Universidad Autónoma Agraria

Antonio Narro

Unidad Laguna

VISUAL BASIC

PROGRAMACION

MC. SONIA LOPEZ GALINDO

No. EXP. 3068

CONTENIDO

- **UNIDAD I ELEMENTOS BÁSICOS**
- **Objetivo**: En esta unidad se describe a la programación orientada de objetos (OOP), en donde un programa es considerado un programa de objetos interactuando entre si; creando ambientes de desarrollo visual para facilitar la solución de un problema.
  - 1.1 Definición Visual Basic
  - 1.2 Conceptos básicos
  - 1.3 Tipos de variables
  - 1.4 Tabla de tipos de datos
  - 1.5 Operadores aritméticos de comparación y lógicos
  - 1.6 Jerarquía de operaciones
  - 1.7 Ventana principal
  - 1.8 Control Label (etiqueta)
  - 1.9 Control Text Box (caja de texto)
  - 1.10 Control Command Bulton (botón de comando)

- **UNIDAD 2 INSTRUCCIONES DE CONTROL DE PROGRAMA**
- **Objetivo:** Nos permite alternar la secuencia normal de ejecución de un programa.
  - 2.1 Instrucciones de control.
  - 2.2 Instrucciones condicionales.
  - 2.3 Operaciones relacionales.
  - 2.4 Instrucción IF…THEN (Si… entonces).
  - 2.5 Instrucciones Select Case (En caso selecto).
  - 2.6 Instrucción For… to (Para… a).
  - 2.7 Ciclo WHILE (Mientras).

CONTENIDO

- **UNIDAD 3. ARREGLOS (ARRAYS)**
- **Objetivo:** Son variables que permite almacenar un conjunto de datos del mismo tipo a la vez.

  3.1 Arreglos tradicionales.

  3.2 Arreglo tipo lista.

  3.3 Arreglo tipo tabla.

  3.4 Control List Box (Caja de lista)

  3.5 Control Grid (Cuadricula).

- **UNIDAD IV BASE DE DATOS**
- Objetivo: Nos permite almacenar conjuntos de datos en medios permanentes de almacenamiento.

  4.1 Almacenamiento de base de datos

  4.2 Tablas

  4.3 Operaciones con campos

  4.4 Búsquedas

  4.5 Filtros

UNIDAD I. ELEMENTOS BÁSICOS

1.1 Definición Visual Basic

Visual Basic es una aplicación y un lenguaje de programación desarrollados por Alan Cooper para Microsoft. Se origina en el clásico lenguaje BASIC. La aplicación Visual Basic, permite crear ventanas, botones, menús, etc. de forma sencilla con solo arrastrar y soltar los elementos. Luego se pueden definir las apariencias, posiciones y comportamientos tanto de forma visual como utilizando códigos de programación

1.2 Conceptos básicos

- **1.2.1 ¿Qué es un programa en Visual Basic?**
- Un programa está formado por una parte de código puro, y otras partes asociadas a los objetos que forman la interface gráfica.
- **1.2.2 ¿Qué es un evento?**
- Es una señal que comunica a una aplicación que ha sucedido algo importante. Los eventos también permiten que las tareas separadas se comuniquen. Suponga, por ejemplo, que una aplicación realiza una tarea de ordenación independientemente de la aplicación principal. Si un usuario cancela la ordenación, la aplicación puede enviar un evento de cancelación que ordene la detención
- **1.2.3 ¿Qué son objetos?**
- Los objetos son el elemento central de la programación en Visual Basic. Los formularios y controles son objetos. Las bases de datos son objetos.

1.2 Conceptos básicos

- **1.2.4 ¿Qué son métodos?**
- Los métodos son un conjunto de procedimientos que permiten que un objeto ejecute una acción o tarea sobre sí mismo.

- **1.2.5 ¿Qué son los componentes?**
- Es una clase que implementa la interfaz System.ComponentModel.IComponent o se deriva directa o indirectamente de una clase que implementa **IComponent**. Un componente es un objeto que es reutilizable, puede interactuar con otros objetos y proporciona control sobre recursos externos y compatibilidad en tiempo de diseño.

- **1.2.6 ¿Qué son propiedades?**
- Las propiedades son como las partes que constituyen al objeto, (Como por ejemplo, una persona se podría decir que presenta propiedades como sus: ojos, oreja, labios, pies, color, etc.), estas propiedades constituyen al control ya que ellas presentan valores, que le dan forma al mismo control.

1.3 Tipos de variables

- Visual Basic, al igual que la mayoría de los lenguajes de programación, utiliza variables para almacenar valores. Una *variable* tiene un nombre (la palabra que se utiliza para referirse al valor que contiene la variable) y un tipo de datos (que determina la clase de datos que puede almacenar la variable). Una variable puede representar una matriz, si es necesario que almacene un conjunto indizado de elementos de datos estrechamente relacionados entre sí.

