CAPÍTULO 1

FUNDAMENTOS

Microsoft Visual Basic es un conjunto de herramientas que posibilitan el desarrollo de aplicaciones para Windows de una manera rápida y sencilla, basado en el lenguaje BASIC y en la programación orientada a objetos.

La palabra “Visual” hace referencia al método que se utiliza para crear la interfaz gráfica de usuario. En lugar de escribir numerosas líneas de código para implementar la interfaz, simplemente se utiliza el ratón para agregar objetos prefabricados en el lugar deseado dentro de la pantalla.

La palabra “Basic” hace referencia al lenguaje BASIC (Beginners All-Purpose Symbolic Instruction Code), un lenguaje utilizado por más programadores que ningún otro lenguaje en la historia de la informática. Visual Basic ha evolucionado a partir del lenguaje BASIC original y ahora contiene centenares de instrucciones, funciones y palabras clave, muchas de las cuales están directamente relacionadas con la interfaz gráfica de Windows.

El lenguaje de programación Visual Basic no es exclusivo de Microsoft Visual Basic. Este lenguaje es utilizado también por Microsoft Excel, Microsoft Access y muchas otras aplicaciones Windows. El lenguaje de programación Visual Basic

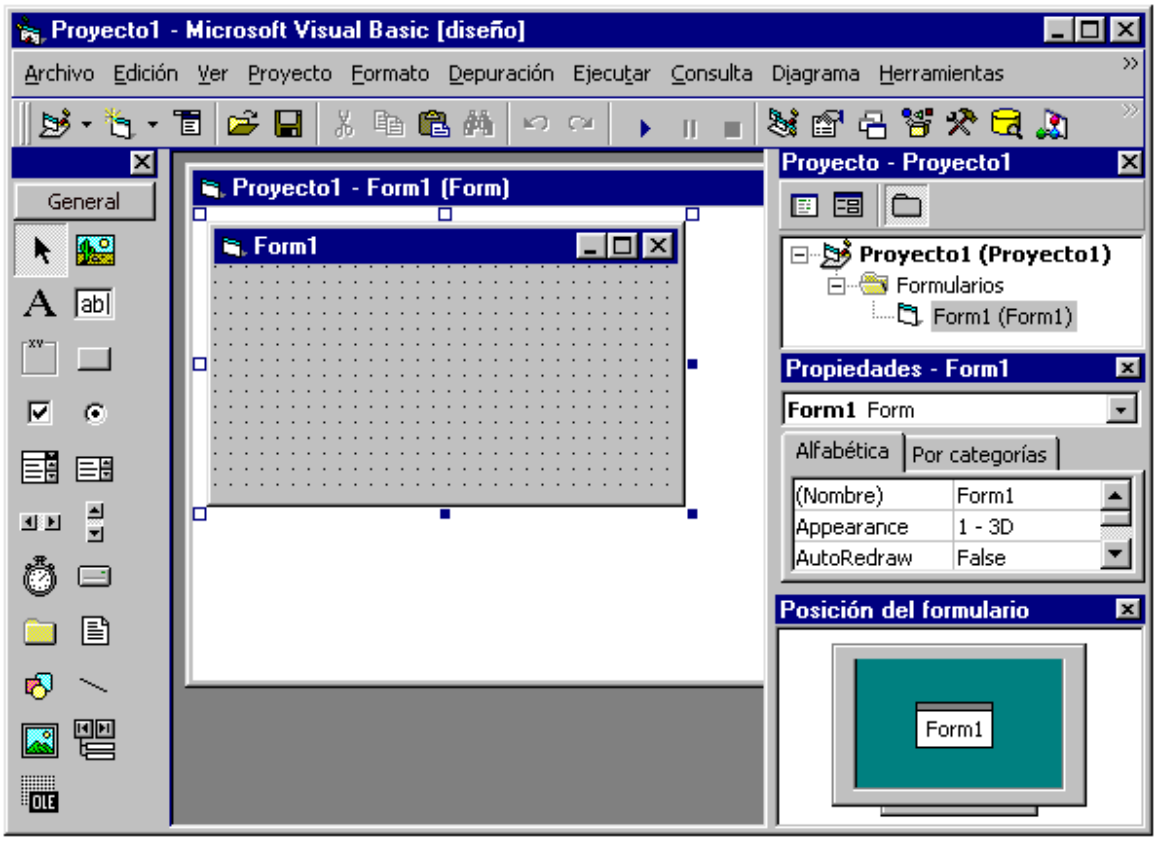
Script para programar en Internet es un subconjunto del lenguaje Visual Basic. De tal forma, que la inversión realizada en el aprendizaje de Visual Basic le ayudará a abarcar estas otras áreas.

EJECUTANDO VISUAL BASIC

Si ya tiene instalado Visual Basic, para ejecutarlo proceda igual que con cualquier otra aplicación Windows: hacer click en el icono correspondiente.

En seguida se presentará una ventana similar a la figura siguiente:

En la ventana anterior indicar el tipo de proyecto que desea crear (para nuestro caso seleccionar EXE estándar) y dar click sobre el botón “Abrir”. En ese instante se presentará el Entorno Integrado de Desarrollo (IDE) de Visual Basic:



Los elementos que componen el IDE de Visual Basic son:

- Barra de menús
- Barra de herramientas
- Cuadro de herramientas
- Diseñador de formularios
- Explorador de proyectos
- Ventana de propiedades
- Posición del formulario
- Menús contextuales

Barra de menús

Presenta las órdenes que se utilizan para desarrollar una aplicación. Las opciones más utilizadas son: Archivo, Edición, Ver, Ventana y Ayuda. Se proporcionan otros menús para tener acceso a funciones específicas de programación como Proyecto, Formato o Depuración.

Barra de herramientas

Facilita el acceso rápido a las órdenes más comúnmente utilizadas. Omitimos el significado de cada botón porque le será mostrado al pasar el puntero del ratón sobre cada uno de ellos.

Puede visualizar otras barras de herramientas ejecutando la opción Barra de herramientas del Menú Ver.

Cuadro de herramientas

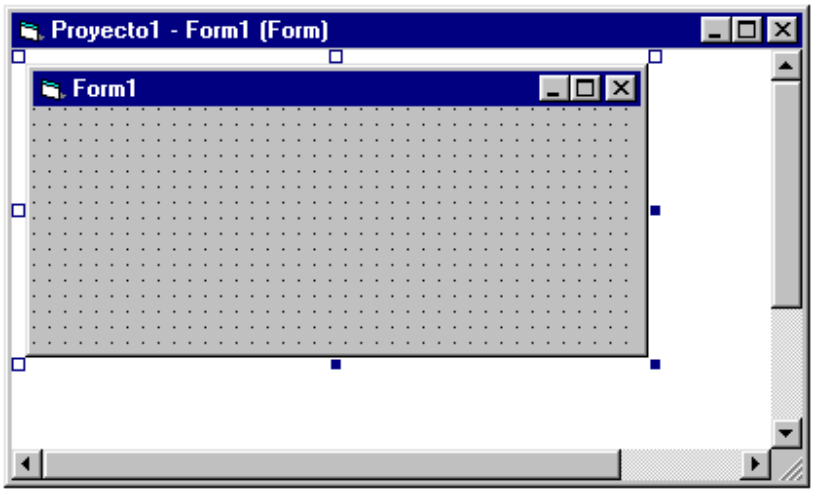
Proporciona un conjunto de herramientas que permiten diseñar la interfaz gráfica de usuario.

