UD1-Aplicaciones de interfaces con diseño amigable

Aprenderás a

Diseñar interfaces gráficas, identificando y aplicando criterios de usabilidad mediante editores visuales, utilizando las funcionalidades del editor y adaptando el código generado.

Objetivos de aprendizaje

- Crear un interfaz gráfico, utilizando los asistentes de un editor visual.
- Utilizar las funciones del editor para ubicar los componentes del interfaz.
- Modificar las propiedades de los componentes para adecuarlas a las necesidades de la aplicación.
- Analizar el código generado por el editor visual.
- Modificar el código generado por el editor visual.
- Asociar a los eventos las acciones correspondientes.
- Desarrollar una aplicación que incluye el interfaz gráfico obtenido.
- Crear menús que se ajustan a los estándares.
- Crear menús contextuales cuya estructura y contenido siguen los estándares establecidos.
- Distribuir las acciones en menús, barras de herramientas, botones de comando, entre otros, siguiendo un criterio coherente.
- Distribuir adecuadamente los controles en la interfaz de usuario.
- Utilizar el tipo de control más apropiado en cada caso.
- Diseñar el aspecto de la interfaz de usuario (colores y fuentes, entre otros) atendiendo a su legibilidad.
- Verificar que los mensajes generados por la aplicación son adecuados en extensión y claridad.

Antes de empezar

¿Hola Mundo en Visual Studio?

Fundamentos y elementos de VB.NET

- Declaración de una variable
- Asignación de valores
- Conversión de tipos
- Comentarios
- Mostrar datos por pantalla
- Expresiones aritméticas
- Ejercicios

Declaración de variables

La declaración de una variable consiste en enunciar el nombre de la misma y asociarle un tipo. Para ello debemos utilizar la sentencia **`Dim`**

```
Boolean, Byte, Char,
Date,Decimal,Double, Integer, Long,
Object, Short, Single, String
```

```
Dim a As Double
```

Declaración de variables

- El compilador de VB no hace diferencia entre mayúsculas y minúsculas. **Ejemplo**: la variable dato1 y Dato1 es la misma variable

- VB permite **no** declarar una variable antes de ser usada. Por tanto, en cualquier punto de un programa podríamos escribir algo similar a “Mostrar velocidad01”, sin necesidad de haber declarado la variable velocidad01.

¿Qué nos mostrará si no hemos asignado contenido a la variable velocidad01?

Asignación de valores

Las variables locales son iniciadas por el compilador al valor por omisión correspondiente a su tipo

- Numéricas = 0
- Boolean = false
- String = Nothing
- Object = Nothing
- Date 01/01/0001 0:00:00

Asignación de valores

- Nosotros vamos a trabajar con Visual Basic **realizando declaración de variables** siempre. Para generar buenos programas y evitar errores y confusiones.

- El propio Visual Basic dispone de una instrucción que obliga a tener que declarar todas las variables, denominada **`Option Explicit On.`**

Option Explicit On

- La obligación de declaración de variables puede establecerse de dos maneras.

  a) En la ventana de código escribir Option Explicit On

  b) En el menú Herramientas, selecciona Opciones... y en la pestaña Editor (en otras versiones Herramientas, Opciones, Proyectos y Soluciones, Valores predeterminados de VB) establece (o comprueba que está establecida) la opción Requerir declaración de variables como activa ó Option Explicit On.

¿Qué ocurrirá si utilizamos en el código una variable que no ha sido declarada?

Visual Basic impedirá la ejecución del programa mostrando el mensaje de error: “Error de compilación. Variable no definida / no declarada” así como la línea de programa donde ha aparecido la variable no declarada.

Asignación de valores

Dim [Nombre variable] As [Tipo variable]

```
Dim d As Double = 3.14
Dim e As Integer = 10, r As Integer = 20
Dim opcion As Char = “a”
Dim cadena As String = “kkkkkkkkkkkkkkkkk”

Const mensaje As String = “INTENTELO DE
NUEVO ”
```

Conversión de tipos

System.Console.ReadLine :asignar un valor a una variable tecleada por el usuario.

El principal inconveniente de este método es que siempre devuelve un dato de tipo String. La conversión al tipo de datos elegido para la variable puede ser automática o si da error, debemos llevarla a cabo explícitamente.

Conversión de tipos

Error que se produce: No se puede convertir implícitamente el tipo “String” a “int”

Solución

```
a = Convert.ToInt32(System.Console.ReadLine());
```

System.Convert

ToBoolean	Convierte un valor especificado en un valor booleano equivalente.
ToByte	Convierte un valor especificado en un entero de 8 bits sin signo.
ToChar	Convierte un valor especificado en un carácter Unicode.
ToDateTime	Convierte un valor especificado en un tipo DateTime.
ToDecimal	Convierte un valor especificado en un número Decimal.
ToDouble	Convierte un valor especificado en un número de punto flotante de precisión doble.
ToInt16	Convierte un valor especificado en un entero de 16 bits con signo.
ToInt32	Convierte un valor especificado en un entero de 32 bits con signo.
ToInt64	Convierte un valor especificado en un entero de 64 bits con signo.
ToSByte	Convierte un valor especificado en un entero de 8 bits con signo.
ToSingle	Convierte un valor especificado en un número de punto flotante de precisión simple.
ToString	Convierte el valor especificado en la representación de tipo String equivalente.
ToUInt16	Convierte un valor especificado en un entero de 16 bits sin signo.
ToUInt32	Convierte un valor especificado en un entero de 32 bits sin signo.
ToUInt64	Convierte un valor especificado en un entero de 64 bits sin signo.

Ejemplo Conversión de tipos

```
Dim x As Single = 123.45
Dim y As Integer
y = Convert.ToInt32(x)
x = 123
```

Comentarios

Un comentario es un mensaje a cualquiera que lea el código fuente. Un comentario comienza con una comilla simple o por la palabra reservada REM y se extiende hasta el final de la línea.

Ejemplo:

```
Rem Este es un comentario
Rem Este es otro comentario
formula = 10 ’También es un comentario cualquier texto detrás de’

formula = 20 : Rem Este tambiés un comentario
```

Un comentario que comienza con *Rem* es un enunciado separado y debe comenzar en una línea nueva o debe estar separado del enunciado anterior por dos puntos.