Los distintos tipos de variables utilizados en Visual Basic son:

Integer	Valor Entero	2 Bytes
Long	Valor Entero Largo	4 Bytes
Single	Valor Real	4 Bytes
Double	Valor Real Doble	8 Bytes
String	Carácter (texto)	1 Byte por carácter
Byte	Byte	1 Byte
Boolean	Valor Booleano (1/0)	2 Bytes
Currency	Monedas y Punto Fijo	8 Bytes
Date	Fecha	8 Bytes
Object	Referencias a objetos	4 Bytes
Variant	Cualquiera	16-22 Bytes

1.4 Tabla de tipos de datos

- El *tipo de datos* de un elemento de programación hace referencia al tipo de datos que puede contener y a cómo se almacenan dichos datos. Los tipos de datos se aplican a todos los valores que pueden almacenarse en la memoria del equipo o participar en la evaluación de una expresión. Cada variable, literal, constante, enumeración, propiedad, parámetro de procedimiento, argumento de procedimiento y valor devuelto por un procedimiento tiene un tipo de datos.

Tabla de tipo de valores (1)

Tipo de Visual Basic	Estructura de tipo CommonLanguageRuntime	Asignación de almacenamiento nominal	Intervalo de valores
Boolean	Boolean	En función de la plataforma de implementación	True o False
Byte	Byte	1 byte	0 a 255 (sin signo)
Char (carácter individual)	Char	2 bytes	0 a 65535 (sin signo)
Date	DateTime	8 bytes	0:00:00 (medianoche) del 1 de enero de 0001 a 11:59:59 p.m. del 31 de diciembre de 9999.
Decimal	Decimal	16 bytes	0 a +/-79.228.162.514.264.337.593.543.950.335 (+/-7,9... E+28) † sin separador decimal; 0 a +/-7,9228162514264337593543950335 con 28 posiciones a la derecha del decimal; el número distinto de cero más pequeño es +/-0,0000000000000000000000000001 (+/-1E-28) †

Tabla de tipo de valores (2)

Tipo de Visual Basic	Estructura de tipo CommonLanguageRuntime	Asignación de almacenamiento nominal	Intervalo de valores
Integer	Int32	4 bytes	-2.147.483.648 a 2.147.483.647 (con signo)
Long (entero largo)	Int64	8 bytes	-9.223.372.036.854.775.808 a 9.223.372.036.854.775.807 (9,2...E+18 †) (con signo)
Object	Object (clase)	4 bytes en plataforma de 32 bits 8 bytes en plataforma de 64 bits	Cualquier tipo puede almacenarse en una variable de tipo **Object**
SByte	SByte	1 byte	-128 a 127 (con signo)
Short (entero corto)	Int16	2 bytes	-32.768 a 32.767 (con signo)
Single (punto flotante de precisión simple)	Single	4 bytes	-3,4028235E+38 a -1,401298E-45 † para los valores negativos; 1,401298E-45 a 3,4028235E+38 † para los valores positivos
String (longitud variable)	String (clase)	En función de la plataforma de implementación	0 a 2.000 millones de caracteres Unicode aprox.

1.5 Operadores aritméticos de comparación y lógicos

- -**Operadores aritméticos**
- Los operadores aritméticos se utilizan para realizar muchas de las operaciones aritméticas habituales que implican el cálculo de valores numéricos representados por literales, variables, otras expresiones, llamadas a funciones y propiedades, y constantes.

- -**Operadores de comparación**
- Los operadores de comparación comparan dos expresiones y devuelven un valor Boolean que representa la relación entre sus valores. Existen operadores para comparar valores numéricos, operadores para comparar cadenas y operadores para comparar objetos.

- -**Comparar valores numéricos**
- Visual Basic compara valores numéricos mediante seis operadores de comparación numéricos. Cada operador toma como operandos dos expresiones que se evalúan como valores numéricos.

La tabla siguiente enumera los operadores

Operador	Condición que prueba	Ejemplos
= (Igualdad)	¿Es igual el valor de la primera expresión que el de la segunda?	23 = 33 ' False 23 = 23 ' True 23 = 12 ' False
<> (Desigualdad)	¿Es distinto el valor de la primera expresión del valor de la segunda?	23 <> 33 ' True 23 <> 23 ' False 23 <> 12 ' True
< (Menor que)	¿Es el valor de la primera expresión menor que el valor de la segunda?	23 < 33 ' True 23 < 23 ' False 23 < 12 ' False
> (Mayor que)	¿Es el valor de la primera expresión mayor que el valor de la segunda?	23 > 33 ' False 23 > 23 ' False 23 > 12 ' True
<= (Menor o igual que)	¿Es el valor de la primera expresión menor o igual que el valor de la segunda?	23 <= 33 ' True 23 <= 23 ' True 23 <= 12 ' False
>= (Mayor o igual que)	¿Es el valor de la primera expresión mayor o igual que el valor de la segunda?	23 >= 33 ' False 23 >= 23 ' True 23 >= 12 ' True

Operadores lógicos

- Los operadores lógicos comparan expresiones Boolean y devuelven un resultado Boolean. Los operadores And, Or, AndAlso, OrElse y Xor son *binarios* porque toman dos operandos, mientras que el operador Not es *unario* porque toma un solo operando.