Icono	Descripción
	Puntero
	Cuadro de imagen
	Etiqueta
	Cuadro de texto
	Marco
	Botón de comando
	Casilla de verificación
	Botón de opción
	Cuadro combinado
	Cuadro de lista
	Barra de desplazamiento horizontal
	Barra de desplazamiento vertical
	Cronómetro
	Cuadro de lista de unidades
	Cuadro de lista de directorios
	Cuadro de lista de archivos
	Formas
	Líneas
	Imagen
	Contenedor OLE

Para utilizar algún objeto simplemente debe hacer doble click sobre el icono correspondiente, luego puede cambiar su posición y sus dimensiones.

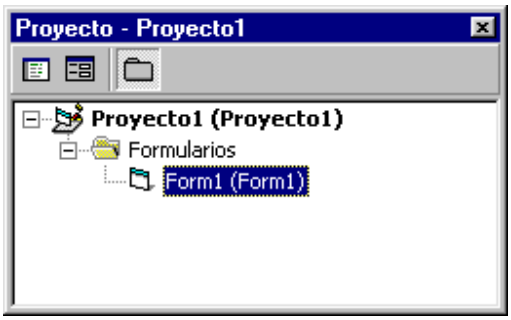
Diseñador de formularios

Es la ventana sobre la que colocaremos los objetos (controles) de la interfaz de usuario.



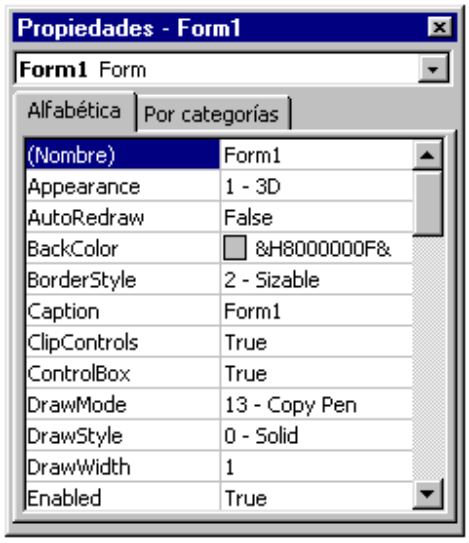
Explorador de proyectos

Contiene el conjunto de archivos que constituyen la aplicación o proyecto.



Ventana de propiedades

Como veremos más adelante, cada objeto lleva asociado un conjunto de propiedades. Para ver o especificar los valores de las propiedades de los objetos, utilizaremos la ventana de propiedades.

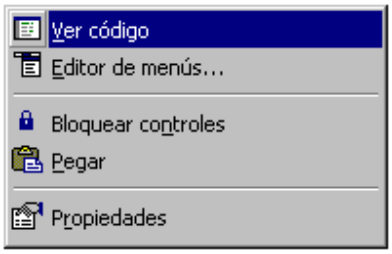


Posición del formulario

Esta ventana le permite especificar la posición de los formularios de la aplicación.

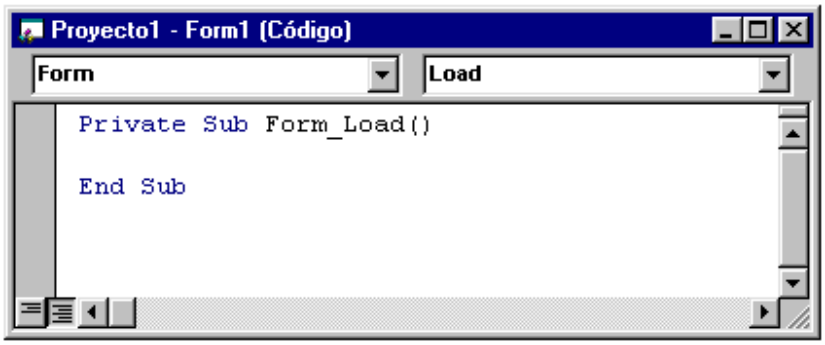
Menús contextuales

Es un menú emergente que presenta órdenes específicas relativas a un determinado objeto. Para abrir un menú contextual, dar click derecho sobre el objeto. Por ejemplo, al dar click derecho sobre el formulario se presenta el siguiente menú contextual.



Ventana de código

Se presenta cuando se hace doble click sobre un objeto que se encuentra en el Diseñador de formularios. Esta ventana muestra dos cuadros combinados (combos) en la parte superior, en una aparece el nombre del objeto (izquierda) y en la otra el nombre del evento (derecha).



MI PRIMERA APLICACIÓN

Hay tres pasos principales para crear una aplicación en Visual Basic:

- Crear la interfaz.
- Establecer propiedades.
- Escribir el código.

Para ver como se realiza esto, siga los pasos del siguiente ejemplo para crear una aplicación sencilla que consiste en un formulario y un botón de comando. Cuando haga click en el botón de comando aparecerá el mensaje “Visual Basic es fácil” en un cuadro de diálogo predefinido.

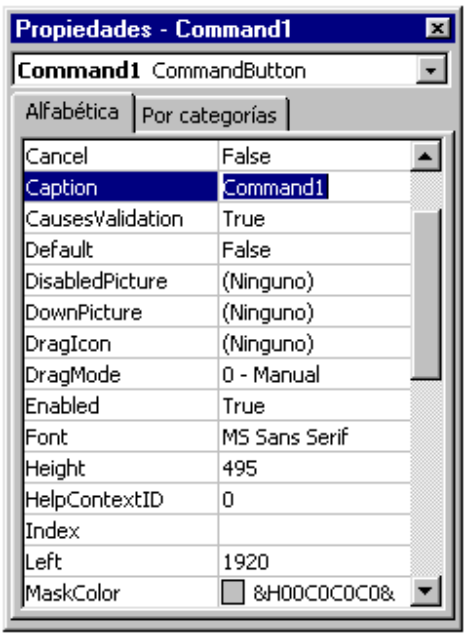
Creación de la interfaz

Para el desarrollo del presente ejemplo proceda a crear una nueva aplicación. En seguida añadir un botón de comando al formulario, para ello en el Cuadro de herramientas, dar doble click sobre el control deseado, tal como se indica en la figura siguiente:

La apariencia de la interfaz debe ser similar a la figura mostrada:

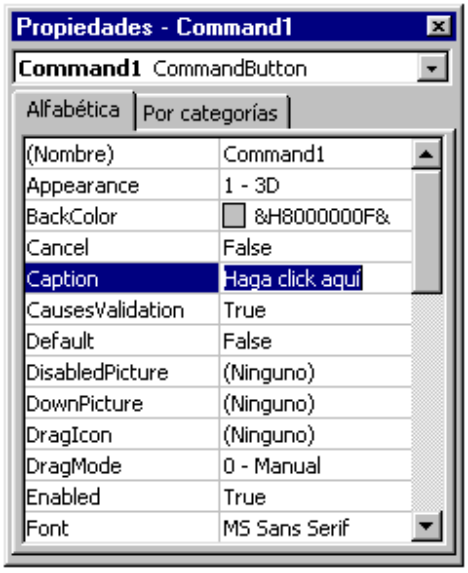
Estableciendo las propiedades

Para ver las propiedades de un objeto, simplemente debe seleccionarlo y pulsar la tecla F4. Por ejemplo la figura siguiente muestra las propiedades del botón de comando y podemos ver que la propiedad Caption tiene el valor “Command1”.



