# Delphi 6.0

## ¿QUÉ ES UN PROGRAMA?

Es una secuencia ordenada y estructurada de pasos y sentencias. Este se divide en "**Subprogramas**" llamados *unidades*, y estas a su vez de dividen en **módulos** los cuales se llaman *procedimientos* que se componen de *bloques de programa*, los cuales son una serie de instrucciones que realizan una tarea muy concreta.
Todos los bloques de programa comienzan por la palabra Begin y terminan con la palabra end. Un ejemplo es:

```
Begin
a := b + c;
Insertar(a);
end;
```

En el ejemplo, después de cada instrucción (excepto begin), se coloca un punto y coma. En Pascal es obligatorio poner punto y coma, pues identifica el fin de una instrucción.

## COMENTARIOS

Es posible incluir en el código texto aclarativo que ayude a entender lo que la instrucción esta realizando. A ese texto se le denomina *comentario*.
En Delphi hay 3 formas distintas de poner comentarios:

Encerrando el texto aclarativo entre llaves ({ }).
Encerrando el texto aclarativo entre paréntesis y asterisco (* *).
Anteponiendo al texto dos símbolos de diagonal (//). Unicamente una línea de texto.

Ejemplo:

```
(*
Todo este trozo es un comentario
*)
begin {esto también es un comentario}
a := b + c;  // aquí se esta haciendo una suma
Insertar(a);
end;
```

Los comentarios no son considerados por Delphi como parte del código. Su única función es la de aclarar el código.

## CONSTANTES Y VARIABLES

Cuando se realiza un programa, constantemente se tiene que almacenar valores dentro del código, valores que pueden ser resultado de una operación. Una variable puede cambiar su valor en cualquier momento del programa, una constante siempre mantendrá el mismo valor durante el programa. Sin embargo se deben seguir unos consejos:

No tiene que haber sido utilizada para identificar otra variable.
No puede tener el nombre de una orden de Pascal.
No puede empezar con un número, ni tener espacios en blanco.
No puede contener caracteres "raros", como comillas, acentos, símbolos de puntuación ni la letra **ñ**. Sin embargo puede tener letras, números(siempre que no sea al principio) y el símbolo "_".

Las constantes se declaran del siguiente modo: en primer lugar se pone la palabra reservada *const* seguida de la contante, el signo igual y el valor de la constante (Sin olvidar el punto y coma reglamentario).
Ejemplos:

```
Const pi = 3.141592;
Const B = 18;
Const anio = 2001;
Const tamano = 12;
```

Las variables se declaran de modo distinto a como lo hacen las constantes. Antes de utilizarla hay que declarar qué es lo que contendrá, para que reserve el espacio en memoria para ella. Antes de empezar a definir variables debes de poner la palabra *var* y deberá ser antes de la palabra de inicio *begin*, abajo el nombre de la variable, seguido de 2 puntos y el tipo de valor que contendrá. Ejemplo:

```
Var
  Nombrevariable1 : tipo;
  Nombrevariable2 : tipo;
```

...

| Nombre (Tipo) | La variable almacenará |
|---|---|
| Boolean | Un valor lógico: Verdadero (TRUE) o Falso (FALSE). |
| Char | Una sola letra. |
| Byte | Un número pequeño de 0 a 255. |
| Smallinit | Un número entero de –32768 a 32767. |
| Word | Un número entero de 0 a 65535. |

| | |
|---|---|
| Integer o Longinit | Un número entero de –2,147’483,648 a 2,147’483,647 |
| Cardinal | Un número entero de 0 a 2,147’483,647 |
| Real | Un número entero y decimal. |
| Double | Un número entero y decimal mas preciso que Real. |
| String | Cadena de caracteres. |

Ejemplos:

```
Var
  Edad : byte;
  Nombre : string;
  DNI : cardinal;
  Curso : byte;
  Puntuación : real;
  Varon : boolean;
```

Cada variable va acorde con lo que almacenara, pues *edad* tiene un tipo byte, pues no almacenara un valor grande. *Varon* es de tipo boolean para que TRUE indique “SI” y FALSE indique “NO”.