Operadores lógicos

El Not (Operador, Visual Basic) realiza la *negación* lógica en una expresión Boolean. Produce el contrario lógico de su operando.

El And (Operador, Visual Basic) realiza la *conjunción* lógica de dos expresiones Boolean. Si ambas expresiones se evalúan como True, And devuelve True. Si al menos una de las expresiones se evalúa como False, And devuelve False.

-Xor (Operador, Visual Basic) realiza la *exclusión* lógica de dos expresiones Boolean. Si exactamente una expresión, pero no ambas, se evalúa como True, Xor devuelve True. Si ambas expresiones se evalúan como True o como False, Xor devuelve False.

El Or (Operador, Visual Basic) realiza la *disyunción* o *inclusión* lógicas de dos expresiones Boolean. Si una de las expresiones o ambas se evalúan como True, Or devuelve True.

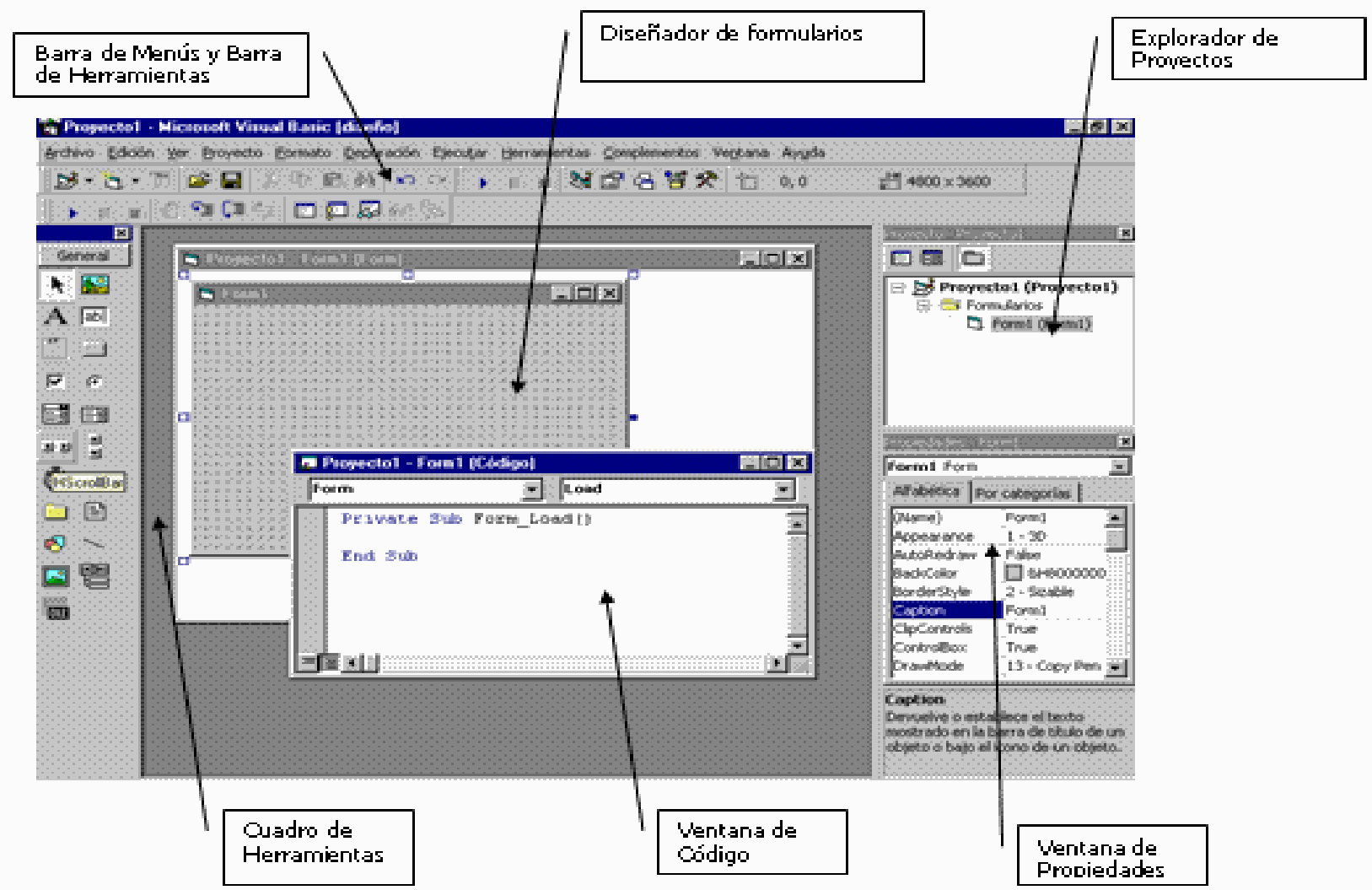
1.6 Jerarquía de operaciones

En Visual Basic recordar siempre, que antes de plantear una formula en un programa se deberá evaluar contra el siguiente:

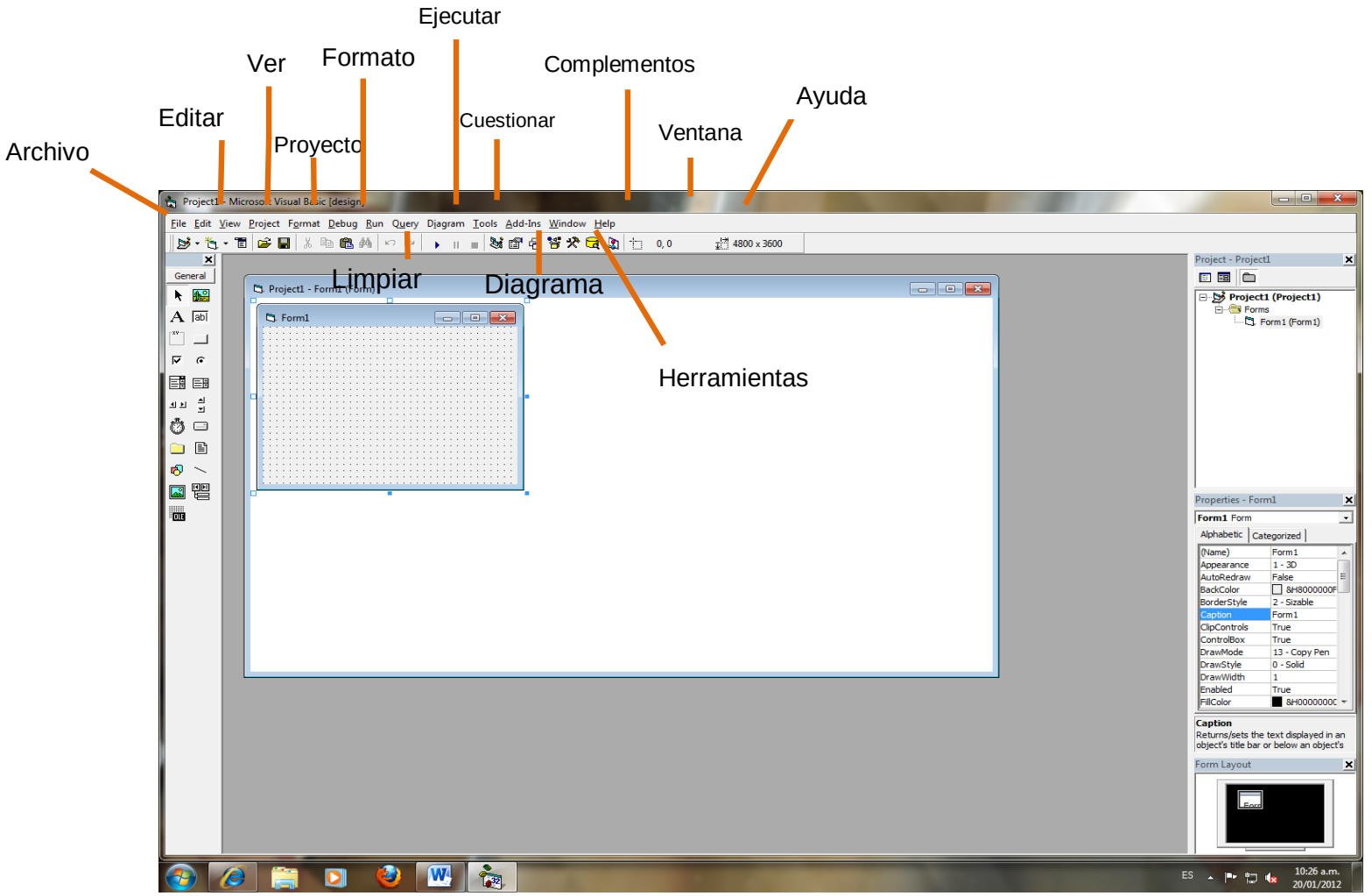
Orden de operaciones:

1.- Paréntesis
2.- Potencias y raíces
3.- Multiplicaciones y divisiones
4.- Sumas y restas
5.- Dos o más de la misma jerarquía u orden, entonces resolver de izquierda a derecha

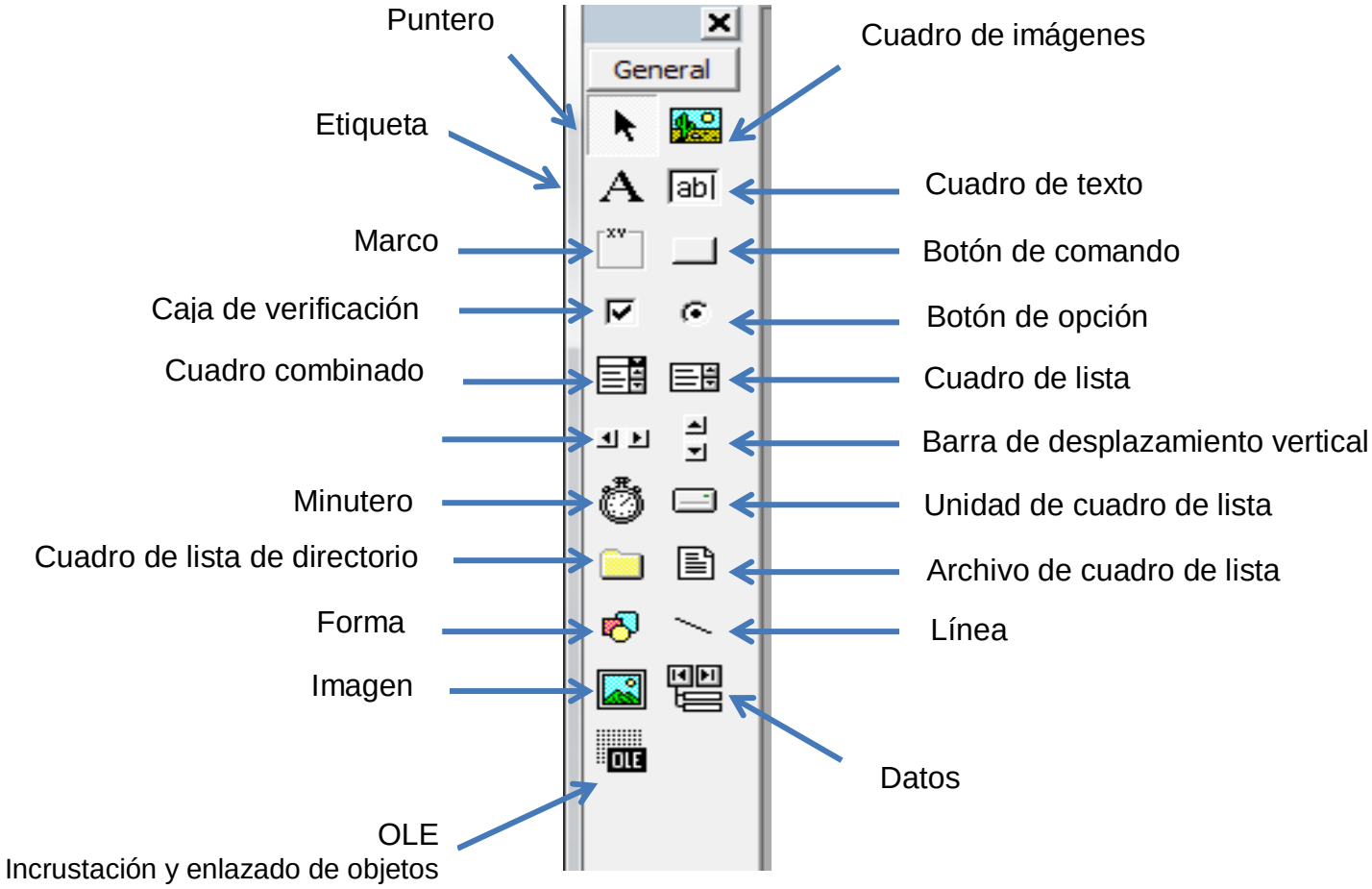
1.7 Ventana principal



Menús de VB


Ejecutar
Ver
Formato
Complementos
Ayuda
Editar
Cuestionar
Archivo
Proyecto
Ventana
Limpiar
Diagrama
Herramientas
File Edit View Project Format Debug Run Query Diagram Tools Add-Ins Window Help
Form1
Project1 (Project1)
Forms
Form1 (Form1)
Properties - Form1
Alphabetic
Categorized
Caption
Form Layout

Barra de Herramientas (Tools)