Puede cambiar el valor de cualquier propiedad seleccionándolo de la ventana anterior y simplemente

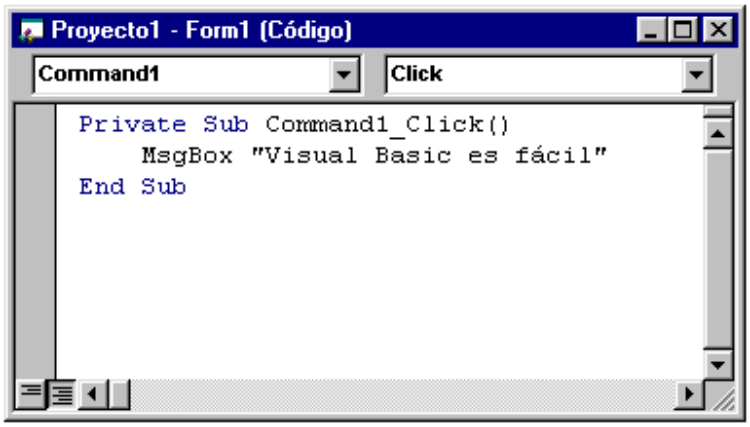
modificando su valor, por ejemplo para el caso anterior establezca el valor de la propiedad Caption de “Command1” a “Haga click aquí”.



De manera similar proceda cambiar la propiedad Caption del formulario a “Mi primera aplicación”. La apariencia de la interfaz debe ser similar a la figura mostrada:

Escribiendo el código

Para ingresar el código a la aplicación debe dar doble click sobre el control al cual asociaremos dicho código. Para nuestro caso dar doble click sobre el botón de comando e ingresar el código que se indica a continuación:



Para guardar la aplicación que acaba de crear, seleccione la opción Guardar Proyecto del Menú Archivo.

Bueno, ahora sólo falta que ejecute su aplicación. Para ello simplemente debe pulsar la tecla F5. El resultado debe ser similar al siguiente:

ºCAPÍTULO 2

© C. Castillo P.

LOS OBJETOS

Visual Basic se basa en la programación orientada a objetos (POO), la cual es una forma de programación que utiliza objetos (similares a los objetos del mundo real) para la solución de problemas. La POO permite descomponer un problema en bloques relacionados. Cada bloque pasa a ser un objeto autocontenido que contiene sus propios datos e instrucciones. De esta manera, la complejidad se reduce y se pueden realizar programas más largos de una manera sencilla.

MECANISMOS BÁSICOS DE LA POO

Los mecanismos básicos de la programación orientada a objetos son:

- Objetos
- Propiedades
- Métodos
- Eventos
- Mensajes
- Clases

Objetos

Un objeto es una entidad que tiene atributos particulares (propiedades) y unas formas de operar sobre ellos (métodos). Por tanto, un objeto contiene variables que especifican su estado y operaciones que definen su comportamiento.

Son ejemplos de objetos: formularios, botones de comando, cuadros de texto, etiquetas, etc.

Propiedades

Las propiedades representan las características del objeto. Hay propiedades particulares, como Caption que la poseen los botones de comando por ejemplo, y genéricas como Name que la poseen todos lo objetos.

Métodos

Los métodos son procedimientos asociados a un objeto. Se ejecutan como respuesta a un evento, por ejemplo al dar click en un botón de comando. También pueden ser invocados explícitamente en el programa.

Eventos

Un evento es la capacidad de un objeto de reaccionar cuando ocurre una determinada acción (acción y reacción). Como respuesta a un evento se envía un mensaje y se ejecuta un determinado método (procedimiento).

Mensajes

Un mensaje es una llamada a un método (procedimiento), de tal forma que cuando un objeto recibe un mensaje la respuesta a ese mensaje es ejecutar el procedimiento asociado.

Cuando se ejecuta un programa orientado a objetos, los objetos están constantemente recibiendo, interpretando y respondiendo a mensajes de otros objetos.

Clases

Una clase es una descripción para producir objetos de esa clase o tipo. Es decir se trata de una generalización de un tipo específico de objetos. En otras palabras, un objeto es una variable del tipo definido por una clase. Por ejemplo, piense en un molde para hacer pasteles, el molde es la clase y los pasteles los objetos.

CARACTERÍSTICAS DE LA POO

Las características fundamentales de la programación orientada a objetos son:

- Abstracción
- Encapsulamiento
- Herencia
- Polimorfismo

Abstracción

La abstracción permite no detenernos en los detalles concretos del funcionamiento de las cosas, sino centrarnos en los aspectos que realmente nos importan y nos son útiles en un determinado momento, en cierta medida, se podría decir que es “úsese el objeto y olvídese de como funciona en forma interna”.

Por ejemplo, para manejar una computadora no necesitamos saber como funcionan sus circuitos electrónicos, en términos de corriente, tensión, etc.

Encapsulamiento

Esta característica permite ver un objeto como una “caja negra” autocontenida en la que se ha metido de alguna manera toda la información que maneja dicho objeto. Esto permite manipular los objetos como unidades básicas, permaneciendo oculta su estructura interna.

Herencia

La herencia es la característica que permite compartir automáticamente propiedades y métodos entre objetos. Es decir, se pueden crear nuevas clases de objetos en base a clases existentes. Más concreto, un objeto puede heredar un conjunto general de propiedades y métodos a las que puede añadir aquellas características que son específicas suyas. El usuario de Visual Basic no dispone de esta característica.

Polimorfismo

Polimorfismo, del griego cuyo significado es “muchas formas”, es la característica que permite implementar múltiples formas de un mismo método, dependiendo cada una de ellas de la clase sobre la que se realiza la implementación. Esto hace posible que se puede acceder a una variedad de métodos distintos (todos con el mismo nombre) utilizando exactamente el mismo medio de acceso.

LOS OBJETOS DE VISUAL BASIC

Visual Basic soporta la abstracción, la encapsulación, el polimorfismo y la reutilización de código.

La reutilización de código es la capacidad de trasladar características de un objeto a otro, lo que se logra con alguna forma de herencia. Esto se consigue mediante la creación y uso del objeto.

