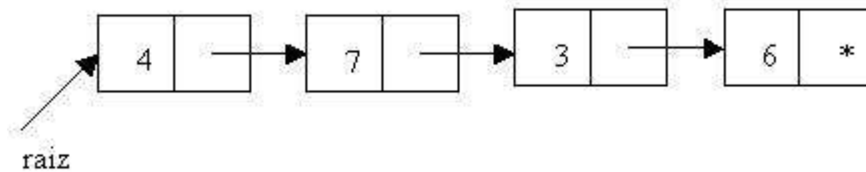
Una lista es un conjunto de nodos, cada uno de los cuales tiene dos campos: uno de información y un apuntador al siguiente nodo de la lista. Además un apuntador externo señala el primer nodo de la lista.
Representación gráfica de un nodo:

Instituto Universitario de Tecnología para la Informática –IutepiInstituto Universitario de Tecnología para la Informática - Iutepi

Información Dirección al siguiente nodo.

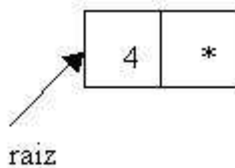


La información puede ser cualquier tipo de dato simple, estructura de datos o inclusive uno o más objetos.
La dirección al siguiente nodo es un puntero.
Representación gráfica de una lista:

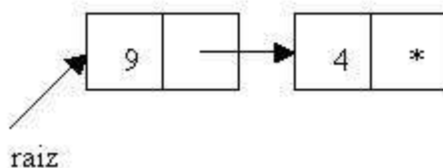


Como decíamos, una lista es una secuencia de nodos (en este caso cuatro nodos). La información de los nodos en este caso es un entero y siempre contiene un puntero que guarda la dirección del siguiente nodo.
raiz es otro puntero externo a la lista que contiene la dirección del primer nodo.
El estado de una lista varía durante la ejecución del programa:

De esta forma representamos gráficamente una lista vacía.
Si insertamos un nodo en la lista quedaría luego:



Si insertamos otro nodo al principio con el valor 9 tenemos:



Lo mismo podemos borrar nodos de cualquier parte de la lista.
Esto nos trae a la mente el primer problema planteado: el desarrollo del procesador de texto. Podríamos utilizar una lista que inicialmente estuviera vacía e introdujéramos un nuevo nodo con cada línea que tipea el operador. Con esta estructura haremos un uso muy eficiente de la memoria.
Tipos de listas.

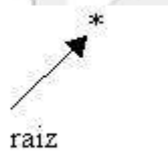
Instituto Universitario de Tecnología para la Informática –IutepiInstituto Universitario de Tecnología para la Informática - Iutepi

Según el mecanismo de inserción y extracción de nodos en la lista tenemos los siguientes tipos:
Listas tipo pila.
Listas tipo cola.
Listas genéricas.
Una lista se comporta como una pila si las inserciones y extracciones las hacemos por un mismo lado de la lista. También se las llama listas LIFO (Last In First Out - último en entrar primero en salir)
Una lista se comporta como una cola si las inserciones las hacemos al final y las extracciones las hacemos por el frente de la lista. También se las llama listas FIFO (First In First Out - primero en entrar primero en salir)
Una lista se comporta como genérica cuando las inserciones y extracciones se realizan en cualquier parte de la lista.

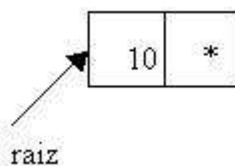
Podemos en algún momento insertar un nodo en medio de la lista, en otro momento al final, borrar uno del frente, borrar uno del fondo o uno interior, etc.

6.3 Listas tipo pila

Una lista se comporta como una pila si las inserciones y extracciones las hacemos por un mismo lado de la lista. También se las llama listas LIFO (Last In First Out - último en entrar primero en salir)
Importante: Una pila al ser una lista puede almacenar en el campo de información cualquier tipo de valor (int, char, float, vector de caracteres, un objeto, etc)
Para estudiar el mecanismo de utilización de una pila supondremos que en el campo de información almacena un entero (para una fácil interpretación y codificación)
Inicialmente la PILA está vacía y decimos que el puntero raiz apunta a null (Si apunta a null decimos que no tiene una dirección de memoria):

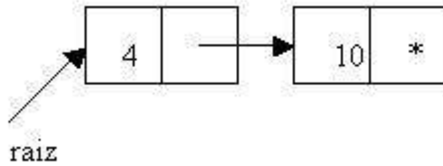


Insertamos un valor entero en la pila: insertar(10)



Luego de realizar la inserción la lista tipo pila queda de esta manera: un nodo con el valor 10 y raíz apunta a dicho nodo. El puntero del nodo apunta a null ya que no hay otro nodo después de este.
Insertamos luego el valor 4: insertar(4)

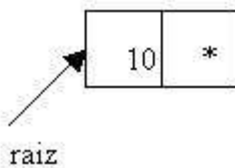
Instituto Universitario de Tecnología para la Informática –IutepiInstituto Universitario de Tecnología para la Informática - Iutepi



Ahora el primer nodo de la pila es el que almacena el valor cuatro. raiz apunta a dicho nodo. Recordemos que raiz es el puntero externo a la lista que almacena la dirección del primer nodo. El nodo que acabamos de insertar en el campo puntero guarda la dirección del nodo que almacena el valor 10.

Ahora qué sucede si extraemos un nodo de la pila. ¿Cuál se extrae? Como sabemos en una pila se extrae el último en entrar.

Al extraer de la pila tenemos: extraer()



La pila ha quedado con un nodo.

