FUNDAMENTOS DE INFORMÁTICA

Tema 4
Programación estructurada: Funciones y procedimientos

Departamento de Ingeniería de Sistemas y Automática
Universidad de Vigo

UNIVERSIDADE DE VIGO

Programación estructurada

- La división de un programa en *unidades funcionales* más pequeñas presenta, entre otras, las ventajas siguientes:
  - **Modularidad**: Cada unidad tiene una funcionalidad concreta con un tamaño y complejidad manejable. Puede ser individualmente depurada, utilizada muchas veces en un mismo programa o en otros programas.
  - **Menor tiempo de desarrollo**: Al ser reutilizada, se reduce el tamaño de los programas y se minimizan los errores.
  - **Independencia de datos y encapsulación de información**: Una función es capaz de mantener una gran independencia con el resto del programa. Tiene sus propios datos aislados del resto y define la interfaz o comunicación con el resto del programa.

Programación estructurada

- Utilizaremos los siguientes tipos de módulos:
  - **Módulos de código**: los utilizamos para escribir las instrucciones y sentencias de nuestro programa. No tienen interfaz gráfica. Tienen la extensión ".bas".
  - **Módulos de formulario**: los utilizamos para escribir las instrucciones y sentencias asociadas a un formulario. Tienen interfaz gráfica. Tienen la extensión ".frm".
- Utilizaremos los siguientes tipos de unidades funcionales:
  - **Funciones** `Function` que devuelven un valor.
  - **Procedimientos** `Sub` que no devuelven ningún valor.

Funciones

- La sintaxis relativa a la **declaración de funciones** definidas por el usuario en Visual Basic es la siguiente:

```
Function nombre ([parámetros])[As tipo]
   [sentencias]
   [nombre = expresion]
   [Exit Function]
   [sentencias]
   [nombre = expresion]
End Function
```

- donde: ***nombre*** es el nombre que identifica la función; ***parámetros*** son los argumentos que son pasados cuando se llama a la función; ***tipo*** es el tipo de datos que devuelve la función (**Integer**, **String**, etc.).

UNIVERSIDADE DE VIGO

Funciones

- El ***nombre*** **de la función** actúa como una *variable* dentro del cuerpo de la función. El valor de *expresión* que se le asigne es almacenado en el propio nombre de la función. Si no se efectúa esta asignación, se devuelve 0 en valores numéricos y cadena vacía (“”) en cadenas.
- **`Exit Function`** permite abandonar la función antes de que ésta finalice normalmente y devolver así el control del programa a la sentencia inmediatamente a continuación de la que efectuó la llamada a la función.
- La sentencia **`End Function`** marca el final del código de la función y, al igual que la anterior, devuelve el control del mismo modo. Es la forma normal de finalizar una función.

Funciones

- La **llamada a una función** puede hacerse de diversas formas, pero la más usual es la siguiente:

```
variable = nombre ([argumentos])
```

- donde *argumentos* es una lista de **constantes**, **variables** o **expresiones**, separadas por ",", que son pasadas a la función como parámetros para su ejecución y `variable` recibe el dato retornado.
- El número de *argumentos pasados* debe ser igual al número de *parámetros declarados* en la función.
- Los **tipos** de los argumentos deben coincidir con los de los parámetros.

Ejemplo de Funciones

- El siguiente ejemplo corresponde a una función que devuelve como resultado la raíz cuadrada de un número:

```
Function RaizCuadrada (numero As Double) As Double
   If numero < 0 Then
     Exit Function
   End If
   RaizCuadrada = Sqr(numero)
End Function
```

- La llamada a esta función, se hace de la forma siguiente:

```
Dim resultado As Double, numero As Double

numero = InputBox("Introduce número para calcular la raíz cuadrada")
resultado = RaizCuadrada(numero)
MsgBox resultado
```

Procedimientos

- La sintaxis relativa a la **declaración de un procedimiento** definido por el usuario en Visual Basic es la siguiente:

```
Sub nombre ([parámetros])
           [sentencias]
           [Exit Sub]
           [sentencias]
End Sub
```

- donde: ***nombre*** es el nombre que identifica al procedimiento y *parámetros* son los argumentos que son pasados cuando se llama al procedimiento.