Los objetos de Visual Basic están divididos en dos tipos: controles y contenedores.

Los controles son un medio gráfico que permiten a los usuarios interactuar con la aplicación para la manipulación de datos y ejecución de tareas. Son ejemplo de controles: etiquetas, cuadros de texto, botones de comando, casillas de verificación, botones de opción, cuadros de lista, cuadros combinados, etc.

Los contenedores son objetos que pueden incluir otros objetos y permiten el acceso a los objetos que contienen. Entre

los más utilizados tenemos a los formularios, marcos y cuadrículas.

El formulario más los controles constituyen la interfaz de la aplicación.

REFERENCIANDO OBJETOS

La sintaxis para referenciar objetos es la siguiente:

```
OBJETO.PROPIEDAD = VALOR
```

Por ejemplo, para establecer el título de un formulario a “Visual Basic es fácil”, la orden sería:

```
Form1.Caption = “Visual Basic es fácil”
```

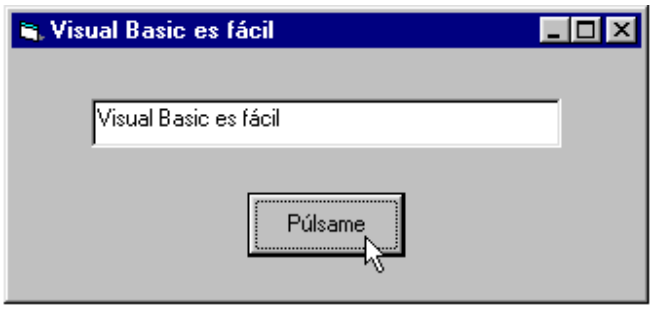
LOS EVENTOS

Cada objeto responde a un conjunto de eventos. Como respuesta a un evento se ejecuta un determinado procedimiento. Los procedimientos asociados a un evento presentan la forma:

```
Private Sub OBJETO_EVENTO()
     SENTENCIAS
End Sub
```

Para aclarar este concepto desarrollaremos la siguiente aplicación, la cual consiste en un formulario, un cuadro de texto y un botón de comando. El usuario debe ingresar un mensaje en el cuadro de texto y al pulsar el botón de comando,

el mensaje recientemente ingresado, se coloca como título del formulario.



Para ello proceda a construir la interfaz mostrada en la figura anterior. Luego, dar doble click sobre el botón de comando e ingrese el siguiente código:

```
Private Sub Command1_Click()
     Form1.Caption = Text1.Text
End Sub
```

Seguidamente guarde y proceda ejecutar su aplicación. Como puede verificar, el evento Click se dispara cada vez que el usuario pulsa (hace click) sobre un control.

CONVENCIÓN DE NOMBRES PARA LOS OBJETOS

La convención a seguir en la presente guía - para dar nombre a los objetos - consiste en utilizar ciertos prefijos, los cuales indicaran el tipo de objeto del que se trata.

Prefijo	Objeto

Frm	Formulario
Fra	Marco
Lbl	Etiqueta
Txt	Cuadro de texto
Cmd	Botón de comando
Chk	Casilla de verificación
Opt	Botón de opción
Lst	Cuadro de lista
Cbo	Cuadro combinado
Tim	Cronómetro
HS	Barra de desplazamiento horizontal
VS	Barra de desplazamiento vertical
Pic	Cuadro de imagen
Img	Imagen
Drv	Cuadro de lista de unidades
Dir	Cuadro de lista de directorios
Fil	Cuadro de lista de archivos
Ole	Contenedor OLE
Grd	Cuadrícula
Gra	Gráfico
Ctr	Control (se utiliza en procedimientos cuando el tipo de control es desconocido)

EL LENGUAJE

Visual Basic es un lenguaje de programación basado en el lenguaje BASIC, al cual incorpora la funcionalidad de la programación orientada a objetos.

El código de Visual Basic se almacena en módulos, donde cada módulo está subdividido en distintas secciones, una para cada objeto del módulo. Cada sección de código puede contener uno o más procedimientos, formados por declaraciones de constantes y variables, expresiones, sentencias de control y llamadas a procedimientos y/o funciones.

En este capítulo vamos a analizar los diferentes elementos que intervienen al momento de realizar el código para una aplicación.

TIPOS DE DATOS

Los datos con los que trabajaremos probablemente incluyan información relacionada con números, dinero, nombres, descripciones, fechas, etc. Cada dato corresponde a un determinado tipo, es decir, pertenece a una categoría de datos que se manipulan de maneras similares.

Tipo	Descripción	Rango

Integer	Entero (2 bytes)	-32768 a 32767
Long	Entero largo (4 bytes)	-2147483648 a 2147483647
Single	Punto flotante de simple precisión (4 bytes)	-3.40E+38 a 3.40E+38
Double	Punto flotante de doble precisión (8 bytes)	-1.79D+308 a 1.79D+308
Currency	Monetario (8 bytes)	+/- 922337203685477.5807
Byte	Carácter (1 byte)	0 a 255
String	Cadena de caracteres (1 byte por carácter)	Aproximadamente hasta 64K (65400 caracteres)
Boolean	Lógico (2 bytes)	True o False
Date	Fecha/Hora (8 bytes)	01/ENERO/100 a 31/DICIEMBRE/9999
Variant (por omisión)	Cualquier tipo de dato	Con números hasta el intervalo de un tipo Double. Con caracteres 22 bytes + 1 byte por carácter

IDENTIFICADORES

Los identificadores son nombres dados a los elementos de una aplicación, tales como constantes, variables, procedimientos, funciones, objetos, etc. Un identificador es una secuencia de caracteres que puede ser de hasta 255 caracteres. Para la construcción de identificadores debemos tener presente las siguientes reglas:

- Deben comenzar con una letra y no puede contener espacios en blanco.
- Letras, dígitos y caracteres subrayados están permitidos después del primer carácter.
- No se puede utilizar una palabra reservada como identificador. Una palabra reservada tiene un significado especial para Visual Basic.

CONSTANTES

Una constante almacena un dato cuyo valor no cambia durante la ejecución de un programa. Para declarar una constante utilice la siguiente sintaxis:

```
Const NOMBRE_CONSTANTE [As TIPO] = VALOR
```

Si no se declara el tipo de constante (utilizando As TIPO) se asigna a la constante el tipo de dato más apropiado a su valor.

El valor de una constante puede ser numérico, alfanumérico, carácter o de tipo fecha y hora. Por ejemplo:

```
Const MAXIT = 25
Const PI As Double = 3.141592
Const CADENA As String = “Visual Basic es fácil”
Const FECHA_POR_DEFECTO = #01/01/99#
```

VARIABLES

Una variable almacena un dato cuyo valor puede cambiar durante la ejecución de un programa. Para declarar una variable utilice la siguiente sintaxis:

```
Dim NOMBRE_VARIABLE [As TIPO]
```

Cuando se declara una variable y no se especifica su tipo (con As TIPO), se asume que es de tipo Variant.