Hay que tener cuidado que si se extrae un nuevo nodo la pila quedará vacía y no se podrá extraer otros valores (avisar que la pila está vacía)

Problema 1:

Confeccionar una clase que administre una lista tipo pila (se debe poder insertar, extraer e imprimir los datos de la pila)

Programa:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ListasTipoPila1
{
    class Pila
    {
        class Nodo
        {
            public int info;
            public Nodo sig;
        }

        private Nodo raiz;

        public Pila()
```


Instituto Universitario de Tecnología para la Informática –IutepiInstituto Universitario de Tecnología para la Informática - Iutepi

```
{
   raiz = null;
}

public void Insertar(int x)
{
   Nodo nuevo;
   nuevo = new Nodo();
   nuevo.info = x;
   if (raiz == null)
   {
      nuevo.sig = null;
      raiz = nuevo;
   }
   else
   {
      nuevo.sig = raiz;
      raiz = nuevo;
   }
}

public int Extraer()
{
   if (raiz != null)
   {
      int informacion = raiz.info;
      raiz = raiz.sig;
      return informacion;
   }
   else
   {
      return int.MaxValue;
   }
}

public void Imprimir()
{
   Nodo reco=raiz;
   Console.WriteLine("Listado de todos los elementos de la pila.");
   while (reco!=null)
   {
```

Instituto Universitario de Tecnología para la Informática –IutepiInstituto Universitario de Tecnología para la Informática - Iutepi

```
            Console.Write(reco.info+"-");
            reco=reco.sig;
        }
        Console.WriteLine();
    }

    static void Main(string[] args)
    {
        Pila pila1=new Pila();
        pila1.Insertar(10);
        pila1.Insertar(40);
        pila1.Insertar(3);
        pila1.Imprimir();
        Console.WriteLine("Extraemos de la pila:"+pila1.Extraer());
        pila1.Imprimir();
        Console.ReadKey();
    }
  }
}
```

Analicemos las distintas partes de este programa:

```
    class Nodo
    {
        public int info;
        public Nodo sig;
    }

    private Nodo raiz;
```

Para declarar un nodo debemos utilizar una clase. En este caso la información del nodo (info) es un entero y siempre el nodo tendrá una referencia de tipo Nodo, que le llamamos sig.

El puntero sig apunta al siguiente nodo o a null en caso que no exista otro nodo. Este puntero es interno a la lista. Para poder acceder a los atributos los definimos de tipo public.

También definimos un puntero de tipo Nodo llamado raiz. Este puntero tiene la dirección del primer nodo de la lista. En caso de estar vacía la lista, raiz apunta a null (es decir no tiene dirección)

El puntero raiz es fundamental porque al tener la dirección del primer nodo de la lista nos permite acceder a los demás nodos.

```
    public Pila()
    {
```

Instituto Universitario de Tecnología para la Informática –IutepiInstituto Universitario de Tecnología para la Informática - Iutepi

```
    raiz = null;
}
```

En el constructor de la clase hacemos que raíz guarde el valor null. Tengamos en cuenta que si raíz tiene almacenado null la lista está vacía, en caso contrario tiene la dirección del primer nodo de la lista.

```
public void Insertar(int x)
{
    Nodo nuevo;
    nuevo = new Nodo();
    nuevo.info = x;
    if (raiz == null)
    {
        nuevo.sig = null;
        raiz = nuevo;
    }
    else
    {
        nuevo.sig = raiz;
        raiz = nuevo;
    }
}
```

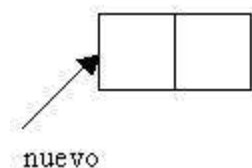
Uno de los métodos más importantes que debemos entender en una pila es el de Insertar un elemento en la pila.
Al método llega la información a insertar, en este caso en particular es un valor entero.
La creación de un nodo requiere dos pasos:
- Definición de un puntero o referencia a un tipo de dato Nodo:

```
Nodo nuevo;
```

- Creación del nodo (creación de un objeto):

```
nuevo = new Nodo();
```

Cuando se ejecuta el operador new se reserva espacio para el nodo. Realmente se crea el nodo cuando se ejecuta el new.

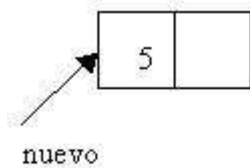


Paso seguido debemos guardar la información del nodo:

```
nuevo.info = x;
```

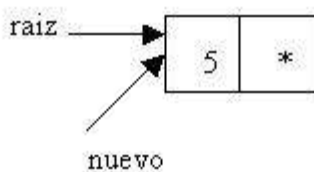
Instituto Universitario de Tecnología para la Informática –IutepiInstituto Universitario de Tecnología para la Informática - Iutepi

En el campo info almacenamos lo que llega en el parámetro x. Por ejemplo si llega un 5 el nodo queda:



Por último queda enlazar el nodo que acabamos de crear al principio de la lista.
Si la lista está vacía debemos guardar en el campo sig del nodo el valor null para indicar que no hay otro nodo después de este, y hacer que raiz apunte al nodo creado (sabemos si una lista esta vacía si raíz almacena un null)

```
if (raíz == null)
{
    nuevo.sig = null;
    raíz = nuevo;
}
```

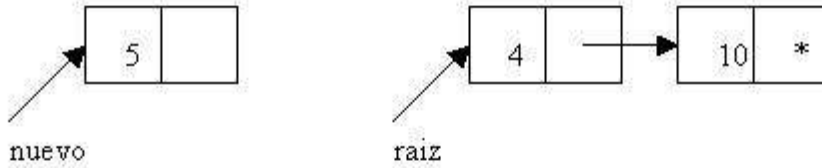


Gráficamente podemos observar que cuando indicamos raiz=nuevo, el puntero raiz guarda la dirección del nodo apuntado por nuevo.
Tener en cuenta que cuando finaliza la ejecución del método el puntero nuevo desaparece, pero no el nodo creado con el operador new.
En caso que la lista no esté vacía, el puntero sig del nodo que acabamos de crear debe apuntar al que es hasta este momento el primer nodo, es decir al nodo que apunta raiz actualmente.