Procedimientos

- A diferencia de una **función**, un **procedimiento** *no* puede ser utilizado en una *expresión* pues no devuelve ningún valor en si mismo.
- **`Exit Sub`** permite abandonar el procedimiento antes de que finalice normalmente y devolver así el control del programa a la sentencia inmediatamente a continuación de la que efectuó la llamada al mismo.
- La sentencia **`End Sub`** marca el final de su código y, al igual que la anterior, devuelve el control del mismo modo. Es la forma normal de finalizar un procedimiento

UNIVERSIDADE DE VIGO

Procedimientos

- Llamaremos a un procedimiento del siguiente modo:

```
Call nombre ([argumentos])
```

- donde *argumentos* es una lista de **constantes**, **variables** o **expresiones**, separadas por comas (“,”), que son pasados a la función como parámetros para llevar a cabo su ejecución.
- El número de *argumentos pasados* debe ser igual al número de *parámetros declarados* en la función.
- Los **tipos** de los argumentos tienen que coincidir con los de los parámetros.

Ejemplo de procedimientos

- Procedimiento:

```
Sub muestralaraizcuadrada(numero As Double)
If numero < 0 Then
    MsgBox ("No calculo raíz cuadrada. " & numero & " es negativo.")
Else
    MsgBox ("La raíz cuadrada de " & numero & " es " & Sqr(numero))
End If
End Sub
```

- Llamada al procedimiento:

```
Dim numero As Double

numero = InputBox("Introduce número para calcular la raíz cuadrada")
Call muestralaraizcuadrada(numero)
```

Pase de argumentos por valor y referencia

- Pasar un *argumento* **por referencia** a una función o procedimiento implica que, en realidad, se le pasa la *variable* original, de modo que la función o procedimiento pueden modificar su valor.
- Pasar un argumento **por valor** implica crear una nueva variable dentro de la función o procedimiento y pasarle el *valor* de la *variable* externa. Si se modifica el valor de la variable copia, la variable original queda *inalterada*.
- Cuando en la llamada a una función o procedimiento se ponen como *argumentos* **constantes numéricas** o **expresiones**, estos *argumentos* se pasan siempre **por valor**.
- En Visual Basic, *por defecto*, los argumentos se pasan por referencia y, por lo tanto, es posible devolver *valores*.

UNIVERSIDADE DE VIGO

Ejemplo de Procedimientos

- El siguiente ejemplo corresponde a un procedimiento que devuelve en un parámetro la raíz cuadrada de un número:

```
Sub RaizCuadrada (ByVal numero As Double, resultado As Double)
   If numero < 0 Then
     resultado = -1
   End If
   resultado = Sqr(numero)
End Sub
```

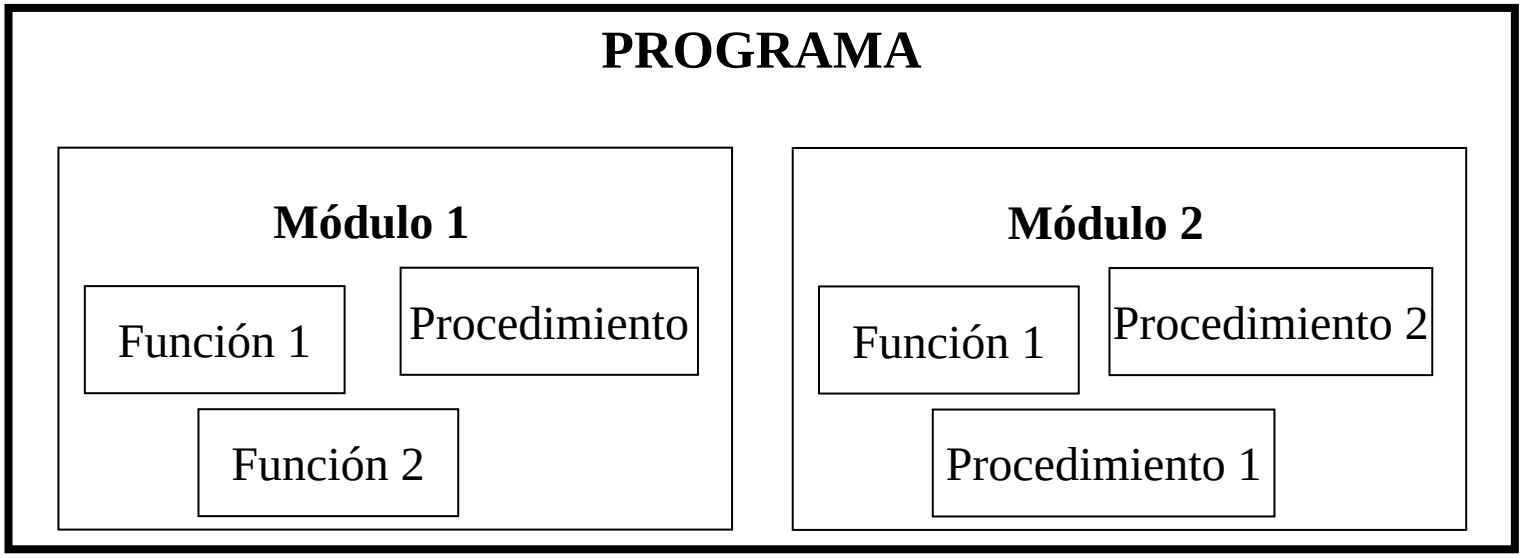
- La llamada a este procedimiento se hace de la forma siguiente:

```
Dim resultado As Double, numero As Double

numero = InputBox("Introduce número para calcular la raiz cuadrada")
Call RaizCuadrada(numero, resultado)
MsgBox resultado
```