La instrucción Dim puede realizar más de una declaración, teniendo en cuenta que la cláusula opcional As TIPO le permite definir el tipo de dato de cada variable que vaya a declarar. Por ejemplo:

```
Dim X, Y As Integer
```

La sentencia anterior le puede inducir a pensar que X e Y son de tipo Integer, lo cual no es cierto, pues X es de tipo Variant (por omisión) e Y es de tipo Integer.

Para asignar valores a una variable, utilizar la siguiente sintaxis:

NOMBRE_VARIABLE = VALOR

A manera de ejemplo considere lo siguiente:

```
Dim CANTIDAD As Integer, PRECIO As Double, TOTAL As Double
CANTIDAD = 30
PRECIO = 1.5
TOTAL = CANTIDAD * PRECIO
```

OPERADORES

Los operadores son símbolos que indican cómo serán manipulados los operandos. Los operandos son los datos (constantes y/o variables) sobre los que actúa los operadores para producir un determinado resultado.

Operadores aritméticos

Operador	Significado
^	Exponenciación
-	Menos unario
* /	Multiplicación y división
\	División entera
Mod	Resto de una división entera
+ -	Suma y resta

Operadores relacionales

Operador	Significado
=	Igual que
<>	Diferente que
<	Menor que
>	Mayor que
<=	Menor o igual que
=>	Mayor o igual que

Operadores lógicos

Operador	Significado
Not	Negación
And	Conjunción
Or	Disyunción inclusiva
Xor	Disyunción exclusiva
Eqv	Equivalencia (opuesto a Xor)
Imp	Implicación (falso si primer operando verdadero y segundo operando falso)

Operadores de cadenas de caracteres

Operador	Significado
&	Concatenación
Like	Compara dos cadenas de caracteres

SENTENCIAS

Una sentencia es una línea de código que indica una o más operaciones a realizar. Una línea puede incluir varias sentencias, separadas unas de otras por dos puntos. Por ejemplo:

```
Dim CANTIDAD As Integer, PRECIO As Double, TOTAL As Double
CANTIDAD = 30 : PRECIO = 1.5 : TOTAL = CANTIDAD * PRECIO
```

Una sentencia Visual Basic puede escribirse en más de una línea física utilizando el carácter de continuación de línea (un espacio en blanco seguido del carácter de subrayado). Por ejemplo:

```
Dim CH4_ENTRADA As Double, CH4_SALIDA As Double, _
    CH4_GENERA As Double, CH4_CONSUME AS Double
```

ENTRADA Y SALIDA DE DATOS

Visual Basic posee una serie de objetos (controles) que pueden ser utilizados como mecanismos de entrada y salida (E/S) de datos. Sin embargo, estos serán estudiados en capítulos posteriores.

Otra posibilidad de proporcionar datos o de visualizarlos durante la ejecución de una aplicación es utilizando cajas de diálogos predefinidas. Debido a que esta es la forma más simple de realizar la E/S de datos nos ocuparemos de ella.

Entrada de datos

La entrada de datos permite proporcionar valores durante la ejecución de una aplicación. Estos valores son ingresados generalmente mediante el teclado y asignados a variables del programa. En Visual Basic una forma de ingresar datos a la aplicación es utilizando la caja de diálogo predefinida provista por la función InputBox. Su sintaxis es la siguiente:

```
VARIABLE = InputBox(MENSAJE [, TÍTULO] [, PREDETERMINADO])
```

Donde MENSAJE es el mensaje que indica al usuario el tipo de información que debe ingresar. Por ejemplo, las sentencias:

```
Dim EDAD As Integer
EDAD = InputBox(“Ingrese su edad?”)
```

Da lugar a que Visual Basic presente la siguiente caja de diálogo solicitando la información requerida:

Como puede observar, el resto de los parámetros son opcionales. TÍTULO es el texto que se muestra en la barra de título del cuadro de diálogo, si se omite TÍTULO, el nombre de

la aplicación es la que se muestra en la barra de título. PREDETERMINADO es una expresión de cadena que aparece en el cuadro de texto como respuesta predeterminada si no se proporciona ningún otro texto. Si se omite PREDETERMINADO el cuadro de texto se muestra vacío. A manera de ejemplo considere lo siguiente:

```
Dim EDAD As Integer
EDAD = InputBox("Ingrese su edad?", "Ficha de datos", 18)
```

En este caso se visualiza la caja de diálogo que se presenta en la figura siguiente:

Salida de datos

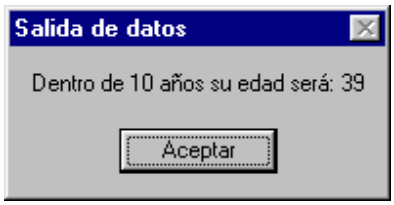
La salida de datos generalmente consiste en tomar la información de la memoria y mostrarla en pantalla. Para realizar la operación de salida de datos en Visual Basic, podemos utilizar el procedimiento MsgBox, cuya sintaxis es como sigue:

```
MsgBox MENSAJE [, ICONO, TÍTULO]
```

MENSAJE es La cadena que se muestra como salida en la caja de diálogo. ICONO es se refiere al estilo de icono que se va ha utilizar y TÍTULO es una texto que se mostrará en la barra de título de la caja de diálogo. Por ejemplo:

```
Dim EDAD As Integer
EDAD = InputBox(“Ingrese su edad?”)
EDAD = EDAD + 10
MsgBox “Dentro de 10 años su edad será:” & Str(EDAD)
```

Si ejecuta el código anterior e ingresa su edad (en mi caso 29 años) la salida debe ser similar a la figura siguiente:



Para que pueda ejecutar el código anterior, proceda a crear una nueva aplicación y haga doble click sobre el formulario, luego ingrese el código anterior:

```
Private Sub Form_Load()
    Dim EDAD As Integer
    EDAD = InputBox(“Ingrese su edad?”)
    EDAD = EDAD + 10
    MsgBox “Dentro de 10 años su edad será:” & Str(EDAD)
End Sub
```

A manera de observación cabe señalar que la función Str() se utiliza para convertir una expresión numérica a una cadena de caracteres.

ESTRUCTURAS DE CONTROL

Las estructuras de control permiten tomar decisiones y realizar un proceso repetidas veces (procesos iterativos). Para ello Visual Basic dispone de las siguientes estructuras de control: If . . . Then . . . Else, If . . . Then . . . ElseIf, Select Case, For . . . Next, While . . . . Wend, Do . . . Loop y GoTo.

Estructura If . . . Then . . . Else

Este tipo de estructura evalúa una determinada condición y en función a ello ejecuta uno de entre dos posibles grupos de sentencias. Su sintaxis es la siguiente:

```
If CONDICIÓN Then
        SENTENCIAS_VERDADERAS
Else
        SENTENCIAS_FALSAS
End If
```

Si CONDICIÓN es verdadera se ejecuta el grupo de SENTENCIAS_VERDADERAS. Si CONDICIÓN es falsa se ejecuta el grupo de SENTENCIAS_FALSAS.