ASIGNACIÓN

Una vez definida la variable, hay que darle un valor. Esto se hace siguiendo la sintaxis siguiente:

Variable := valor;

Esto es la *Asignación*, únicamente se debe anteponer el nombre de la variable, seguido de 2 puntos y un signo igual y finalmente el valor que almacenará. Ejemplo:

Edad := 24;
Nombre := ‘David Osornio’;

También es posible realizar operaciones aritméticas en una asignación como por ejemplo:

A := 1+2;
Res := 7+2;

Tan sólo se tiene que saber como indicar las operaciones aritméticas. Las operaciones aritméticas se describen en la siguiente tabla:

| Operación | Descripción | Ejemplo |
|---|---|---|
| + | Suma | a := 1 + 4; |

| | | |
|---|---|---|
| - | Resta | a := 8 – 2; |
| * | Multiplicación | a := 2 * 17; |
| / | División de números reales | a := 14 / 3; |
| Div | División de números enteros | a := 14 div 3; |
| Mod | Resto de la división | a := 14 mod 3; |

**REGISTROS**

En algunas ocasiones, necesitamos construir variables que se compongan a su vez de otras variables. Por ejemplo, si se desea construir una variable de los datos de una persona y que esta a su vez este construida de otras variables mas limitadas como nombre, teléfono, edad. Los registros se declaran del siguiente modo:

```
Variable: record
  Subvariable: tipo;
  Subvariable: tipo;
  ...
  end;
```

Donde variable es el nombre de la variable "*contenedora*"( llamada registro) y las subvariables, son los nombres de las variables contenidas, las que se denominan "*Campos del registro*".

```
Var
Fichapersona : record
  Nombre: string;
  Edad: byte;
  Telefono: longint;
End;
```

De este modo se declara una variable de registro llamada Fichapersona que se compone de 3 variables: Nombre, Edad y Telefono. Para acceder a los datos de este registro, se deberán colocar los valores del siguiente modo.

```
Fichapersona.Nombre := 'David Osornio Fernández';
Fichapersona.Edad := 22;
Fichapersona.Telefono := 7550164;
```

MATRICES

En Pascal, podemos definir matrices del mismo tipo que las variables, las cuales son variables divididas en secciones

independientes que pueden almacenar distintos valores. Las matrices se declararán del siguiente modo.

Nombre_matriz :
array[1..elementos1,1..elementos2,1..elementos3,  ...] of tipo;

Donde el *Nombre_matriz*, es el nombre que recibirá la matriz de datos, *array* es para definir que se tratará de una matriz de datos. Y los elementos serán el numero de celdas que tendrá dicha dimensión. Ejemplos:

```
a : array[1..10] of byte;
b : array[1..100, 1..20] of word;
c : array[1..10, 1..10, 1..10] of float;
```

## OPERACIONES CON CADENAS

Las cadenas de tipo string, utilizan operadores especiales para trabajar con ellas:

Unión.- Podemos unir dos cadenas en una sola utilizando el signo +. Por ejemplo:

```
Nombre := 'David';
Apellido := 'Osornio';
NombreCompleto := nombre + ' ' + apellido;
```

Recorte.- Podemos eliminar letras de una cadena utilizando la función *delete*, que se utiliza del siguiente modo:

```
Delete (cadena, inicio, cantidad);
```

Donde *cadena* es la variable con la que estamos trabajando, *inicio* es la posición en la que vamos a iniciar el recorte y *cantidad* es el número de caracteres que quitaremos.

```
a := 'Hola a todo el mundo!';
delete (a, 6, 10);
```

En este código estamos eliminando de la variable a los 10 caracteres que están a partir de la posición 6, por lo que después de esta instrucción la variable *a* tendrá el valor '*Hola mundo!*'.

## CONVERSIÓN DE TIPOS

Object Pascal es lo que se denomina un lenguaje *fuertemente tipado*. Esto quiere decir que no permite asignar valores entre variables de distinto tipo. Esto es que si tenemos una variable *a* de tipo byte y una variable *c* de tipo string, y ésta ultima contiene el valor '3', no podremos hacer algo como:

```
a := c;
```

Ya que con esto no le damos a la variable *a* el valor 3, sino que se le da el valor '3', se le esta dando una cadena de texto, no un número.
Para permitir asignaciones entre variables de distinto tipo, e utilizan las siguientes funciones:

Trunc.- esta función permite asignar un número real a una variable de tipo byte, word o integer. Recibe como parámetro un número real y retorna dicho número sin parte decimal. No hace redondeo, solo elimina la parte que está después del punto decimal. Ejemplo:

```
a : byte;
f : real;
f : 1.8934;
a := trunc (f);      // a = 1
```

StrToInt.- Permite asignar un texto que contiene en su interior un número a una variable de tipo entero. Si la cadena no contiene un número, sino letras, se produce un error. Ejemplo:

```
a : string;
b : integer;
a := '123';
b .= StrToInt (a);      // b=123
```

IntToStr.- Es la contraria de la función anterior. Permite convertir un número entero a cadena. Ejemplo:

```
a : integer;
b : string;
a := 12;
b := 'el valor de a es   ' + IntToStr (a);        //  b= 'el valor de a es   12'
```