```
else
{
    nuevo.sig = raiz;
    raiz = nuevo;
}
```

Como primera actividad cargamos en el puntero sig del nodo apuntado por nuevo la dirección de raiz, y posteriormente raíz apunta al nodo que acabamos de crear, que será ahora el primero de la lista.
Antes de los enlaces tenemos:

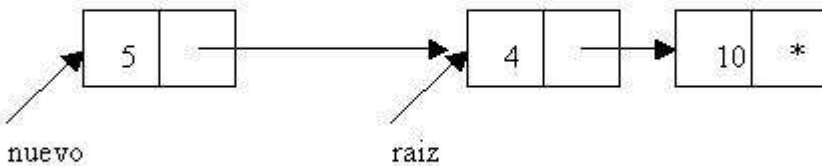
Instituto Universitario de Tecnología para la Informática –IutepiInstituto Universitario de Tecnología para la Informática - Iutepi



Luego de ejecutar la línea:

nuevo.sig = raiz;

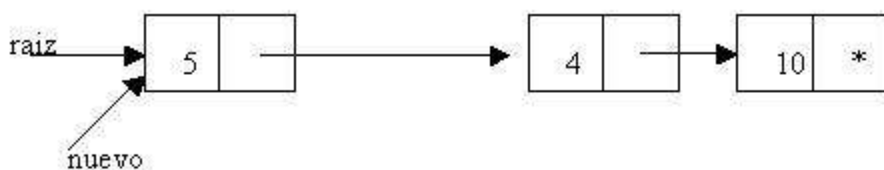
Ahora tenemos:



Por último asignamos a raíz la dirección que almacena el puntero nuevo.

raiz = nuevo;

La lista queda:



El método Extraer:

```
public int Extraer()
{
    if (raiz != null)
    {
        int informacion = raiz.info;
        raiz = raiz.sig;
        return informacion;
    }
    else
    {
        return int.MaxValue;
    }
}
```

El objetivo del método extraer es retornar la información del primer nodo y además borrarlo de la lista.

Si la lista no está vacía guardamos en una variable local la información del primer nodo:

Instituto Universitario de Tecnología para la Informática –IutepiInstituto Universitario de Tecnología para la Informática - Iutepi

```
int informacion = raiz.info;
```

Avanzamos raiz al segundo nodo de la lista, ya que borraremos el primero:

```
raiz = raiz.sig;
```

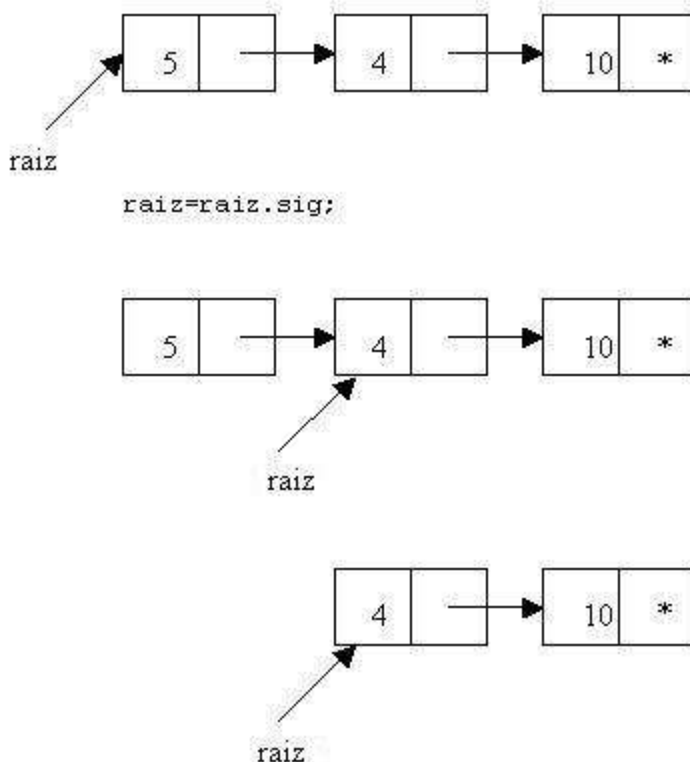
El nodo que previamente estaba apuntado por raíz es eliminado automáticamente, al no tener ninguna referencia.
Retornamos la información:

```
return informacion;
```

En caso de estar vacía la pila retornamos el número entero máximo y lo tomamos como código de error (es decir nunca debemos guardar el entero mayor en la pila)

```
return int.MaxValue;
```

Es muy importante entender gráficamente el manejo de las listas. La interpretación gráfica nos permitirá plantear inicialmente las soluciones para el manejo de listas.



Por último expliquemos el método para recorrer una lista en forma completa e imprimir la información de cada nodo:

```
public void Imprimir()
{
    Nodo reco=raiz;
    Console.WriteLine("Listado de todos los elementos de la pila.");
    while (reco!=null)
```


Instituto Universitario de Tecnología para la Informática –IutepiInstituto Universitario de Tecnología para la Informática - Iutepi

```
    {
       Console.Write(reco.info+"-");
       reco=reco.sig;
    }
    Console.WriteLine();
}
```

Definimos un puntero auxiliar reco y hacemos que apunte al primer nodo de la lista:

```
Nodo reco=raiz;
```

Disponemos una estructura repetitiva que se repetirá mientras reco sea distinto a null. Dentro de la estructura repetitiva hacemos que reco avance al siguiente nodo:

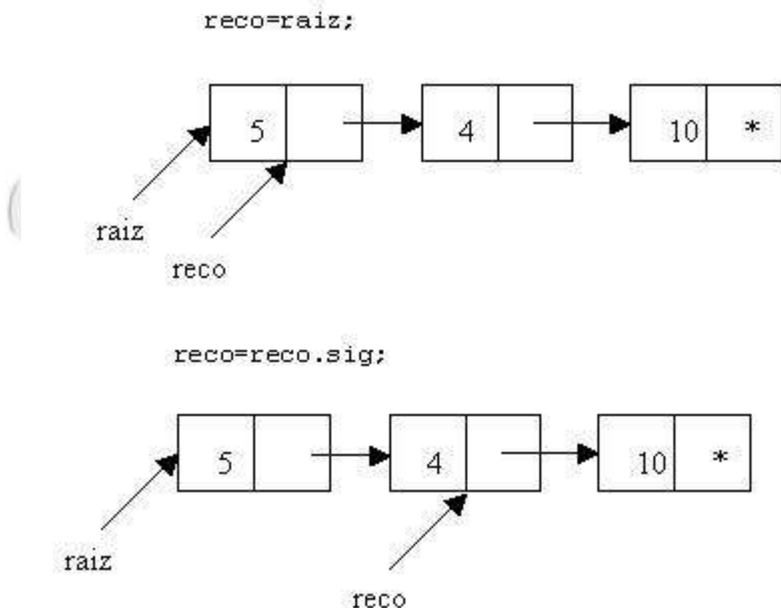
```
while (reco!=null)
{
   Console.Write(reco.info+"-");
   reco=reco.sig;
}
```

Es muy importante entender la línea:

```
reco=reco.sig;
```

Estamos diciendo que reco almacena la dirección que tiene el puntero sig del nodo apuntado actualmente por reco.

Gráficamente:



Al analizarse la condición:

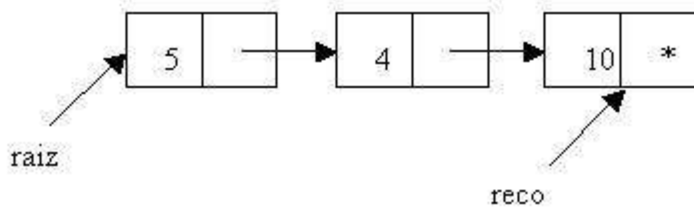
```
while (reco!=null)
```

se valúa en verdadero ya que reco apunta a un nodo y se vuelve a ejecutar la línea:

Instituto Universitario de Tecnología para la Informática –IutepiInstituto Universitario de Tecnología para la Informática - Iutepi

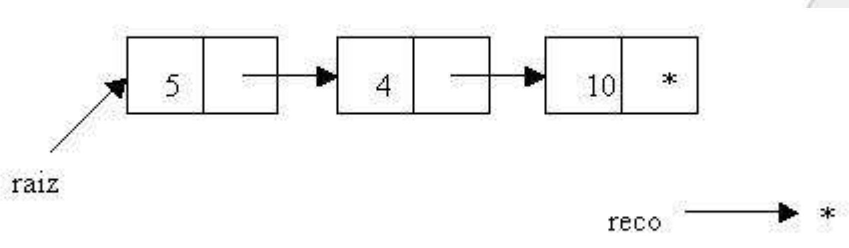
reco=reco.sig;

Ahora reco apunta al siguiente nodo:



La condición del while nuevamente se valúa en verdadera y avanza el puntero reco al siguiente nodo:

reco=reco.sig;



Ahora sí reco apunta a null y ha llegado el final de la lista (Recordar que el último nodo de la lista tiene almacenado en el puntero sig el valor null, con el objetivo de saber que es el último nodo)

Para poder probar esta clase recordemos que debemos definir un objeto de la misma y llamar a sus métodos:

```
static void Main(string[] args)
{
    Pila pila1=new Pila();
    pila1.Insertar(10);
    pila1.Insertar(40);
    pila1.Insertar(3);
    pila1.Imprimir();
    Console.WriteLine("Extraemos de la pila:"+pila1.Extraer());
    pila1.Imprimir();
    Console.ReadKey();
}
```

Insertamos 3 enteros, luego imprimimos la pila, extraemos uno de la pila y finalmente imprimimos nuevamente la pila.

Problema 2:

Agregar a la clase Pila un método que retorne la cantidad de nodos y otro que indique si esta vacía.

Instituto Universitario de Tecnología para la Informática –IutepiInstituto Universitario de Tecnología para la Informática - Iutepi

Programa:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ListasTipoPila1
{
  class Pila
  {
    class Nodo
    {
      public int info;
      public Nodo sig;
    }

    private Nodo raiz;

    public Pila()
    {
      raiz = null;
    }

    public void Insertar(int x)
    {
      Nodo nuevo;
      nuevo = new Nodo();
      nuevo.info = x;
      if (raiz == null)
      {
        nuevo.sig = null;
        raiz = nuevo;
      }
      else
      {
        nuevo.sig = raiz;
        raiz = nuevo;
      }
    }

    public int Extraer()
```


Instituto Universitario de Tecnología para la Informática –IutepiInstituto Universitario de Tecnología para la Informática - Iutepi

Instituto Universitario de Tecnología para la Informática –IutepiInstituto Universitario de Tecnología para la Informática - Iutepi

```
{
   if (raiz != null)
   {
      int informacion = raiz.info;
      raiz = raiz.sig;
      return informacion;
   }
   else
   {
      return int.MaxValue;
   }
}

public void Imprimir()
{
   Nodo reco=raiz;
   Console.WriteLine("Listado de todos los elementos de la pila.");
   while (reco!=null)
   {
      Console.Write(reco.info+"-");
      reco=reco.sig;
   }
   Console.WriteLine();
}

public bool Vacia()
{
   if (raiz == null)
   {
      return true;
   }
   else
   {
      return false;
   }
}

public int Cantidad()
{
   int cant = 0;
   Nodo reco = raiz;
```

```
        while (reco != null)
        {
            cant++;
            reco = reco.sig;
        }
        return cant;
    }

    static void Main(string[] args)
    {
        Pila pila1=new Pila();
        pila1.Insertar(10);
        pila1.Insertar(40);
        pila1.Insertar(3);
        pila1.Imprimir();
        Console.WriteLine("La cantidad de nodos de la lista es:"+pila1.Cantidad());
        while (pila1.Vacia()==false)
        {
            Console.WriteLine(pila1.Extraer());
        }
        Console.ReadKey();
    }
  }
}
```

Para verificar si la pila esta vacía verificamos el contenido de la variable raiz, si tiene null luego la lista esta vacía y por lo tanto retornamos un true:

```
    public bool Vacia()
    {
        if (raiz == null)
        {
            return true;
        }
        else
        {
            return false;
        }
    }
```

El algoritmo para saber la cantidad de nodos es similar al imprimir, pero en lugar de mostrar la información del nodo procedemos a incrementar un contador:

Instituto Universitario de Tecnología para la Informática –IutepiInstituto Universitario de Tecnología para la Informática - Iutepi

```
public int Cantidad()
{
   int cant = 0;
   Nodo reco = raiz;
   while (reco != null)
   {
      cant++;
      reco = reco.sig;
   }
   return cant;
}
```

Para probar esta clase en la main creamos un objeto de la clase Pila insertamos tres enteros:

```
Pila pila1=new Pila();
pila1.Insertar(10);
pila1.Insertar(40);
pila1.Insertar(3);
```

Imprimimos la pila (nos muestra los tres datos):

```
pila1.Imprimir();
```

Llamamos al método Cantidad (nos retorna un 3):

```
Console.WriteLine("La cantidad de nodos de la lista es:"+pila1.Cantidad());
```

Luego mientras el método Vacia nos retorne un false (lista no vacía) procedemos a llamar al método extraer:

```
while (pila1.Vacia()==false)
{
   Console.WriteLine(pila1.Extraer());
}
```

6.4 Listas tipo cola

Una lista se comporta como una cola si las inserciones las hacemos al final y las extracciones las hacemos por el frente de la lista. También se las llama listas FIFO (First In First Out - primero en entrar primero en salir)

Confeccionaremos un programa que permita administrar una lista tipo cola. Desarrollaremos los métodos de Insertar, Extraer, Vacia e Imprimir.