Por ejemplo, el siguiente código determina si un entero A es o no divisible por otro entero B:

```
Dim A As Integer, B As Integer
A = InputBox(“Ingrese A?”)
B = InputBox(“Ingrese B?”)
If A Mod B = 0 Then
     MsgBox Str(A) & “ es divisible por ”  & Str(B)
Else
     MsgBox Str(A) & “ no es divisible por ” & Str(B)
End If
```

Estructura If . . . Then . . . ElseIf

Este tipo de estructura se utiliza para elegir una de entre múltiples alternativas. Su sintaxis es como sigue:

```
If CONDICIÓN_1 Then
          SENTENCIAS_1
ElseIf CONDICIÓN_2 Then
          SENTENCIAS_2
               .
          .
          .
Else
     SENTENCIAS_FALSAS
End If
```

Si CONDICIÓN_1 es verdadera se ejecuta el grupo de SENTENCIAS_1, y si es no se cumple, se evalúan secuencialmente las condiciones siguientes hasta Else, ejecutándose las sentencias correspondientes al primer ElseIf cuya condición sea verdadera. Si todas las condiciones son falsas, se ejecutan las SENTENCIAS_FALSAS correspondientes a Else.

Por ejemplo considere el siguiente código, el cual imprime el nombre del mes correspondiente a los números del 1 al 12.

```
Dim N As Integer
N = InputBox(“Ingrese N?”)
If N = 1 Then
     MsgBox “Enero”
ElseIf N = 2 Then  : MsgBox “Febrero”
ElseIf N = 3 Then  : MsgBox “Marzo”
ElseIf N = 4 Then  : MsgBox “Abril”
ElseIf N = 5 Then  : MsgBox “Mayo”
ElseIf N = 6 Then  : MsgBox “Junio”
ElseIf N = 7 Then  : MsgBox “Julio”
ElseIf N = 8 Then  : MsgBox “Agosto”
ElseIf N = 9 Then  : MsgBox “Setiembre”
ElseIf N = 10 Then : MsgBox “Octubre”
ElseIf N = 11 Then : MsgBox “Noviembre”
ElseIf N = 12 Then : MsgBox “Diciembre”
Else : MsgBox “Error de datos”
End If
```

Estructura Select Case

Este estructura es una alternativa a la estructura If . . . Then . . . ElseIf, cuando lo que necesita es comparar la misma expresión con diferentes valores. Su sintaxis es la siguiente:

```
Select Case EXPRESIÓN_TEST
        Case EXPRESIÓN_1
             SENTENCIAS_1
        Case EXPRESIÓN_2
             SENTENCIAS_2
                .
                .
                .
        Case Else
             SENTENCIAS_FALSAS
End Select
```

En este caso se comprueba el valor de EXPRESIÓN_TEST frente a la lista expresiones EXPRESIÓN_1, EXPRESIÓN_2, . . . y así sucesivamente, y busca el primer Case que incluya el valor evaluado en EXPRESIÓN_TEST, ejecutando a continuación el bloque de sentencias correspondiente. Si no existe un valor igual a EXPRESIÓN_TEST, entonces se ejecuta las SENTENCIAS_FALSAS correspondientes al Case Else.

A manera de ejemplo vamos a codificar el programa de la sección anterior el cual imprime el nombre del mes correspondiente a los números del 1 al 12. Pero en esta vez utilizaremos la estructura Select Case (compare con cual de ambas estructuras es más cómodo trabajar).

```
Dim N As Integer
N = InputBox(“Ingrese N?”)
Select Case N
     Case 1  : MsgBox “Enero”
     Case 2  : MsgBox “Febrero”
     Case 3  : MsgBox “Marzo”
     Case 4  : MsgBox “Abril”
     Case 5  : MsgBox “Mayo”
     Case 6  : MsgBox “Junio”
     Case 7  : MsgBox “Julio”
     Case 8  : MsgBox “Agosto”
     Case 9  : MsgBox “Setiembre”
     Case 10 : MsgBox “Octubre”
     Case 11 : MsgBox “Noviembre”
     Case 12 : MsgBox “Diciembre”
     Case Else
          MsgBox “Error de datos”
End Select
```

Cabe destacar que Select Case también se puede utilizar de la siguiente manera:

```
Dim X As Integer
X = InputBox(“Ingrese X?”)
Select Case X
     Case 1
          MsgBox “X = 1”
     Case 2, 3
          MsgBox “X = 2 o X = 3”
     Case 4 To 10
          MsgBox “4 <= X <= 10”
     Case Else
          MsgBox “X < 1 o X > 10”
End Select
```

Estructura For . . . Next

Esta estructura es utilizada para ejecutar un bucle un número determinado de veces. El número de iteraciones deberá ser conocido de antemano. Su sintaxis es la que se presenta a continuación:

```
For CONTADOR = INICIO To FINAL [Step INCREMENTO/DECREMENTO]
    SENTENCIAS_REPETITIVAS
    [Exit For]
Next
```

Donde la variable CONTADOR es inicializada con el valor de INICIO y se incrementa o decrementa hasta un valor FINAL. INCREMENTO/DECREMENTO define la manera en que cambia el valor de la variable CONTADOR en cada iteración.

La sentencia Exit For permite salir del bucle For ... Next antes de que este finalice.

Por ejemplo, el siguiente código imprime los cuadrados de los números enteros del 1 al 25.

```
Dim C As Integer
For C = 1 To 25
     Print C
Next
```

Para que este código funcione correctamente debe establecer la propiedad AutoRedraw del formulario a True.

Estructura While . . . Wend

La estructura While . . . Wend es aquella en la que el número de iteraciones no se conoce por anticipado y el cuerpo del bucle se repite mientras se cumple una determinada condición. Su sintaxis es la siguiente:

```
While CONDICIÓN
          SENTENCIAS_REPETITIVAS
Wend
```

La estructura While . . . Wend evalúa la CONDICIÓN en cada iteración y si el resultado es verdadero continúa su ejecución. El bucle termina cuando CONDICIÓN es falsa.

A manera de ejemplo, considere el siguiente código, el cual imprime en pantalla la suma de los N primeros números naturales, es decir S = 1 + 2 + 3 + 4 + . . . + N

```
Dim N As Integer, S As Integer
S = 0
N = InputBox(“Ingrese N?”)
While N <> 0
     S = S + N
     N = N - 1
Wend
MsgBox “La suma es S= ” & Str(S)
```

Estructura Do . . . Loop

Esta estructura ejecuta un bucle mientras una condición dada sea cierta, o hasta que una condición dada sea cierta. La condición puede ser verificada antes o después de ejecutarse el cuerpo del bucle. Su sintaxis es:

Formato 1:

```
Do [While/Until] CONDICIÓN
     SENTENCIAS_REPETITIVAS
     [Exit Do]
Loop
```

Formato 2:

```
Do
     SENTENCIAS_REPETITIVAS
     [Exit Do]
Loop [While/Until] CONDICIÓN
```

Esta estructura (como se puede observar en ambos formatos) permite realizar varias estructuras diferentes dependiendo si la condición de terminación está al principio o al final del cuerpo del bucle.