Mostrar datos por pantalla

System.Console.WriteLine, vinculado con la salida estándar, normalmente la pantalla, que permite visualizar datos numéricos de cualquier tipo, y cadenas de caracteres.

```
System.Console.WriteLine(“El valor de la
variable a es igual a ”& a)
```

La expresión a mostrar se escribe entre los paréntesis del método y puede estar formada por elementos numéricos, cadenas de caracteres y constantes de caracteres. Para unir estos elementos y formar la expresión a mostrar se utiliza el operador **&**.

Mostrar datos por pantalla

También se pueden incluir en el primer parámetro de este método subcadenas de la forma {i}. Con ello se consigue que se muestre por la ventana de consola la cadena que se le pasa como primer parámetro pero sustituyéndole las subcadenas {i} por el valor convertido en cadena de texto del parámetro que ocupe la posición i+2 en la llamada a WriteLine.

```
System.Console.WriteLine(“Tengo{0}meses y{1}
años”,meses,años)
```

Expresiones aritméticas

Una expresión es un conjunto de operandos unidos mediante operadores para especificar una operación determinada.
Todas las expresiones cuando se evaluan retornan un valor.

+	Suma. Los operandos pueden ser enteros o reales	+=
-	Resta. Los operandos pueden ser enteros o reales	-=
*	Multiplicación. Los operandos pueden ser enteros o reales	*=
/	División real. Los operandos pueden ser enteros o reales. El resultado es de tipo Double en todos los casos.	/=

\	División entera. Los operandos deben ser enteros (Byte, Short, Integer o Long). Si alguno de los operandos es de tipo real, será convertido a entero. El resultado es entero en todos los casos.	\=
^	Exponencial. Los operandos pueden ser enteros o reales.	^=
Mod	Módulo o resto de una división. Los operandos pueden ser enteros o reales. Si ambos operandos son enteros el resto será entero; en otro caso, el resto será real.	

2.7 Ejercicios

1. Escribe un programa que reciba una cantidad en euros y la convierta a dolares. Para llevar a cabo el cambio también tiene que pedir el tipo de cambio.

2. Escribe un programa que lea los valores de tres resistencias electromacnéticas conectadas en paralelo y muestre en pantalla el valor global de las tres. El valor se calcula aplicando la siguiente formula: 1/(1/r1 + 1/r2 + 1/r3).

3. Escribe un programa que pida una hora en segundos y la saque por pantalla en formato hh:mm:ss.

Nota: Para realizar los ejercicios creamos aplicaciones **tipo consola** y lo codificamos todo dentro del main, es decir,todavía no utilizamos clases.

Expresiones condicional

```
If condicion Then instruccion [Else instruccion]

If [(]expresion condicional[)] Then
  ..................
[Else
.................]
End If

If [(]expresion condicional[)] Then
    If [(]expresion condicional[)] Then
    ..................
    [Else .................]
    End If
[Else .................] End If
```

Estructura condicional anidada

```
If [()]condicion1[()] Then Sentencias
[ElseIf [()]condicion2[()] Then sentencias
[ElseIf [()]condcion3[()] Then sentencias
[Else
Sentencia n
End If
```

Select

```
Select [Case] [(]expresion-test[)]
 Case expresion1
  ..............
  [Case expresion2] [................]
  [Case expresion3] [................]
  [Case Else] [..........]
End Select
```

Operadores lógicos

Operador	Operación
And o AndAlso	Da como resultado True si al evaluar cada uno de los operandos el resultado es true. Si uno de ellos es False, el resultado es False. Si se utiliza AndAlso en lugar de And y el primer operando es False, el segundo operando no es evaluado.
Or u OrElse	El resultado es False si al evaluar cada uno de los operandos el resultado es False. Si uno de ellos es True, el resultado es True. Si se utiliza OrElse en lugar de Or y el primer operando es True, el segundo operando no es evaluado.
Not	El resultado de aplicar este operador es False si al evaluar su operando el resultado es True, y True en caso contrario.
Xor	Da como resultado True si al evaluar cada uno de los operandos el resultado de uno es True y el de otro False; en otro caso el resultado es False.

```
Dim a, b, c, d, e, f, g As Boolean

a = 23 > 14 And 11 > 8
b = 14 > 23 And 11 > 8

c = 23 > 14 Or 8 > 11
d = 23 > 67 Or 8 > 11

e = 23 > 67 Xor 11 > 8
f = 23 > 14 Xor 11 > 8
g = 14 > 23 Xor 8 > 11
```

Operadores unitarios

Los operadores unitarios se aplican a un solo operando y son los siguientes: Not, + y -

+	Da como resultado el valor del operando. El operando debe de ser de tipo Byte, Short, Integer,Long, Single, Double o Decimal.
-	Cambia de signo al operando. El operando puede ser de un tipo entero o real.

```
Dim a As Integer, b As Integer, c As Integer
C = -a ’Si a = 2 c = ?
C = +b ’c = ?
```

```
Dim x, y As Boolean
x = Not 23 > 14
y = Not 23 > 67
```

Concatenación

Este operador permite generar una cadena de caracteres a partir de otras dos. La forma de utilizarlo es la siguiente:

Var = expresion1 & expresion2

La variable var tiene que ser de tipo String u Object y el tipo de expresion1 y expresion2 si no es de tipo String es convertido a String.

El operador & puede ser reemplazado por el operador + sólo cuando expresion1 y expresion2 sean de tipo String.

Ejemplos:

```
Dim s1,s2,s3 As String, n As Integer = 3 S2 = “Hola”
S3 = “Amigos”
S1 = n & s3 ’s1 = “3 amigos”
S1 = s2 + s3 ’s1 = “Hola amigos”
```

Manual Visual Basic

https://msdn.microsoft.com/es-es/library/2x7h1hfk.aspx

2.17 Ejercicios

1. Realizar un programa que lea una fecha representada por dos enteros, mes y año,y dé como resultado el número de días de ese mes. Hacer el ejercicio de formas distintas (con If, con ElseIf, con Select).

2. Realizar un programa que a través de un menú permita realizar las operaciones de sumar, restar, multiplicar, dividir y salir. Las operaciones constarán solamente de dos operandos.

¿Cuándo es un año bisiesto?