Programa:

```
using System;
using System.Collections.Generic;
```

Instituto Universitario de Tecnología para la Informática –IutepiInstituto Universitario de Tecnología para la Informática - Iutepi

```
using System.Linq;
using System.Text;

namespace ListasTipoCola1
{
   class Cola
   {
      class Nodo
      {
         public int info;
         public Nodo sig;
      }

      private Nodo raiz,fondo;

      public Cola()
      {
         raiz=null;
         fondo=null;
      }

      public bool Vacia ()
      {
         if (raiz == null)
            return true;
         else
            return false;
      }

      public void Insertar (int info)
      {
         Nodo nuevo;
         nuevo = new Nodo ();
         nuevo.info = info;
         nuevo.sig = null;
         if (Vacia ())
         {
            raiz = nuevo;
            fondo = nuevo;
         }
         else
```

Instituto Universitario de Tecnología para la Informática –IutepiInstituto Universitario de Tecnología para la Informática - Iutepi

```
    {
        fondo.sig = nuevo;
        fondo = nuevo;
    }
}

public int Extraer ()
{
    if (!Vacia ())
    {
        int informacion = raiz.info;
        if (raiz == fondo)
        {
            raiz = null;
            fondo = null;
        }
        else
        {
            raiz = raiz.sig;
        }
        return informacion;
    }
    else
        return int.MaxValue;
}

public void Imprimir()
{
    Nodo reco=raiz;
    Console.WriteLine("Listado de todos los elementos de la cola.");
    while (reco!=null)
    {
        Console.Write(reco.info+"-");
        reco=reco.sig;
    }
    Console.WriteLine();
}

static void Main(string[] args)
{
    Cola cola1=new Cola();
```


Instituto Universitario de Tecnología para la Informática –IutepiInstituto Universitario de Tecnología para la Informática - Iutepi

```
            cola1.Insertar(5);
            cola1.Insertar(10);
            cola1.Insertar(50);
            cola1.Imprimir();
            Console.WriteLine("Extraemos uno de la cola:"+cola1.Extraer());
            cola1.Imprimir();
            Console.ReadKey();
        }
    }
}
```

La declaración del nodo es igual a la clase Pila. Luego definimos dos punteros externos:

```
        class Nodo
        {
            public int info;
            public Nodo sig;
        }

        private Nodo raiz,fondo;
```

Destacando que raíz apunta al principio de la lista y fondo al final de la lista. Utilizar dos punteros tiene como ventaja que cada vez que tengamos que insertar un nodo al final de la lista no tengamos que recorrerla. Por supuesto que es perfectamente válido implementar una cola con un único puntero externo a la lista.

En el constructor inicializamos a los dos punteros en null (Realmente esto es opcional ya que los atributos de una clase en C# se inicializan automáticamente con null):

```
        public Cola()
        {
            raiz=null;
            fondo=null;
        }
```

El método vacía retorna true si la lista no tiene nodos y false en caso contrario:

```
        public bool Vacia ()
        {
            if (raiz == null)
                return true;
            else
                return false;
        }
```

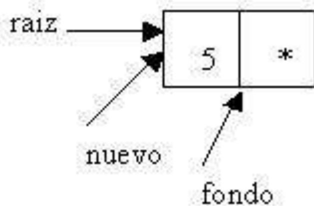

Instituto Universitario de Tecnología para la Informática –IutepiInstituto Universitario de Tecnología para la Informática - Iutepi

En la inserción luego de crear el nodo tenemos dos posibilidades: que la cola esté vacía, en cuyo caso los dos punteros externos a la lista deben apuntar al nodo creado, o que haya nodos en la lista.

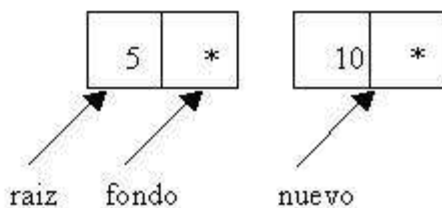
```
Nodo nuevo;
 nuevo = new Nodo ();
 nuevo.info = info;
 nuevo.sig = null;
 if (Vacia ())
 {
     raiz = nuevo;
     fondo = nuevo;
 }
 else
 {
     fondo.sig = nuevo;
     fondo = nuevo;
 }
```

Recordemos que definimos un puntero llamado nuevo, luego creamos el nodo con el operador new y cargamos los dos campos, el de información con lo que llega en el parámetro y el puntero con null ya que se insertará al final de la lista, es decir no hay otro después de este.

Si la lista está vacía:



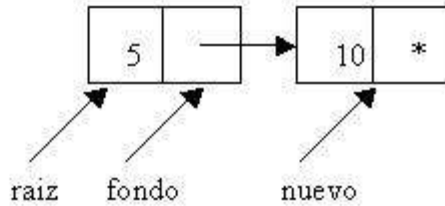
En caso de no estar vacía:



Debemos enlazar el puntero sig del último nodo con el nodo recién creado:

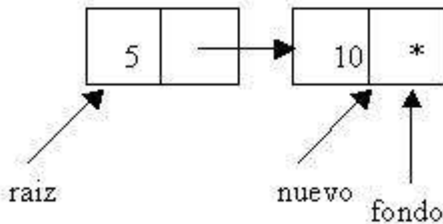
```
fondo.sig = nuevo;
```


Instituto Universitario de Tecnología para la Informática –IutepiInstituto Universitario de Tecnología para la Informática - Iutepi



Y por último el puntero externo fondo debe apuntar al nodo apuntado por nuevo:

```
fondo = nuevo;
```



Con esto ya tenemos correctamente enlazados los nodos en la lista tipo cola. Recordar que el puntero nuevo desaparece cuando se sale del método insertar, pero el nodo creado no se pierde porque queda enlazado en la lista.