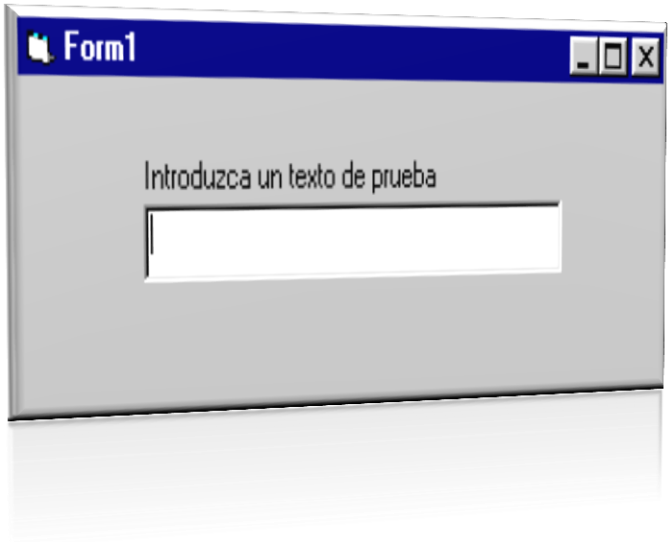
1.8 Control Label (Etiqueta).

Este control es también uno de los más utilizados, aunque su utilidad queda restringida a la visualización de datos en el mismo, no permitiendo la introducción de datos por parte del usuario.

- La forma de utilizarlo es dibujar el control en el formulario con el tamaño que queramos y asignarle un texto en tiempo de diseño o de ejecución esta vez sin utilizar la propiedad text puesto que no la incorpora, sino utilizando la propiedad caption.

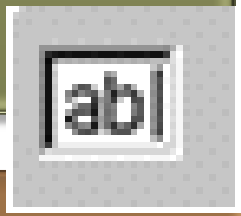
Control Label

- Este control sirve para mostrar mensajes en nuestro formulario que orienten al usuario sobre la utilidad de los demás controles que tengamos en la aplicación o para indicarnos acciones que podemos realizar



1.9 Control TextBox (Caja de texto).

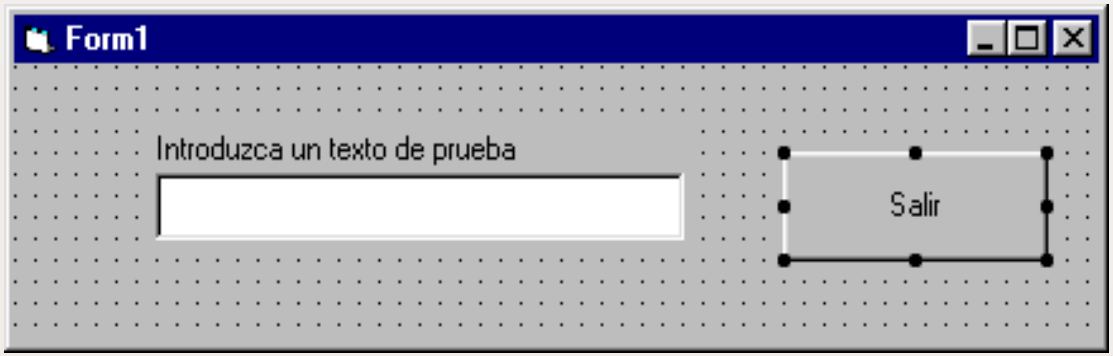
Mediante este control podremos realizar tanto la entrada como la salida de datos en nuestras aplicaciones. No hace falta que indiquemos las coordenadas de la situación del formulario en pantalla, simplemente tendremos que marcar sobre el control de la caja de herramientas y dibujarlo con el tamaño que queramos en nuestro formulario.



Las propiedades de las que dispone el control son las siguientes:(para obtener el cuadro de propiedades, seleccionar el control y pulsar F4 o pulsar con el boton derecho para obtener el menú contextual y marcar ***Propierties***

1.10 Control Command Button
(Botón de comando).

- Este control es el típico botón que aparece en todas las aplicaciones y que al hacer click sobre él nos permite realizar alguna operación concreta, normalmente Aceptar o Cancelar. Aunque según el código que le asociemos podremos realizar las operaciones que queramos.



- En el ejemplo anterior podemos añadir un control de este tipo para salir de la aplicación sin tener pulsar sobre la equis de la esquina superior derecha.

EJEMPLO


Form1
UNIVERSIDAD AUTONOMA AGRARIA
ANTONIO NARRO
MAESTRA: MC. SONIA LOPEZ
GALINDO
SIGUIENTE
SALIR

2.1 Instrucciones de CONTROL

- Las estructuras de control le permiten controlar el flujo de ejecución del programa. Tenemos dos tipos de estructuras de control:
- Estructuras de decisión
- Estructuras de bucle

2.2 Instrucciones CONDICIONALES

- Las instrucciones condicionales nos van a permitir representar
- Éste tipo de comportamiento.
- En otros casos, nos encontraremos con la necesidad de repetir una
- Instrucción ó instrucciones un número determinado de veces. En éstos Casos utilizaremos **instrucciones de control iterativas ó repetitivas**

2.3 Operaciones RELACIONALES.

Las estructuras de repetición o bucle le permiten ejecutar una o más líneas de código repetidamente.