Como ejemplo, consideremos el siguiente programa, el cual calcula el factorial de un entero N ingresado por teclado:

```
Dim N As Integer, FACT As Integer, C As Integer
FACT = 1
C = 1
N = InputBox(“Ingrese N?”)
Do
     FACT = FACT * C
     C = C + 1
Loop While C <= N
MsgBox “El factorial de ” & Str(N) & “ es=” & Str(FACT)
```

Sentencia GoTo

Transfiere el control a una línea específica de código, identificada por una etiqueta o por un número de línea. Su sintaxis es:

GoTo ETIQUETA/NUMERO_LÍNEA

Aunque el uso del GoTo se ha depreciado considerablemente, en el presente trabajo no se juzgará su validez. Sin embargo, se ha establecido que no hay situaciones de programación que requieran el uso del GoTo, es decir, no es un elemento necesario para hacer completo el lenguaje.

Sin embargo, el GoTo es un convenio que, si su usa con cuidado, puede ser beneficioso en ciertas situaciones de programación.

Por ejemplo podría escribir un bucle para imprimir los cuadrados de los números enteros del 1 al 25.

```
Dim X As Integer
X = 0
LABEL1:
     X = X + 1
     Print X ^ 2
     If X < 25 Then GoTo LABEL1
```

Un uso abusivo de la sentencia GoTo da lugar a códigos difíciles de interpretar y de mantener. Por ello, se recomienda su uso solamente en ocasiones excepcionales. La tarea que vaya a desempeñar una sentencia GoTo puede suplirse utilizando cualquiera de las estructuras de control vistas en las secciones anteriores.

ARREGLOS

Un arreglo o matriz es una estructura de datos en la que se almacena una colección finita de datos del mismo tipo, que comparten un nombre común, a los que se puede acceder por la posición (índice) que ocupa cada uno de ellos dentro del arreglo. Cada elemento del arreglo es una variable que puede contener un número o una cadena de caracteres, dependiendo del tipo de arreglo que se declare.

Los arreglos en Visual Basic se clasifican en estáticos y dinámicos.

Arreglos estáticos

Son aquellos arreglos cuyo tamaño no puede cambiar en tiempo de ejecución. La declaración de un arreglo estático se puede realizar mediante la siguiente sintaxis:

```
Dim NOMBRE_ARREGLO(DIMENSONES) As TIPO
```

Donde DIMENSIONES es una lista de números, separados por comas y que definen las dimensiones del arreglo. Esta lista puede ser de la siguiente forma:

DIMENSIÓN_1, DIMENSIÓN_2, DIMENSIÓN_3, . . ., DIMENSIÓN_K

Para el caso de un arreglo formado por "K" dimensiones, es decir un arreglo K-dimensional.

A manera de ejemplo considere los siguientes casos que se pueden presentar:

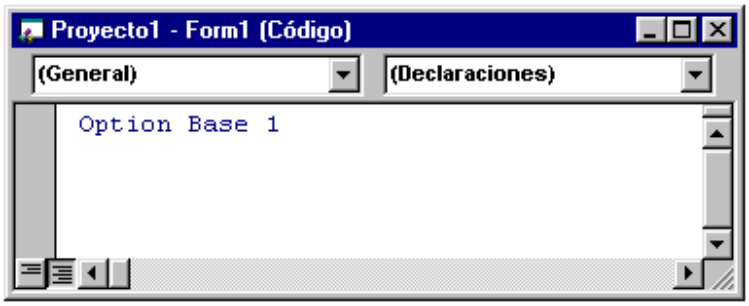
```
Dim LISTA(9) As Integer
Dim MATRIZ(4, 3) As Double
Dim NOMBRES(19) As String * 30
```

En el primer caso se declara un arreglo unidimensional de 10 elementos de tipo entero (por defecto los elementos de un arreglo se numeran a partir de 0).

En el segundo caso se define de un arreglo bidimensional de 20 elementos (5 filas por 4 columnas) de tipo punto flotante en doble precisión.

En el tercer caso se define una arreglo unidimensional de 20 elementos de tipo cadena de caracteres (cada elemento tiene una longitud fija de 30 caracteres).

Como se observa la numeración de los elementos de un arreglo por defecto comienza en cero, para hacer que la numeración comience en 1, debe ingresar la siguiente orden en la sección de Declaraciones del módulo de formulario:



Luego, para declarar los mismos arreglos de los casos anteriores, las sentencias serían las siguientes:

```
Dim LISTA(10) As Integer
Dim MATRIZ(5, 4) As Double
Dim NOMBRES(20) As String * 30
```

Establecer la sentencia Option Base a 1 o trabajar con el valor predeterminado (cero) queda a consideración del lector.

Arreglos dinámicos

Son aquellos arreglos en los que su tamaño puede definirse o modificarse en tiempo de ejecución. Para declara una arreglo dinámico utilice la siguiente sintaxis:

```
Dim NOMBRE_ARREGLO() As TIPO
```

Para definir el tamaño del arreglo utilizar la siguiente sintaxis:

```
ReDim NOMBRE_ARREGLO(NÚMERO_ELEMENTOS)
```

Cada vez que se ejecuta la sentencia ReDim, todos los valores almacenados en el arreglo se pierden. Para definir o cambiar el tamaño del arreglo conservando los valores del mismo use la siguiente sintaxis:

```
ReDim Preserve NOMBRE_ARREGLO(NÚMERO_ELEMENTOS)
```

Para liberar el espacio de memoria utilizado por arreglos dinámicos que ya no son útiles, utilizar la sintaxis:

Erase NOMBRE_ARREGLO

La orden Erase asigna cero a cada elemento de los arreglos numéricos y nulo (“”) a cada elemento de los arreglos de cadena de caracteres.

A manera de ejemplo considere el siguiente código que declara y hace uso de un arreglo dinámico:

```
Dim I As Integer, N As Integer, TEMP() As Integer
N = InputBox(“Ingrese N?”)
ReDim TEMP(N - 1)
For I = 0 To N - 1
     TEMP(I) = I + 1
     Print TEMP(I)
Next
Erase TEMP
```

REGISTROS

Un registro es un nuevo tipo de dato, que se define como una colección de datos de diferentes tipos, conocidos como “campos”, los cuales se encuentran evidentemente relacionados. Un registro sólo se puede ser creado en la sección de declaraciones de un módulo. La sintaxis correspondiente es:

```
Private Type NOMBRE_REGISTRO
        DECLARACIONES_DE_LOS_MIEMBROS
End Type
```

Por ejemplo considere el siguiente código que define la estructura ALUMNO:

```
Private Type ALUMNO
     ID_ALUMNO As String * 7
     NOMBRE As String * 25
     DIRECCION As String * 35
     TELEFONO As String * 7
     ESTADO As Boolean
End Type
```

Luego, podemos declarar una variable tipo ALUMNO de la siguiente forma:

```
Dim X As ALUMNO
```

Para referirse a un determinado miembro del registro se utiliza el operador selector de campos (.), veamos:

```
X.ID_ALUMNO = “980976G”
X.NOMBRE = “Carlos Castillo Peralta”
X.DIRECCION = “Jr. C. Richardson 412 Chorrillos”
X.TELEFONO = “2510850”
X.ESTADO = True
```

Las mismas órdenes del párrafo anterior se pueden escribir de una manera más fácil, más legible y más eficiente si utiliza la sentencia With . . . End With, del siguiente modo:

```
With X
     . ID_ALUMNO = “980976G”
     .NOMBRE = “Carlos Castillo Peralta”
     .DIRECCION = “Jr. C. Richardson 412 Chorrillos”
     .TELEFONO = “2510850”
     .ESTADO = True
End With
```

FUNCIONES

Las funciones son uno de los elementos básicos en programación. A continuación serán estudiadas detalladamente.