El funcionamiento del método extraer es similar al de la pila:

```
public int Extraer ()
{
   if (!Vacia ())
   {
      int informacion = raiz.info;
      if (raiz == fondo)
      {
         raiz = null;
         fondo = null;
      }
      else
      {
         raiz = raiz.sig;
      }
      return informacion;
   }
   else
      return int.MaxValue;
}
```

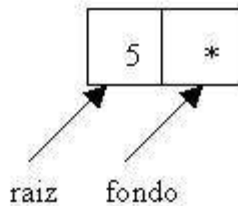
Si la lista no está vacía guardamos en una variable local la información del primer nodo:

```
int informacion = raiz.info;
```


Instituto Universitario de Tecnología para la Informática –IutepiInstituto Universitario de Tecnología para la Informática - Iutepi

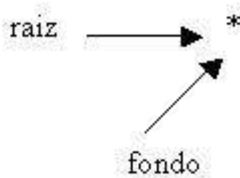
Para saber si hay un solo nodo verificamos si los dos punteros raiz y fondo apuntan a la misma dirección de memoria:

if (raiz == fondo)

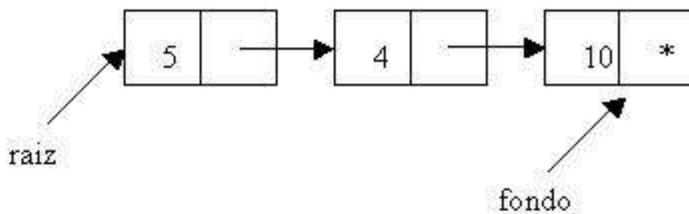


Luego hacemos:

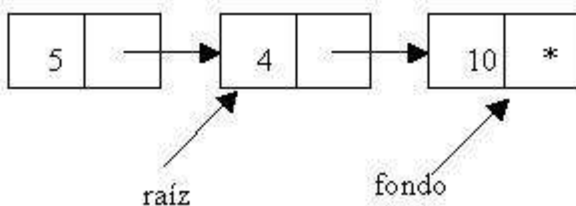
raiz = null;
fondo = null;



En caso de haber 2 o más nodos debemos avanzar el puntero raiz al siguiente nodo:



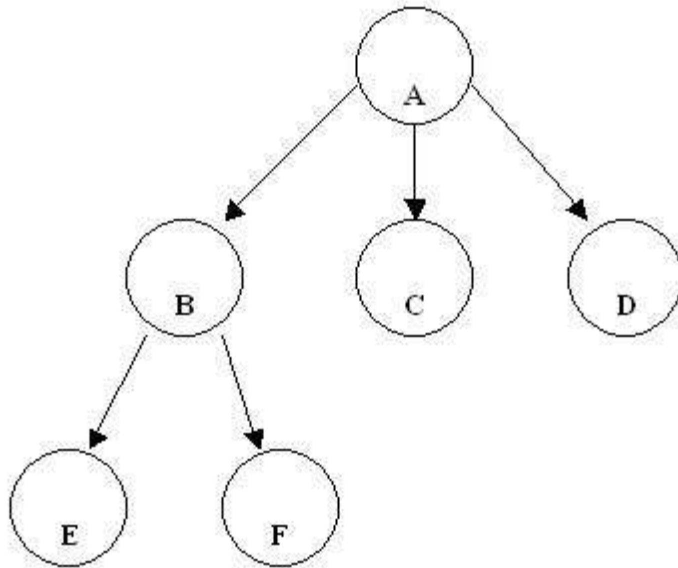
raiz = raiz.sig;



6.5 Arboles

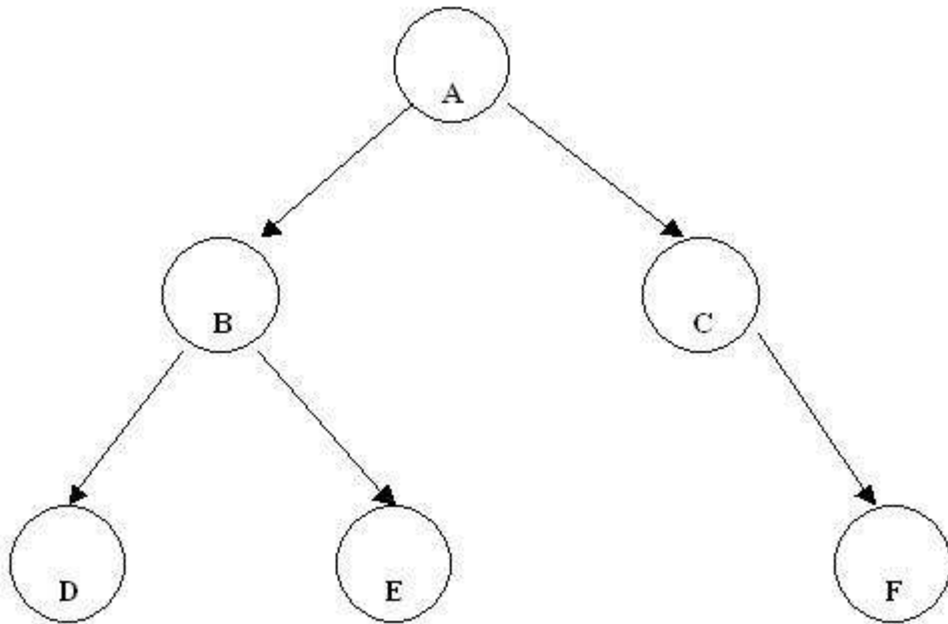
Igual que la lista, el árbol es una estructura de datos. Son muy eficientes para la búsqueda de información. Los árboles soportan estructuras no lineales.

Instituto Universitario de Tecnología para la Informática –IutepiInstituto Universitario de Tecnología para la Informática - Iutepi



Algunos conceptos de la estructura de datos tipo árbol:
Nodo hoja: Es un nodo sin descendientes (Nodo terminal)
Ej. Nodos E – F – C y D.
Nodo interior: Es un nodo que no es hoja.
Ej. Nodos A y B.
Nivel de un árbol: El nodo A está en el nivel 1 sus descendientes directos están en el nivel 2 y así sucesivamente.
El nivel del árbol está dado por el nodo de máximo nivel.
Ej. Este árbol es de nivel 3.
Grado de un nodo: es el número de nodos hijos que tiene dicho nodo (solo se tiene en cuenta los nodos interiores)
Ej. El nodo A tiene grado 3.
El nodo B tiene grado 2.
Los otros nodos no tienen grado porque no tienen descendientes.
Grado de un árbol: Es el máximo de los grados de todos los nodos de un árbol.
Ej. El grado del árbol es 3.
Longitud de camino del nodo x: Al número de arcos que deben ser recorridos para llegar a un nodo x, partiendo de la raiz.
La raiz tiene longitud de camino 1, sus descendientes directos tienen longitud de camino 2, etc. En forma general un nodo en el nivel i tiene longitud de camino i.
Árbol binario: Un árbol es binario si cada nodo tiene como máximo 2 descendientes.