- Si es múltiplo de 4 y no es múltiplo de 100

```
año Mod 4 = 0 And año Mod 100 <> 0
```

- Los años múltiplos de 100 no son bisiestos, excepto si son múltiplos de 400

```
año Mod 400 = 0
```

While

La sentencia `While` ejecuta una o más sentencias cero o más veces, dependiendo del valor de una expresión booleana.

```
While [(]condicion[)]
  sentencias
  End While
```

Se evalúa la condición y si el resultado es False, el bloque de sentencias no se ejecuta y se pasa el control a la siguiente sentencia en el programa. Si el resultado de la evaluación es verdadero, se ejecuta el bloque de sentencias y el proceso descrito se repite.

```
Dim index As Integer = 0
While index <= 10
    Console.Write(index.ToString & " ")
     index += 1
End While

Console.WriteLine("")
' Output: 0 1 2 3 4 5 6 7 8 9 10
```

Do .. Loop While

Esta sentencia ejecuta una sentencia, simple o compuesta, una o más veces dependiendo del valor de una expresión. Su sintaxis es la siguiente:

```
Do
  Sentencias
Loop While [(]condicion[)]
```

Se ejecuta el bloque de sentencias,se evalúa la expresión correspondiente a la condición de finalización del bucle obteniéndose como resultado verdadero o falso. Si el resultado es falso, se pasa el control a la siguiente sentencia en el programa. Si el resultado es verdadero, el proceso descrito se repite.

```vb
Dim index As Integer = 0
Do
    Console.Write(index.ToString & " ")
    index += 1
Loop Until index > 10

Console.WriteLine("")
' Output: 0 1 2 3 4 5 6 7 8 9 10
```

For

La sentencia For permite ejecutar una sentencia simple o compuesta, repetidamente un número de veces conocido. Su sintaxis es la siguiente:

```
For variable = expresion1 To expresion2 [Step expresion3]
Sentencias
Next
```

Variable representa la variable de control que será iniciada con el valor de la expresión1, expresion2 representa el valor final que tomará la variable y expresion3 representa el valor positivo o negativo en el que se incrementa

La variable de control cada vez que se ejecuta el bloque de sentencias, lo cual ocurrirá mientras variable no alcance el valor de la expresion2. El valor de expresión3 es 1 por omisión.

```
For index As Integer = 1 To 5
    Console.Write(index.ToString & " ")
Next
Console.WriteLine("")
' Output: 1 2 3 4 5
```

```vb
For number As Double = 2 To 0 Step -0.25
    Console.Write(number.ToString & " ")
Next
Console.WriteLine("")
' Output: 2 1.75 1.5 1.25 1 0.75 0.5
0.25 0
```

2.19 Ejercicios

1. Realizar un programa que pida dos valores y calcule la suma de todos los números comprendidos entre ellos. (el primer número debe ser menor que el segundo).

2. Escribe un programa que pida al usuario un número concreto de valores enteros (el usuario nos dirá cuantos) y nos muestre en pantalla el resultado acumulado de multiplicar todos ellos.

Procedimientos

- Es un bloque de instrucciones incluido entre una instrucción de declaración **`Function, Sub`** y una declaración **`End`** correspondiente. Todas las instrucciones ejecutables de Visual Basic deben estar incluidas en algún procedimiento.

- Procedimiento = Función = Método

Sub

- Si el método **no** devuelve ningún valor utilizamos **`Sub`** para crear el método y **`End Sub`** para finalizarlo

```
Sub nombreMetodo (nombreparametro As tipoDeParametro)

…code…

End Sub
```

Argumentos

Los argumentos se pueden transferir de dos formas:

- **Por valor (Byval):**se transmite una copia del mismo al invocar al procedimiento, por lo que si el procedimiento varía el valor del argumento, la variable original no se ve afectada
- **Por referencia (ByRef),** se transmite una referencia a la variable original, por lo que si el procedimiento varía el valor, el valor de la variable original se ve afectado.

Function

- Si el método devuelve un valor utilizamos **`Function`** para crear el método y **`End Función`** para finalizarlo

```
Function nombreMetodo (parametro As
String)As Single

End Function
```

Para devolver un valor desde un procedimiento Function, se puede utilizar la instrucción **Return** o asignar el valor de devolución al nombre del procedimiento Function.

```
Function CalcularDescuento(ByVal importe As Double) As Double
    Const dto1 As Single = 0.1
    Const dto2 As Single = 0.15
    Dim descuentoAplicado As Single

        If importe >= 150 And importe < 250 Then
            descuentoAplicado = dto1
        ElseIf importe >= 250 Then
            descuentoAplicado = dto2
        End If

        'Return importe * descuentoAplicado
        CalcularDescuento = importe * descuentoAplicado
    End Function
End Module
```

2.21 Ejercicios

- Repetir los ejercicios del apartado 2.19 utilizando fuciones.

- En el primer ejercicio crear los procedimientos de EntradaDatos(proceso SIN retorno) Operacion (proceso con retorno) y VisualizarDatos.

- En el segundo EntradaDatos, Proceso sin retornar y VisualizarDatos.