Ámbito de las constantes y variables en Visual Basic

- Un programa en Visual Basic se puede componer de varios módulos.
- Los módulos a su vez suelen contener funciones y/o procedimientos:



Ámbito de las constantes y variables en Visual Basic

- El **ámbito** de una variable es la zona del programa informático donde ésta **se puede utilizar**.
- Cuando se utiliza una variable, Visual Basic "buscará" la declaración más cercana de esa variable. Puede haber dos variables diferentes, con el mismo nombre, en ámbitos distintos.
- La tabla siguiente resume cómo debe declararse una variable en función de la zona del programa donde se quiere utilizar:

ÁMBITO	DECLARACIÓN
Local	**`Dim`** (dentro de una **función** o **procedimiento**)
Módulo	**`Dim`** (dentro de la sección de **Declaraciones** de un módulo)
Global	**`Public`** (dentro de la sección de **Declaraciones** de un módulo)

Procedimientos intrínsecos de Visual Basic

Función Len

Devuelve el número de caracteres contenido en una variable de caracteres.

```
nombre_variable = Len(expresion_de_cadena)
```

Función Left

Devuelve los *n* caracteres de la *expresión_de_cadena* situados más a la izquierda.

```
nombre_variable = Left(expresion_de_cadena,n)
```

Procedimientos intrínsecos de Visual Basic

Función Right

Devuelve los *n* caracteres de la *expresión_de_cadena* situados más a la derecha.

```
nombre_variable = Right(expresion_de_cadena,n)
```

Función Mid

Devuelve una subcadena de una cadena de caracteres.

```
nombre_variable = Mid(expresion_de_cadena,n[,m])
```

Procedimientos intrínsecos de Visual Basic

Función InStr

Devuelve la posición del primer carácter de una subcadena en una cadena.

```
pos = InStr([n,]cadena,buscada)
```

Función String

Devuelve una cadena de caracteres igual a un caracter dado.

```
cadena = String(n,caracter)
```

Procedimientos intrínsecos de Visual Basic

Función Str

Convierte una expresión numérica en una expresión de caracteres.

```
nombre_variable = Str(número)
```

Función Val

Devuelve el valor numérico de una cadena de caracteres.

```
número = Val(expresion_de_cadena)
```

Procedimientos intrínsecos de Visual Basic

Función Chr

Devuelve el carácter ANSI correspondiente al código de carácter especificado.

```
nombre_variable = Chr(código_carácter)
```

Función Asc

Devuelve el código de carácter correspondiente al primer carácter de la cadena especificada.

```
código_carácter = Asc(expresion_de_cadena)
```

Procedimientos intrínsecos de Visual Basic

Función Now

Devuelve la fecha-hora actuales de acuerdo a la configuración de la fecha y la hora del sistema.

```
nombre_variable = now
```

Esta función devuelve un valor de tipo `Date` que se corresponde con una cadena de caracteres de la forma:

"dd/mm/aa HH:MM:SS"(día/mes/año hora:minuto:segundo)

Para visualizar esta fecha y hora según diversos patrones, utilizar la **función Format** con los símbolos especiales **d**, **m**, **y**, **h**, **m** y **s**. (*consultar la ayuda de esta función en la MSDN*).