Las estructuras de repetición que acepta Visual Basic son:

- **Do...Loop**
- **For...Next**

2.4 Instrucción IF…THEN (Si…entonces).

Use la estructura **If...Then** para ejecutar una o más instrucciones basadas en una condición. Puede utilizar la sintaxis de una línea o un bloque de varias líneas:

- ***If*** *condición* ***Then*** *Sentencias*
- ***If*** *condición* ***Then***

Sentencias
End If

Condición normalmente es una comparación, pero puede ser cualquier expresión que dé como resultado un valor numérico. Visual Basic interpreta este valor como **True** o **False**; un valor numérico cero es **False** y se considera **True** cualquier valor numérico distinto de cero. Si ***condición*** es **True,** Visual Basic ejecuta todas las *sentencias* que siguen a la palabra clave **Then.** Puede utilizar sintaxis de una línea o de varias líneas para ejecutar una sentencia basada en una condición, los siguientes dos ejemplos son equivalentes

EJEMPLO 1:

```
1  If cualquierFecha < Now Then CualquierFecha = Now

2  If cualquierFecha < Now Then
       CualquierFecha = Now
   End If
```

Observe que el formato de una única línea de **If...Then** no utiliza la instrucción **End If.** Si se desea ejecutar más de una línea de código cuando ***condición*** sea **True,** debe utilizar la sintaxis de bloque de varias líneas **If...Then...End If.**

```
1  If cualquierFecha < Now Then
       CualquierFecha = Now
       Timer1.Enabled = False ' Desactiva el control Timer.
   End If

2  If chkAlumnoUNI.Value=1 Then
       txtCosto = Format (txtCosto*0.70,"Fixed")
       txtCódigo.Enabled = True
   End If
```

EJEMPLO 2

1

```
Private Sub mnuCut_Click (Index As Integer)
        If Index = 0 Then       'Comando Cortar
                CopyActiveControl   'Llama a procedimientos generales
                ClearActiveControl
        ElseIf Index = 1 Then 'Comando Copiar
        CopyActiveControl
    ElseIf Index = 2 Then       'Comando Borrar
                ClearActiveControl
        Else                                    'Comando Pegar
            PasteActiveControl
    End If
End Sub
```

2

```
If ClaveUsuario="DSI" Then
        'Permite al usuario entrar al sistema
        ...
        ...
Else
        'Mostrar un mensaje advirtiendo error en la clave
        ...
        ...
End If
```

3

```
Private Sub DeterminaCondición ( )
        If Val (txtPromedio)  >=13 Then
                txtCondición = "Aprobado"
        ElseIf Val (txtPromedio)  >=  10 Then
                txtCondición  =  "Asistente"
        Else
                txtCondición  =  "Desaprobado"
        End If
End Sub
```

2.5 Instrucciones Select Case (En caso selecto).

- Visual Basic proporciona la estructura **Select Case** como alternativa a **If...Then...Else** para ejecutar selectivamente un bloque de sentencias entre varios bloques. La sentencia **Select Case** ofrece posibilidades similares a la instrucción **If...Then...Else,** pero hace que el código sea más legible cuando hay varias opciones.

- La estructura **Select Case** funciona con una única expresión de prueba que se evalúa una vez solamente, al principio de la estructura. Visual Basic compara el resultado de esta expresión con los valores de cada **Case** de la estructura. Si hay una coincidencia, ejecuta el bloque de sentencias asociado a ese **Case:**

Bloque de Sentencia Select case

- Cada ***lista_expresiones*** es una lista de uno a más valores. Si hay más de un valor en una lista, se separan los valores con comas. Cada ***bloque de sentencias*** contiene cero o más instrucciones.

- Si más de un **Case** coincide con la expresión de prueba, sólo se ejecutará el bloque de instrucciones asociado con la primera coincidencia. Visual Basic ejecuta las instrucciones de la cláusula (opcional) **Case Else** si ningún valor de la lista de expresiones coincide con la expresión de prueba.

```
Selec Case expresión_prueba
    [Case lista_expresiones1
        [bloque de sentencias 1]]
    [Case lista_expresiones2
        [bloque de sentencias 2]]
        .
        .
        .
    [Case Else
        [bloque de sentencias n]]
End Select
```