Matrices

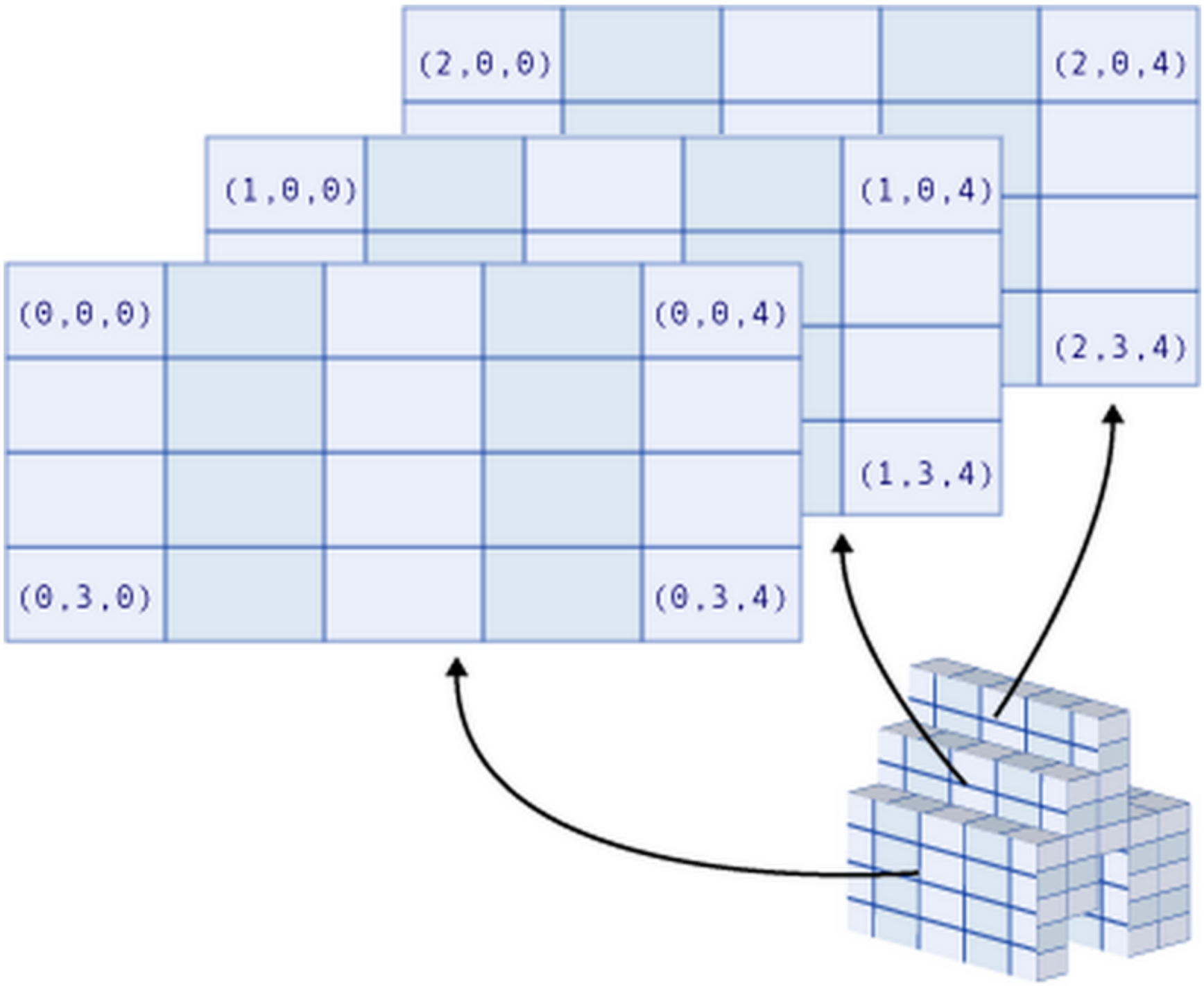
Matriz unidimensional

(0)	(1)	(2)	(3)	(4)

Matriz bidimensional

(0,0)				(0,4)
(1,0)				
(3,0)				(3,4)

Matriz tridimensional



¿Qué es una matriz?

Una matriz es un conjunto de celdas de memoria que se utilizan para almacenar múltiples variables. Todos los elementos de una matriz son del mismo tipo (por ejemplo, Integer o String) y para hacer referencia a ellos se utiliza un índice o subíndice, que es el orden en que se almacenaron en la matriz.

Todas las matrices derivan de System.Array

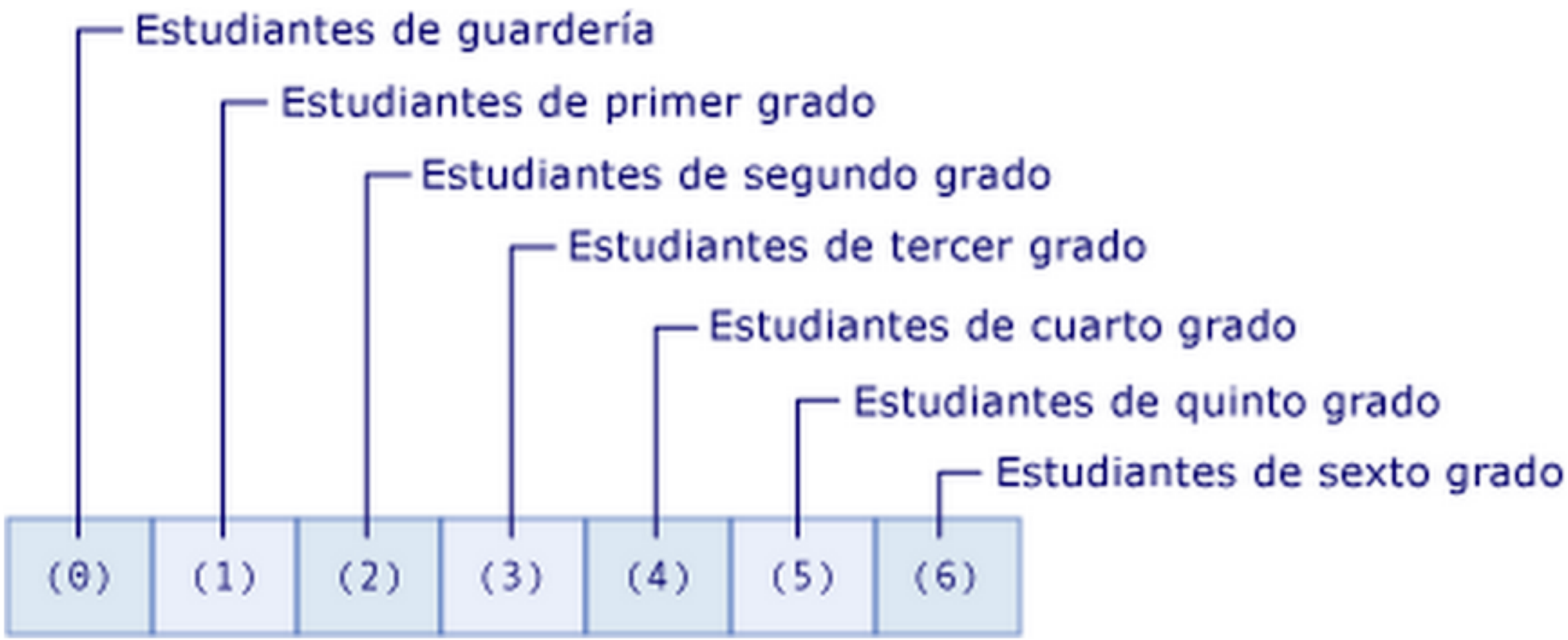
Declaración

Matriz unidimensional

```
Dim students(6) As Integer
```

La matriz students del ejemplo anterior contiene siete elementos. Los índices de los elementos van de 0 a 6. Tener esta matriz es más fácil que declarar siete variables diferentes.