Instituto Universitario de Tecnología para la Informática –IutepiInstituto Universitario de Tecnología para la Informática - Iutepi



Para cada nodo está definido el subárbol izquierdo y el derecho.

Para el nodo A el subárbol izquierdo está constituido por los nodos B, D y E. Y el subárbol derecho está formado por los nodos C y F.

Lo mismo para el nodo B tiene el subárbol izquierdo con un nodo (D) y un nodo en el subárbol derecho (E).

El nodo D tiene ambos subárboles vacíos.

El nodo C tiene el subárbol izquierdo vacío y el subárbol derecho con un nodo (F).

Árbol binario perfectamente equilibrado: Si para cada nodo el número de nodos en el subárbol izquierdo y el número de nodos en el subárbol derecho, difiere como mucho en una unidad.

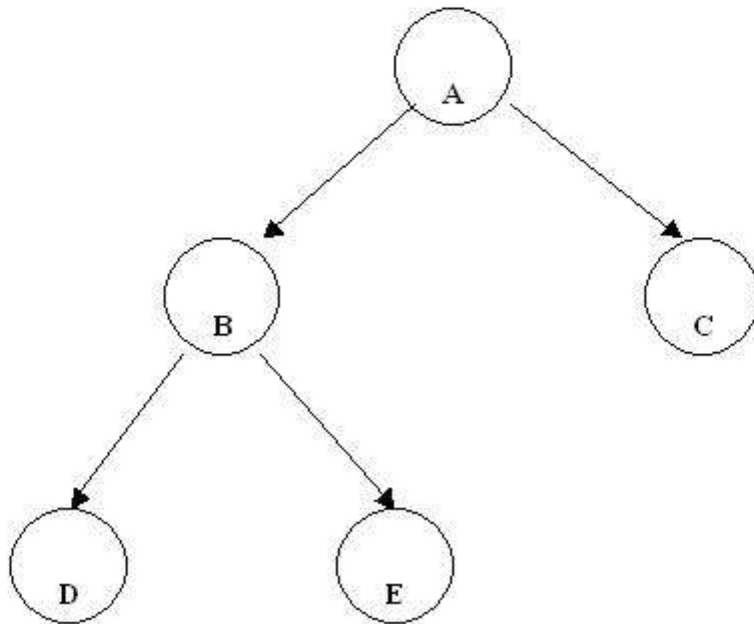
Hay que tener en cuenta todos los nodos del árbol.

El árbol de más arriba es perfectamente equilibrado.

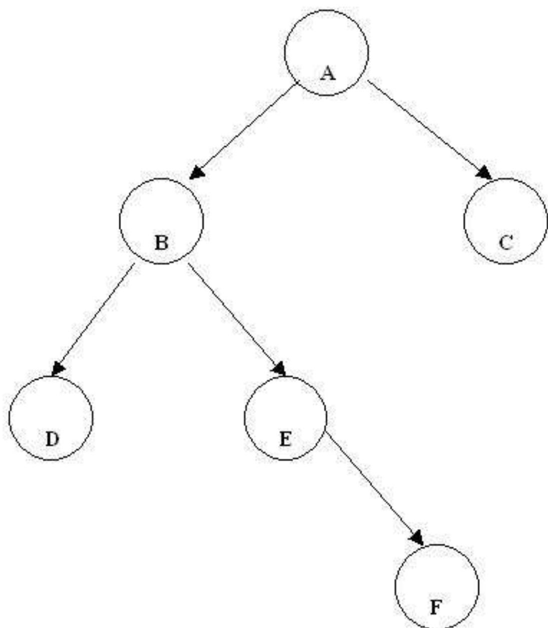
Ej. árbol que no es perfectamente equilibrado:

El nodo A tiene 3 nodos en el subárbol izquierdo y solo uno en el subárbol derecho, por lo que no es perfectamente equilibrado.

Instituto Universitario de Tecnología para la Informática –IutepiInstituto Universitario de Tecnología para la Informática - Iutepi

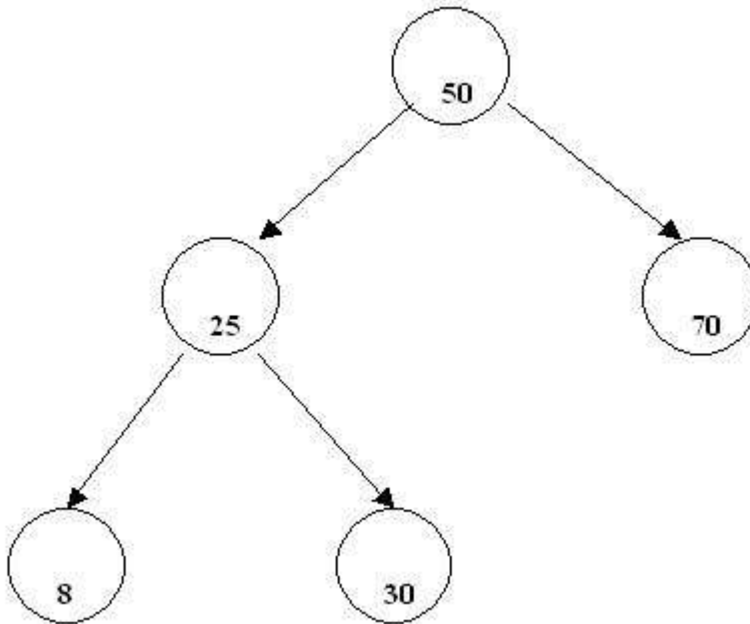


Árbol binario completo: Es un árbol binario con hojas como máximo en los niveles n-1 y n (Siendo n el nivel del árbol)
Los dos árboles graficados son completos porque son árboles de nivel 3 y hay nodos hoja en el nivel 3 en el primer caso, y hay nodos hoja en los niveles 3 y 2 en el segundo caso.
Ej. Árbol binario no completo:



Hay nodos hoja en los niveles 4, 3 y 2. No debería haber nodos hojas en el nivel 2.
Árbol binario ordenado: Si para cada nodo del árbol, los nodos ubicados a la izquierda son inferiores al que consideramos raíz para ese momento y los nodos ubicados a la derecha son mayores que la raíz.