Procedimientos intrínsecos de Visual Basic

- **Función Date**

  Devuelve la fecha actual del sistema en una cadena de caracteres.

  ```
  cadena = Date
  ```

- **Instrucción Date**

  Permite establecer la fecha actual del sistema.

  ```
  Date = fecha
  ```

- **Función Time**

  Devuelve la hora actual del sistema en una cadena de caracteres.

  ```
  cadena = Time
  ```

- **Instrucción Time**

  Permite establecer la hora actual del sistema.

  ```
  Time = hora
  ```

Procedimientos intrínsecos de Visual Basic

- **Función LCase**

  Convierte una cadena de caracteres a letras minúsculas.

```
cadena = LCase(expresión_de_cadena)
```

- **Función UCase**

  Convierte una cadena de caracteres a letras mayúsculas.

```
cadena = Ucase(expresión_de_cadena)
```

- **Funciónes LTrim**

  Devuelve `expresión_de_cadena` sin espacios en blanco a la izquierda

```
cadena = LTrim(expresión_de_cadena)
```

- **Funciónes RTrim**

  Devuelve `expresión_de_cadena` sin espacios en blanco a la derecha

```
cadena = RTrim(expresión_de_cadena)
```

- **Función Trim**

  Devuelve `expresión_de_cadena` sin espacios en blanco a la derecha ni a la izquierda

```
cadena = Trim(expresión_de_cadena)
```

Procedimientos intrínsecos de Visual Basic

- Funciones **Sin**, **Cos**, **Tan**, **Atn** dan como resultado los valores del *seno*, *coseno*, *tangente* y *arco tangente*, respectivamente.

```
valor = Sin(ángulo)
valor = Cos(ángulo)
valor = Tan(ángulo)
valor = Atn(ángulo)
```

- **Log** da como resultado el *logaritmo neperiano*.

```
variable = Log(expresión)
```

- **Exp** da como resultado el valor del número ***e*** elevado a la *expresión*.

```
variable = Exp(expresión)
```

Procedimientos intrínsecos de Visual Basic. Funciones de conversión

- **CBool**(expresión) conversión a un valor de tipo **Boolean**
- **CByte**(expresión) conversión a un valor de tipo **Byte**
- **CCur**(expresión) conversión a un valor de tipo **Currency**
- **CDate**(expresión) conversión a un valor de tipo **Date**
- **CDbl**(expresión) conversión a un valor de tipo **Double**
- **CDec**(expresión) conversión a un valor de tipo **Decimal**
- **CInt**(expresión) conversión a un valor de tipo **Integer**
- **CLng**(expresión) conversión a un valor de tipo **Long**
- **CSng**(expresión) conversión a un valor de tipo **Single**
- **CStr**(expresión) conversión a un valor de tipo **String**
- **CVar**(expresión) conversión a un valor de tipo **Variant**

Procedimientos intrínsecos de Visual Basic

Función Fix

Devuelve el número entero resultante de *truncar* el valor de la *expresión numérica*.

```
variable = Fix(expresión_numérica)
```

Función Int

Devuelve el mayor número entero que sea menor o igual que el valor de `expresión_numérica`.

```
variable = Int(expresión_numérica)
```

La diferencia entre **Int** y **Fix** es que si el número es negativo, **Int** devuelve el primer entero negativo menor o igual a número, mientras que **Fix** devuelve el primer entero negativo mayor o igual a número. Por ejemplo, **Int** convierte -8.4 en -9 y **Fix** convierte -8.4 a -8.

UNIVERSIDADE DE VIGO

Procedimientos intrínsecos de Visual Basic

- **Función Abs**

  Da como resultado el *valor absoluto* de la *expresión*.

  ```
  variable = Abs(expresión_numérica)
  ```

- **Función Sgn**

  Da como resultado un entero (1, -1 ó 0) indicando el *signo* del valor de una *expresión numérica*.

  ```
  variable = Sgn(expresión_numérica)
  ```

- **Función Sqr**

  Da como resultado la raíz cuadrada de una expresión numérica.

  ```
  variable = Sqr(expresión_numérica)
  ```

- **Función Round**

  Devuelve un número redondeado en el número especificado de lugares decimales.

  ```
  variable = Round(expresión[,numlugaresdecimales])
  ```

Procedimientos intrínsecos de Visual Basic

Función Rnd

Devuelve un número al azar, de tipo **`Single`**, mayor o igual que 0 y menor que 1.

```
variable = Rnd[(expresión)]
```

Si:

expresión < 0	devuelve siempre el mismo número.
expresión > 0 (defecto)	devuelve el siguiente nº aleatorio.
expresión = 0	devuelve el último nº generado.

Sentencia Randomize

Activa el generador de números aleatorios a partir de un número determinado (llamado *semilla*).

```
Randomize[n]
```

donde *n* es una expresión entera que se utiliza para activar una secuencia aleatoria.

FUNDAMENTOS DE INFORMÁTICA

Tema 4
Programación estructurada: Funciones y procedimientos

Departamento de Ingeniería de Sistemas y Automática
Universidad de Vigo