Elementos de la matriz "students"



Elementos

```
Dim guarder As Integer = students(0)

Dim primero As Integer = students(1)

Dim sexto As Integer = students(6)
```

Crear una matriz

```
Dim cargoWeights As Double()

ReDim cargoWeights(15)
```

```
Dim cargoWeights(10) As Double

Dim atmospherePressures(2, 2, 4) As Integer
```

```
cargoWeights = New Double(10) {}

atmospherePressures = New Integer(2, 2, 4){}
```

Matrices dinámicas

A veces no sabremos el número de elementos que necesitaremos almacenar en una matriz. En una situación así se utilizan matrices dinámicas. El tamaño de una matriz puede variar durante la ejecución de un programa. Las matrices dinámicas se crean normalmente sin especificar su tamaño en su declaración, para ello se utilizará la instrucción Redim

Redim

- Si se tiene una matriz existente, se puede volver a definir su tamaño utilizando la instrucción Redim. Puede especificar que la instrucción Redim deba mantener los valores que se encuentren en la matriz o que cree una nueva matriz vacía. En el ejemplo siguiente se muestran usos diferentes de la instrucción Redim para modificar el tamaño de una matriz existente.

Ejemplo Redim

Asigna un nuevo tamaño a la matriz y conserva a los valores de los elementos actuales

```
ReDim Preserve cargoWeights(20)
```

Asignar un nuevo tamaño de la matriz y retener sólo los cinco primeros valores de los elementos

```
ReDim Preserve cargoWeights(4)
```

Asigna un nuevo tamaño a la matriz y elimina todos los valores de los elementos actuales.

```
ReDim cargoWeights(15)
```

Redim Perserve

- Descripción: modificador utilizado para conservar los datos de la matriz existente cuando se cambia el tamaño de **sólo la última dimensión**.

- Inicialización con Preserve. Si especifica Preserve, copias de Visual Basic los elementos de la matriz existente al nuevo array.

Redim Perserve

Preserve, cambia el tamaño de **sólo la última dimensión de la matriz**.

Si la matriz tiene sólo **una dimensión**, se puede cambiar el tamaño de esa dimensión y **conservar** todos los contenidos de la matriz, ya que va a cambiar la última y única dimensión.

Si la matriz tiene **dos o más dimensiones**, sólo se puede cambiar el tamaño de la última dimensión y conservar el contenido de ésta última

Ejemplo RedDim Perserve

```
Dim intArray(10, 10, 10) As Integer
ReDim Preserve intArray(10, 10, 20)
ReDim Preserve intArray(10, 10, 15)
ReDim intArray(10, 10, 10)
```

Asignación de valores

```
Dim valores As Double() = {1, 2, 3, 4,
5, 6}
Dim nombres() As String={“a”,“b”,“c”}
Dim B(,) As Short = New Short(,) { {5,
6}, {7, 8} }
```

Array

La clase Array consta de varios métodos para la manipulación de matrices. Obtén información detallada sobre ellos en la página 35 del pdf de Desarrollo de Interfaces

2.23 Ejercicios

1.Crear una matriz con el número de elementos y la dimensión indicada por el usuario(entre uno y tres).A continuación el usuario debe teclear valores para llenar la matriz y por último debemos visualizar los datos almacenados en la matriz.

2. Desarrolla un ejercicio en el que se cree un array de una dimensión del tamaño indicado por el usuario. Tras crearlo guarda en él los datos tecleados por el usuario. A continuación visualiza un menú para que el usuario pueda realizar distintas tareas: Buscar un valor en el array, Copiar un número determinado de elementos, inicializar el array, terminar la ejecución.

Matrices escalonadas

Matrices escalonadas

Una matriz de la que cada elemento es una matriz se llama una matriz de matrices o una matriz escalonada.

Ejemplo

En algunas ocasiones, la estructura de datos de la aplicación es bidimensional pero no rectangular. Por ejemplo, puede tener una matriz de meses, siendo cada elemento a su vez una matriz de días. Puesto que los distintos meses tienen un número distinto de días, los elementos no forman una matriz bidimensional rectangular. En este caso, puede utilizar una matriz escalonada en lugar de una matriz multidimensional.

```
Dim ventas()() As Double = New Double(11)() {}
Dim mes As Integer
Dim dias As Integer
For mes = 0 To 11
    dias = DateTime.DaysInMonth(Year(Now), mes
+ 1)
    ventas(mes) = New Double(dias - 1) {}
Next mes
```

Estructuras

```
Structure Empleado
    Dim nombre As String
    Dim apellido As String
    Dim telf As Long
    Dim sueldo As Decimal
End Structure
```

```
Dim empleado1 As Empleado
empleado1.nombre=“valor”
empleado1 = empleado2
```

Ejercicio

Crea un arrayList con los nombres de 10 alumnos de clase. Muéstralos ordenados alfabéticamente de forma ascendente. Muéstralos en forma descendente. Inserta en la primera posición una etiqueta con el código del curso y el nombre del tutor. Busca dentro del arrayList un nombre que te introducirán desde el teclado y le indicaras si está o no en la lista.

Ejercicio 2.25

Desarrollar un programa que permita guardar las notas de los alumnos. El programa debe ir pidiendo el **nombre del alumno**, las **asignaturas** de las que está matriculado y la **nota** obtenida en cada una de ellas. Hay que tener en cuenta que hay seis posibles asignaturas, pero no todos los alumnos están matriculados de todas.