Declaración de una función

Una función es un procedimiento que cuando se ejecuta devuelve un único resultado al procedimiento que la invocó. La sintaxis correspondiente a la declaración de una función es:

```
Function NOMBRE_FUNCIÓN([LISTA_PARÁMETROS]) [As TIPO]
     SENTENCIAS
     [NOMBRE_FUNCIÓN = VALOR_RETORNADO]
     [Exit Function]
End Function
```

Donde LISTA_PARÁMETROS es una secuencia de variables separadas por comas que se corresponden con los argumentos pasados cuando es invocada la función.

Para especificar el tipo de datos que será retornado por la función utilice la cláusula opcional As TIPO, el tipo es Variant por omisión.

El valor retornado por la función es almacenado en su propio nombre, es decir en NOMBRE_FUNCIÓN, que actúa como variable dentro del cuerpo de la función. Si no se efectúa esta asignación el valor devuelto será cero si la función es de tipo numérica, nulo (“”) si la función es de tipo cadena, o vacío (Empty) si la función es de tipo Variant.

La cláusula opcional Exit Function permite salir de una función antes de que la función finalice, en caso sea esto necesario.

La sintaxis para la llamada a una función es de la siguiente forma:

```
VARIABLE = NOMBRE_FUNCIÓN([LISTA_ARGUMENTOS])
```

Donde LISTA_ARGUMENTOS es una secuencia de constantes, variables o expresiones separadas por comas. El número de argumentos debe ser igual al número de parámetros de la función. Los tipos de los argumentos deben coincidir con los tipos de sus correspondientes parámetros.

A manera de ejemplo considere la implementación de la siguiente función:

$$f(x) = x^2 + 2x + 3$$

En la sección de Declaraciones del módulo de formulario ingrese el siguiente código:

```
Function F(X As Double) As Double
     F = X ^ 2 + 2 * X + 3
End Function
```

Para invocar la función creada recientemente, codifique las siguientes líneas:

```
Dim A As Double, RESP As Double
A = InputBox("Ingrese A?")
RESP = F(A)
MsgBox (Str(RESP))
```

Paso de argumentos a una función

El paso de argumentos a una función es la forma como se ingresan los datos y variables al interior de la función y se presenta de dos formas:

- Por referencia
- Por valor

Paso de argumentos por referencia

En las funciones de Visual Basic, los argumentos se pasan por referencia (forma por defecto), de este modo cualquier cambio de valor que sufra un parámetro en el cuerpo de la función, también se produce en el argumento correspondiente de la llamada a la función. Esta forma de pasar los argumentos a una función es útil para funciones que devuelven más de un valor. Por ejemplo, considere la siguiente función que calcula las raíces reales de una ecuación cuadrática $Ax^2 + Bx + C = 0$.

```
Function RAIZ(A As Double, B As Double, C As Double, _
                    X1 As Double, X2 As Double) As Boolean
     Dim D As Double
     D = B ^ 2 - 4 * A * C
     If D >= 0 Then
          X1 = (-B - Sqr(D)) / (2 * A)
          X2 = (-B + Sqr(D)) / (2 * A)
          RAIZ = True    ' Verdadero si hay raíces reales
     Else
          RAIZ = False   ' Falso si no hay raíces reales
     End If
End Function
```

La llamada a esta función se puede realizar de la manera que se presenta a continuación:

```
     Dim A As Double, B As Double, C As Double
     Dim X1 As Double, X2 As Double
```

```
A = InputBox(“Ingrese A?”)
B = InputBox(“Ingrese B?”)
C = InputBox(“Ingrese C?”)
If RAIZ(A, B, C, X1, X2) Then
     MsgBox “X1= ” & Str(X1)
     MsgBox “X2= ” & Str(X2)
Else
     MsgBox “NO EXISTEN RAÍCES REALES”
End If
```

Paso de argumentos por valor

Cuando se ejecuta una función, se podrá especificar que el valor de un argumento no sea cambiado por esta función, pasando dicho argumento por valor. Para ello se debe anteponer la palabra reservada ByVal a la declaración del parámetro en la cabecera de la función. Por ejemplo:

```
Function F(ByVal X As Double) As Double
     F = X ^ 2 + 2 * X + 3
End Function
```

La cabecera de la función F especifica que X será pasado por valor y no por referencia.

Funciones recursivas

Se dice que una función es recursiva si se llama a sí misma. Por ejemplo la función FACTORIAL cuyo código se presenta a continuación es recursiva:

```
Function FACTORIAL(N As Integer) As Long
     If N <> 0 Then
          FACTORIAL = FACTORIAL(N - 1) * N
     Else
          FACTORIAL = 1
     End If
End Function
```

PROCEDIMIENTOS

La sintaxis para definir un procedimiento es la siguiente:

```
Private Sub NOMBRE_PROCEDIMIENTO([LISTA_PARÁMETROS])
     SENTENCIAS
     [Exit Sub]
End Sub
```

La explicación es análoga a la dada para las funciones. Sin embargo, un procedimiento no puede ser utilizado en una expresión, ya que un procedimiento no retorna ningún valor a través de su nombre.

La llamada a un procedimiento puede ser realizada de alguna de las dos formas siguientes:

```
Call NOMBRE_PROCEDIMIENTO([LISTA_ARGUMENTOS])
```

ó

```
NOMBRE_PROCEDIMIENTO([LISTA_ARGUMENTOS])
```

Por ejemplo, el siguiente código corresponde a un procedimiento que calcula e imprime la suma de los N primeros números naturales impares, es decir: S = 1 + 3 + 5 + . . . + N

```
Private Sub SUMA_IMPAR(N As Integer)
     Dim S As Integer, I As Integer
     S = 0
     For I = 1 To N
          If I Mod 2 <> 0 Then
               S = S + I
          End If
     Next
     MsgBox “La suma es S= ” & Str(S)
End Sub
```

La llamada a este procedimiento podría ser de la forma:

```
Call SUMA_IMPAR(25)
```