EJEMPLO 1

1

```
Private Sub mnuCut_Click (Index As Integer)
        Select Case Index
          Case 0                    ' Comando Cortar
                CopyActiveControl ' Llama a procedimientos generale
                ClearActiveControl
          Case 1                ' Comando copiar.
                CopyActiveControl
          Case 2                ' Comando borrar.
                ClearActiveControl
          Case 3                ' Comando Pegar.
                PasteActiveControl
          Case Else
                frmFind.Show ' Muestra el cuadro de
                                ' diálogo Buscar.
    End Select
  End Sub
```

2

```
Select Case TipoUsuario
        Case "Supervisor"
          ' Proporciona al usuario privilegios de Supervisor
          ...
          ...
     Case "Usuario"
          ' Proporciona al usuario privilegios de Usuario
          ...
          ...
    Case Else
          ' Proporciona al usuario privilegio de invitado
          ...
          ...
End Select
```

2.6 Instrucción For… to (Para… a)

- Los bucles **Do** funcionan bien cuando no se sabe cuántas veces se necesitará ejecutar las ***sentencias*** del bucle. Sin embargo, cuando se sabe que se va a ejecutar las ***sentencias*** un número determinado de veces, es mejor elegir el bucle **For...Next.** A diferencia del bucle **Do,** el bucle **For** utiliza una variable llamada ***contador*** que incrementa o reduce su valor en cada repetición del bucle.

La sintaxis es la siguiente:

- ***For*** *contador = iniciar* ***To*** *finalizar [****Step*** *incremento]*
- *Sentencias*
- ***Next*** *[contador]*

Los argumentos **contador, iniciar, finalizar** e **incremento** son todos numéricos.

SINTAXIS

Establece contador al mismo valor que ***iniciar.***

Comprueba si ***contador*** es mayor que ***finalizar.*** Si lo es, Visual Basic sale del bucle. (Si ***incremento*** es negativo, Visual Basic comprueba si ***contador*** es menor que ***finalizar.***)

Ejecuta las ***sentencias.***

Incrementa ***contador*** en 1 o en ***incremento,*** si se especificó.

Repite los pasos 2 a 4.

Este código imprime los nombres de todas las fuentes de pantalla disponibles:

```
Private Sub Form-Click ( )
    Dim I As Integer
    For i = 0 To Screen.FontCount
    Print Screen.Fonts (i)
    Next
End Sub
```

Por ejemplo

El siguiente procedimiento **Sub** abre la base de datos Biblio.mdb y agrega el nombre de cada tabla a un cuadro de lista.

```
Sub ListTableDefs ()
    Dim objDb As Database
    Set objDb = OpenDatabase("c:/Archivos de programa/Devstudio/" & _
        "vb/biblio.mdb", True, False)
    For Each TableDef In objDb.TableDefs ()
            List1.AddItem TableDef.Name
    Next TableDef
End Sub
```

2.7 Ciclo WHILE (Mientras).

- Cuando Visual Basic ejecuta este bucle **Do**, primero evalúa ***condición***. Si ***condición*** es **False** (cero), se salta todas las ***sentencias***. Si es **True** (distinto de cero) Visual Basic ejecuta las ***sentencias***, vuelve a la instrucción **Do While** y prueba la condición de nuevo.

- Por tanto, el bucle se puede ejecutar cualquier número de veces, siempre y cuando ***condición*** sea distinta de cero o **True**. Nunca se ejecutan las ***sentencias*** si ***condición*** es **False** inicialmente. Por ejemplo, este procedimiento cuenta las veces que se repite una cadena destino dentro de otra cadena repitiendo el bucle tantas veces como se encuentre la cadena de destino:

EJEMPLO

```
Function ContarCadenas (cadenalarga, destino)
    Dim posición, contador
    posición = 1
    Do While InStr (posición, cadenalarga, destino)
        posición = InStr (posición, cadenalarga, destino)+1
      contador = contador + 1
    Loop
    ContarCadenas = contador
End Function
```

Do

Sentencias

Loop While condición

Hay otras dos variantes análogas a las dos anteriores, excepto en que repiten el bucle siempre y cuando **condición** sea **False** en vez de **True.**

Arreglo de Controles

- Cuando creamos un arreglo de controles, todos los controles que forman el arreglo deben tener el mismo nombre (Propiedad Name), la propiedad **Index** establece el índice de cada control en el arreglo, esta propiedad comienza desde 0.

- Un arreglo de controles es un grupo de Controles que comparten el mismo:

- Tipo de objeto
- Nombre del control
- Procedimientos de evento

Por ejemplo

```
Sub ListTableDefs ( )
    Dim objDb As Database
    Set objDb = OpenDatabase("c:/Archivos de programa/Devstudio/" &_
        "vb/biblio.mdb", True, False)
    For Each TableDef In objDb.TableDefs ( )
            List1.AddItem TableDef.Name
    Next TableDef
End Sub
```

BASE DE DATOS