Ejercicio 2.25

Cuando dispongamos de la información de entrada, debemos **visualizar las notas de cada alumno más su nota media**. Tras visualizar las notas de todos visualizaremos la **nota media de cada una de las asignaturas.**

Ayuda

Crear una tabla <asignaturas> e inicializarla con el nombre de 6 asignaturas

Crear una estructura <asignaturaNota> que contendrá el nombre asignatura y la nota

Crear la tabla para guardar los nombre de los alumnos <alumnos>, que introduciremos mas tarde

Crea una matriz escalonada <matriculaAlumnos> donde luego guardaremos los bloques de (asignatura y notas de la asignatura) en las que se haya matriculado cada uno de los alumnos

Crea el **procedimiento entradaDatos**

- repetir – tantas veces como alumnos se matriculen
  - pedir nombre alumno

    guardar en la tabla
  - visualiza (procedimiento) el nombre de las asignaturas
    - pedir número de asignaturas en las que se ha matriculado
    - crear huecos (columnas) en la M. escalonada matriculaAlumnos
  - repetir – tanta veces como asignaturas tengas ese alumno
    - pedir nombre asignatura y comprobar si el nombre es correcto (procedimiento) y guardar nombre asignatura en AsignaturaNotas en la posición asociada para ese alumno
    - pedir nota en esa asignatura y guardar en AsignaturaNotas misma posición que asignatura

Ficheros

Tipos de acceso

a) Acceso secuencial: orientado a su uso en archivos donde la información está dispuesta como tipo texto

b) Acceso aleatorio: orientado a su uso en archivos con datos contenidos en registros de longitud fija, que a su vez pueden estar subdivididos en campos. Un campo puede contener un valor numérico o ser tipo texto.

Operar con ficheros

1. Abrir
2. Operar
3. Cerrar

Stream

La comunicación entre un programa y el origen o el destino de cierta información se realiza mediante un **flujo de información** que no es más que un objeto que hace de intermediario entre el programa, y el origen o el destino de la información. De esta forma, el programa leerá o escribirá datos en el flujo sin importarle desde dónde viene la información o a dónde va. Este nivel de abstracción hace que un programa no tenga que saber nada del dispositivo, lo que se traduce en una facilidad más a la hora de escribir programas, ya que las operaciones para leer y escribir datos serán siempre más o menos las mismas.

Clases asociadas a las operaciones de gestión de ficheros

System.IO

Nos proporciona clases que nos van va permitir realizar operaciones para Leer y/o Escribir de un fichero, y utilizaremos una u otra dependiendo del tipo de información con la que vayamos a leer/escribir (byte, caracteres o números binarios)

`FileStream`: leer o escribir datos de un fichero byte a byte

`StreamReader/StreamWriter`: leer/escribir cadenas de caracteres

`BinaryReader/BinaryWriter`: leer/escribir datos en binario

Todas las clases a partir de la página 43

Clase	Descripción
BinaryReader	Lee tipos de datos primitivos como valores binarios en una codificación específica.
BinaryWriter	Escribe tipos primitivos en binario en una secuencia y admite escribir cadenas en una codificación específica.
BufferedStream	Agrega una capa de almacenamiento en búfer a las operaciones de lectura y escritura en otra secuencia. No se puede heredar esta clase.
Directory	Expone métodos estáticos para crear, mover y enumerar archivos en directorios y subdirectorios.
DirectoryInfo	Expone métodos de instancia para crear, mover y enumerar archivos en directorios y subdirectorios.
DirectoryNotFoundException	Excepción que se inicia cuando no encuentra parte de un archivo o directorio.
EndOfStreamException	Excepción que se inicia cuando se intenta realizar una operación de lectura más allá del final de una secuencia.
ErrorEventArgs	Proporciona datos para el evento Error.
File	Proporciona métodos estáticos para crear, copiar, eliminar, mover y abrir archivos y contribuye a la creación de objetos FileStream.
FileInfo	Proporciona métodos de instancia para crear, copiar, eliminar, mover y abrir archivos y contribuye a la creación de objetos FileStream.
FileLoadException	Excepción que se inicia cuando se encuentra un ensamblado administrado que no se puede cargar.
FileNotFoundException	Excepción que se inicia cuando se produce un error al intentar tener acceso a un archivo que no existe en el disco.
FileStream	Expone un objeto Stream alrededor de un archivo; se admiten operaciones de lectura y escritura sincrónica y asincrónica.

Flujos de bytes:

FileStream

FileStream

- Constructor
  - New FileStream(nombre As *String*, modo As *FileMode*, acceso As *FileAccess*)
- Modo: FileMode
    - CreateNew: crear nuevo,si el archivo ya existe, IOException.
    - Create: crear un archivo nuevo. Si el archivo ya existe, se sobrescribirá
    - Open: abre un que archivo exista
    - OpenOrCreate: si existe se abre, sino deberá crearse
    - Append: si existe se abre, si no lo crea, coloca el puntero al final sólo para añadir
- Acceso: FileAccess
    - Read: se lee
    - ReadWrite: se lee o escribe
    - Write: se escribe

Flujo E/S

Definir un flujo de entrada desde fichero xxx.txt

```
Dim fe As FileStream
fe = New FileStream(“c:\Desktop\doc.txt”, FileMode.Open,
FileAccess.Read)
```

Definir un flujo de salida al fichero xxxx.txt

```
Dim fs As FileStream
fs = New FileStream(“c:\Desktop\doc.txt”, FileMode.Create,
FileAccess.Write)
```

Lectura/Escritura

Lectura

```
nbytes = fe.Read(cbuffer, 0, cbuffer.Length)
```

cbuffer: matriz donde dejará los byte leídos

0: posición desde donde se empieza a dejar

dbuffer.Length: número de bytes a leer

nbytes: nro de bytes leídos ó 0 si no hay datos

Escritura

```
fs.Write(buffer,0,nbytes)
```

buffer: matriz que contiene los byte

0: posición desde donde se empieza a coger

nbytes: nro de bytes a coger para escribir en el fichero

Cerrar el flujo

fs.Close()

fe.Close()

Close puede lanzar la excepción IOException

Ejercicios

- P610 lee del teclado el contenido del fichero y escríbelo en fichero
- P611 Lee los datos del fichero y muéstralos en pantalla
- P612 Lee del teclado el contenido del fichero y escríbelo en fichero. Lee los datos del fichero y muéstralos en pantalla
- Tenéis un ejemplo de leer una línea de texto desde teclado y guardarla en un fichero en la página 50

Flujos de caracteres:

StreamReader
StreamWriter

`StreamReader/StreamWriter`

Nos permite leer carácter a carácter en un fichero respectivamente.