Instituto Universitario de Tecnología para la Informática –IutepiInstituto Universitario de Tecnología para la Informática - Iutepi



Ej. Analicemos si se trata de un árbol binario ordenado:
Para el nodo que tiene el 50:
Los nodos del subárbol izquierdo son todos menores a 50? 8, 25, 30 Si
Los nodos del subárbol derecho son todos mayores a 50? 70 Si.
Para el nodo que tiene el 25:
Los nodos del subárbol izquierdo son todos menores a 25? 8 Si
Los nodos del subárbol derecho son todos mayores a 25? 30 Si.
No hace falta analizar los nodos hoja. Si todas las respuestas son afirmativas podemos luego decir que se trata de un árbol binario ordenado.

EVALUACIÓN

Agregar un método a la clase Pila que retorne la información del primer nodo de la Pila sin borrarlo.
Desarrollar un programa para la simulación de un cajero automático.
Se cuenta con la siguiente información:
- Llegan clientes a la puerta del cajero cada 2 ó 3 minutos.
- Cada cliente tarda entre 2 y 4 minutos para ser atendido.

Obtener la siguiente información:
Cantidad de clientes que se atienden en 10 horas.
Cantidad de clientes que hay en cola después de 10 horas.
Hora de llegada del primer cliente que no es atendido luego de 10 horas (es decir la persona que está primera en la cola cuando se cumplen 10 horas)

Instituto Universitario de Tecnología para la Informática –IutepiInstituto Universitario de Tecnología para la Informática - Iutepi

UNIDAD 7 FORMULARIOS Y CONTROLES

7.1 Windows Forms

El espacio de nombres System.Windows.Forms contiene clases para crear aplicaciones basadas en ventanas que aprovechan las ventajas de las características avanzadas de la interfaz de usuario disponibles en el sistema operativo Microsoft Windows.

.:. Uso de Formularios .:.
Datos Generales
Datos Opcionales
Datos del Usuario
Nombre: Liliana Nataly
Apellido P: Arcila
Apellido M: Díaz
Sexo: Masculino
Femenino
Nacionalidad: Perú
Datos del Perfil
Fecha Nacimiento:
23/02/1993
Desea Mostrar estos datos

7.2 Controles

GroupBox: Componente que muestra un marco alrededor de un grupo de controles con un título opcional.

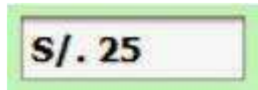
Button: Componente que permite implementar un Botón de pulsación.

Label: Proporciona información en tiempo de ejecución o texto descriptivo para un control

Instituto Universitario de Tecnología para la Informática –IutepiInstituto Universitario de Tecnología para la Informática - Iutepi

Sueldo Parcial:

TextBox: Componente que permite ingresar una línea de texto y editarlo



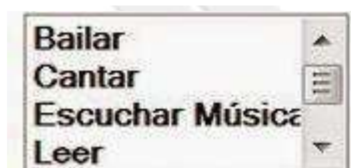
PictureBox: Componente permite mostrar una imagen en el formulario.

CheckBox: Permite seleccionar o deseleccionar uno o varios ítems de un conjunto de opciones.

RadioButton: Componente para mostrar varias opciones de las cuales sólo se puede seleccionar una.

ComboBox: Usado para implementar una lista desplegable de varias opciones de las cuales sólo se puede seleccionar una.

ListBox: Permite implementar una lista con varias opciones de las cuales se puede seleccionar una o varias.



MenuStrip: Permite implementar un menú de ítems y subítems de opciones en formularios Windows.

Instituto Universitario de Tecnología para la Informática –IutepiInstituto Universitario de Tecnología para la Informática - Iutepi

DataGridView: Muestra filas y columnas de datos en una cuadrícula que se puede personalizar.

1.3 Propiedades de los Controles

BackColor: Permite modificar el color de fondo de un componente.
ForeColor: Propiedad para modificar el color de la letra de un componente.
Font: Permite modificar la fuente, el estilo y el tamaño de la letra de un componente.
Visible: Propiedad que permite establecer si el control estará o no visible.
Text: Para definir la línea de texto que se va a visualizar en el componente.
Cursor: Define el estilo del cursor del Mouse.
ToolTipText: Permite ingresar un mensaje de ayuda que se visualizará cuando pasemos el mouse sobre el componente.
Enable: Permite habilitar o deshabilitar un componente.

1.4 Manejo de Eventos

Un evento es un suceso que ocurre como consecuencia de la interacción del usuario con la interfaz gráfica.

- Pulsación de un botón.
- Cambio del contenido en una caja de texto.
- Deslizamiento de una barra.
- Activación de un CheckBox.
- Movimiento de la ventana.
- Etc.

Videos interactivos de formularios en el siguiente enlace https://www.youtube.com/watch?v=O1n9g3Gu_Ro

EVALUACIÓN

Crear una aplicación:
Esta aplicación recreará el proceso de votación durante unas elecciones, pero con una variedad. La aplicación debería:
Incluir un formulario de registro para registrar un nuevo votante
Comprobar si el usuario cumple los requisitos para votar (mayor de 18 años, no tiene antecedentes penales, etc.)
Ofrecer de 5-10 opciones entre las que elegir
Mostrar los resultados de las elecciones

Instituto Universitario de Tecnología para la Informática –IutepiInstituto Universitario de Tecnología para la Informática - Iutepi

REFERENCIAS

https://www.incanatoit.com/2014/11/formularios-controles-programacion-csharp-net.html
https://docs.microsoft.com/es-es/dotnet/csharp/tour-of-csharp/arrays
https://www.tutorialesprogramacionya.com/csharpya/detalleconcepto.php?codigo=195&inicio=60

Instituto Universitario de Tecnología para la Informática –IutepiInstituto Universitario de Tecnología para la Informática - Iutepi