Constructores

1. `New StreamReader(nomfichero As String)`
   referencia al nombre del fichero

2. `New StreamReader(flujo As FileStream)`
   referencia a un flujo FileStream

Flujo de entrada

1er constructor

```
Dim sr As StreamReader  'definir un flujo de entrada
sr = New StreamReader("c:\temp\nombreFichero.txt") -
nombreFichero: fichero de donde se cogerán los datos
```

Lectura:

`sr.ReadLine( )` permite leer una línea. Una línea está definida como un conjunto de caracteres que termina en CR, LF, o bien CR+LF. La cadena de caracteres devuelta no contiene los caracteres de terminación.

```
cadena = sr.ReadLine()
```

cadena: es una variable de tipo String que guardará los caracteres leídos del flujo

Flujo de salida

1er constructor

```
Dim sw As StreamWriter

sw = New StreamWriter("c:\temp\nombreFichero.txt")
```

Si el fichero existe, se machacaría la información con los datos nuevos. Si se quiere conservar tendremos que utilizar el segundo constructor.

2er constructor

```
Dim fs As FileStream
Dim sw As StreamWriter
fs = New FileStream("c:\temp\doc.txt", FileMode.Append,
FileAccess.Write)

sw = New StreamWriter(fs)
```

Escritura

`sw.WriteLine(cadena)`

cadena: es una variable de tipo String que contiene los caracteres a grabar en el flujo

`sw.Write:` permite escribir cualquier tipo primitivo de datos como una cadena de caracteres

`sw.WriteLine:` Hace lo mismo que Write y añade CR+LF

Ejercicios

- P6E20 lee datos del teclado y los graba en un fichero. Pide el nombre del fichero por consola.

- P6E21 lee datos del teclado y los añade al final del fichero

- P6E22 lee datos del fichero y los muestra en pantalla

- P6E23 lee datos del teclado y se graban en el fichero a través del flujo FileStream y lee del fichero y muestra en pantalla

Flujos de datos binarios:

BinaryReader
BinaryWriter

`BinaryReader y BinaryWriter` son las clases de los flujos que nos va a permitir leer y escribir datos de cualquier tipo en formato binario y cadenas de caracteres en formato UTF-8. **`BinaryReader`** sólo puede leer datos almacenados en un fichero a través de un flujo **`BinaryWriter`** en un fichero.

El constructor para crear los flujos de salida y entrada son:

`Sub New BinaryReader(flujo As Stream)` 'referencia a un flujo FileStream

Lectura/ Escritura

Definir un flujo de entrada

```
Dim fs As FileStream
Dim br As BinaryReader  'definir un flujo de entrada
fs = New FileStream("c:\temp\doc.dat",
FileMode.Open, FileAccess.Read)
br = New BinaryReader(fs)  -  constructor
```

Definir un flujo de salida

```
Dim fs As FileStream
Dim bw As BinaryWriter
fs = New FileStream("c:\temp\doc.dat",
FileMode.Append, FileAccess.Write)
bw = New BinaryWriter(fs)  -  constructor
```

Métodos Lectura

br.ReadByte: Devuelve un valor de tipo **Byte**
br.ReadBytes: Devuelve una matriz unidimensional de **Bytes()** (matriz de bytes)
br.ReadChar: Devuelve un valor de tipo **Char**
br.ReadChars: Devuelve una matriz unidimensional de **caracteres**
br.ReadInt16: Devuelve un valor de tipo **Short**
br.ReadInt32: Devuelve un valor de tipo **Integer**
br.ReadInt64: Devuelve un valor de tipo **Long**
br.ReadDecimal: Devuelve un valor de tipo **Decimal**
br.ReadSingle: Devuelve un valor de tipo **Single**
br.ReadDouble: Devuelve un valor de tipo **Double**
br.ReadString: Devuelve una cadena de caracteres de formato UTF-8; el primer o los dos primeros bytes especifican el numero de bytes de datos que serán leídos a continuación.

Nota: los datos serán recuperados del fichero en el mismo orden y con el mismo formato con el que fueron escritos, de lo contrario los resultados serán inesperados.

EOF

Al leer si se alcanza final de fichero, lanzará la excepción **EndOfStreamException**, me obligará a poner un bloque Try .. Catch

```
Try
   ' Crear un flujo desde el fichero texto.txt
   fs = New FileStream(nombreFichero, FileMode.Open, FileAccess.Read)
   br = New BinaryReader(fs)

   'Leer los datos del fichero.
   'Cuando alcance el final del fichero el método utilizado para leer
   'lanzará una excepción del tipo EndOfStreamException
   While (True)
      'Leer mas datos
      nombre = br.ReadString()
      ...
      'Tto de datos

   End While

   Catch e As EndOfStreamException
      Console.WriteLine("Fin del fichero...")
   Catch e As IOException
      Console.WriteLine("Error imprevisto: " + e.Message)
   Catch e As Exception
      Console.WriteLine("Otro Error imprevisto: " + e.Message)
   Finally
      'Cerrar los flujos
      br.Close() : fs.Close()
```

```
'leo mientras que el puntero no llegue a la posición final del
fichero
While (fs.Length > fs.Position And nombreleido <> nombre)
   'leo el rgtro completo
   nombreleido = br.ReadString() 'lee nombre
   tfno = br.ReadString()        'lee tfno
   ...
   'TTo de datos
End While

if (nombreleido = nombre) Then
   'Mostrar o TTo de datos

Else
   ' cierro los flujos abiertos
   br.Close()
   fs.Close()
   Console.WriteLine("El nombre tecleado no se encuentra en la
   agenda")
End If
```

Escritura

bw.Write(Byte): Escribe un valor de tipo **Byte**

bw.Write(Byte()): Escribe una matriz unidimensional de **Bytes**

bw.Write(Char): Escribe un valor de tipo **Char**

bw.Write(Char()): Escribe una matriz unidimensional de **caracteres**

bw.Write(Short): Escribe un valor de tipo **Short**

bw.Write(Integer): Escribe un valor de tipo **Integer**

bw.Write(Long): Escribe un valor de tipo **Long**

bw.Write(Decimal): Escribe un valor de tipo **Decimal**

bw.Write(Single): Escribe un valor de tipo **Single**

bw.Write(Double): Escribe un valor de tipo **Double**

bw.Write(String): Escribe una cadena de caracteres de formato UTF-8; el primer o los dos primeros bytes especifican el numero de bytes de datos escritos a continuación.

Ejercicios

U2P630 lee del teclado (nombre, apellido, edad) y graba a través del flujo FileStream

U2P631 lee del fichero (nombre, apellido, edad) y muestra en pantalla. Para detectar el EOF, lo haremos utilizando: EndOfStreamException o mediante las propiedades del flujo Length y Position.

U2P632 lee del fichero (nombre, apellido, edad) y muestra en pantalla. Utilizamos procedimiento para lectura LeerString(br) y LeerInt16(br) y detectaremos el fin de fichero sin utilizar EndOfStreamException, ni propiedades del flujo

U2P633 lee del teclado (nombre, apellido, edad) y graba en fichero, lee del fichero y muestra en pantalla.

Ejercicio 2.26.2

Desarrolla un programa que cree un fichero, con el nombre indicado por el usuario, que guarde los datos de un conjunto de personas (nombre, dirección, teléfono y dirección de correo). Posteriormente lee los datos de este fichero y visualízalos en pantalla.

Hazlo de las tres formas vistas

Acceso Aleatorio

Acceso Aleatorio

Para que el acceso sea mas rápido, deberán de tener todos los registros la misma longitud, sino nos obligaría a tener un área donde guardara el tamaño de cada registro.

Las propiedades **Length** y **Position**, combinados con el método **Seek** nos van a facilitar el acceso aleatorio a un fichero

Length: propiedad del flujo que me da la longitud total del fichero en bytes

Position: propiedad del flujo me da la posición en bytes de donde se encuentra el puntero

Seek: mueve el puntero de L/E a una nueva localización desplazada *desp* bytes de la posición *pos* del fichero. Los desplazamientos pueden ser positivos o negativos

```
Seek (desp As Long, pos As SeekOrigin) As Long
```

`SeekOrigin` define las constantes:

- `Begin`: Referencia la primera posición en el fichero
- `Current`: Referencia la primera posición actual
- `End`: Referencia a la última posición en el fichero

Seek()

```
'    referencia a la primera posición del fichero
br.BaseStream.Seek (desplazamiento, SeekOrigin.Begin)
'    referencia a la primera posición actual
br.BaseStream.Seek (desplazamiento, SeekOrigin.Current)
'    referencia a la última posición del fichero
br.BaseStream.Seek (desplazamiento, SeekOrigin.End)
```

Desplazamiento, puede ser + o -, a partir del punto elegido avanza o retrocede

br no soporta el método Seek , es por lo que recurrimos a través de la propiedad BaseStream al método Seek del flujo fs

EOF

Tendré que controlar si al posicionarme rebaso la longitud del fichero

fe.Seek(): establece el puntero de L/E en el flujo
fe.Length: longitud total del fichero en bytes
fe.Position: posición en bytes de donde se encuentra el puntero, donde empezará la siguiente operación de Lectura o escritura en el fichero

```
If (fs.Length = fs.Position)
   Console.WriteLine(“Fin de Fichero..”)
End If

If (fe.Length <= fe.Position) Then
   Console.WriteLine("Fin de Fichero..")
End If
```

Leer/Escribir

```
Dim fs As FileStream
fs = New FileStream("c:\temp\doc.dat", FileMode.OpenOrCrearte,
FileAccess.ReadWrite)

Dim br As BinaryReader
br = New BinaryReader(fs)  ' para leer
br.BaseStream.Seek (desplazamiento, SeekOrigin.Begin)
```

```
Dim bw As BinaryWriter
bw = New BinaryWriter(fs)  ' para escribir
bw.BaseStream.Seek (desplazamiento, SeekOrigin.Begin)
```

Tratamiento General

' Calculo donde colocar el puntero

```
Dim desp As Integer = ( nroRgtro – 1) * longRgtro
```

' posiciono el puntero desplazado tantos bytes a partir del principio

```
br.BaseStream.Seek(desp, SeekOrigin.Begin)
```

' dependiendo del tipo de flujo abierto

```
br.ReadString(xxxxx) - leo
bw.Write(xxxxxx)     - grabo
```

Lectura aleatoria

```
' Crear un flujo al fichero de entrada (binario) a través de un
flujo FileStream
Dim fs As FileStream=
NewFileStream(nombreFichero,FileMode.Open,FileAccess.ReadWrite)
Dim br As BinaryReader = New BinaryReader(fs)

Dim longitudRegistro As Long = 100  'longitud del registro
Dim codigoBusqueda As Integre 'valor que proporciona el usuario
'coloco el puntero a partir el inicio del fichero
br.BaseStream.Seek((codigoBusqueda - 1) * longitudRegistro,
SeekOrigin.Begin)
'<Length: longitud del fichero > - < Position: donde se
encuentra el puntero >
If (fs.Length <= fs.Position) Then
   Console.WriteLine("Se sobrepaso el final de Fichero..")
Else
   'leo el rgtro completo
   nombre = br.ReadString()
   altura = br.ReadSingle()
   ...
   'Tto de los datos leído del rgtro
   Console.WriteLine("nombre: {0}", nombre)
   Console.WriteLine("Nota Ingles: {0:f2}", altura)
End If
```

Escritura aleatoria

```
Dim tamañoReg As Integer= 100    'longitud en byte del registro
Dim nroRgtro As Integer      'valor que te proporcionará el usuario

'antes de escribir coloco el puntero a partir el inicio del fichero
br.BaseStream.Seek(( nroRgtro - 1) * tamañoReg, SeekOrigin.Begin)

'así me aseguro que cada rgtro comienza en múltiplo de 100
bw.Write(obj.obtenerNombre())
bw.Write(obj.obtenerDireccion())
```

Ejercicios

2.26.5 Desarrolla un programa que nos permita ir grabando en un fichero los datos de un conjunto de personas. De cada persona se quiere guardar un código, el nombre, la dirección, el teléfono y la edad. Los códigos de las personas son consecutivos(1, 2, 3, 4, 5, etc...).