ESTRUCTURAS DE DECISIÓN

Son aquellas instrucciones que permiten al usuario obtener distintas respuestas en base al valor introducido o al resultado de alguna operación. Se seguirá la siguiente sintaxis:

```
if condición then instrucción1
else instrucción2;
```

Después del **if**(“*si*”) debe ir la condición que ha de evaluarse. A continuación de ésta , se coloca la palabra **then**(“*entonces*”) seguida de la *instrucción1*, que se ejecutará si la condición es cierta o se cumple. En caso contrario, se ejecutara la *instrucción2*, que esta precedida por la palabra **else**(“*si no*”). Ejemplo:

```
If a = b Then c := 1
Else c := 2;
```

En caso de que se deseen poner mas instrucciones en un solo bloque, se deberá realizar del siguiente modo:

```
If a = b then
Begin
  c := 1;
  d := 2;
  e := 3;
end
Else
Begin
  c := 5;
  d := 3;
  e := 0;
end;
```

La parte del Else no es obligatoria, y puede ser omitida, con lo cual la instrucción quedara reducida, sin embargo, si la parte else se omite, se deberá poner punto y coma al final de la instrucción. Ejemplo:

```
If a = b Then
Begin
  c := 1;
  d := 2;
  e := 3;
end;
```

En este ejemplo, si las variables *a* y *b* son iguales se darán valores a las variables c, d y e, en caso contrario seguirán con sus valores iniciales.

EXPRESIONES BOOLEANAS

En Pascal se llaman expresiones booleanas a las expresiones que indican una condición y que producen un resultado de *verdadero* o *falso*.
Una expresión booleana se puede colocar de dos modos distintos:

1. Una variable o constante seguida de un operador de comparación y seguida de otra variable.

2. Una expresión booleana seguida de un operador booleano y de otra operación booleana.

Los operadores que se utilizan en Delphi son:

| Operador | Descripción | Ejemplo |
|---|---|---|
| > | mayor que | > b |
| < | menor que | < 4 |
| >= | mayor o igual | a >= 3 |
| <= | menor o igual | c <= 7 |
| <> | distinto | a <> 0 |
| = | igual | a = 4 |

Las operaciones booleanas sirven para unir expresiones y son las que a continuación se presentan.

| Operación | Descripción | Ejemplo |
|---|---|---|
| and | y | (a > b) and (c < 3) |
| or | o | (a > b) or (c < 3) |
| not | no | not (a = b) |

Un ejemplo de una estructura de decisión con una expresión booleana es el siguiente:

```
If ((a > b) and (c < 3)) or not (a > 7) Then c:= 0;
```

**ANIDAMIENTOS**

Es sabido que después de la palabra reservada *Then* se pueden ejecutar secuencias de instrucciones las cuales pueden ser de cualquier tipo, ahora bien, ¿Es posible colocar un *if* dentro del

bloque then de otro?. La respuesta es si, a estas sentencias de "*If´s*" encadenados se le denomina anidamiento.
***El punto clave es tomar en cuenta que cuando se coloquen anidamientos es recomendable colocar las instrucciones Begin y End entre Then y Else de este modo sabremos exactamente a que if pertenece cada Else. Ejemplo:***

```
If a = 7 then
Begin
If b = 2 then
c := 3
Else
c := 2
End
Else
c := 1;
```

## SELECCIÓN MULTIPLE

Cuando se realiza un programa, es frecuente encontrarse con alguna variable que según su valor realizara alguna acción. Esto se podría realizar con muchos If´s anidados, pero resultaria algo enredado, por ejemplo: Si se desea que cuando *a* tenga el valor 1, *c* tome el valor 10, cuando *a* tenga el valor 2, *c* tome el valor 15, cuando *a* tenga el valor 3, *c* tome el valor 20 y cuando no sea alguno de los 3 valores, entonces que *c* tome el valor 0:

```
If a = 1 then c := 10 Else
If a = 2 then c := 15 Else
If a = 3 then c := 20 Else
c := 0;
```

Esta forma de tomar decisiones resulta muy poco ortodoxa. El lenguaje Pascal nos ofrece para dicho propósito otra forma mas fácil de hacerlo. Mediante la palabra reservada *case of*. La sintaxis de dicha instrucción es la siguiente.

```
case variable of
valor1: acción1;
valor2: acción2;
....
Else acción N;
End;
```

Donde *variable* es el identificador de la variable que será comprobada y *valor1, valor2...* son los diferentes valores que puede tomar dicha variable. Si tomamos el problema anteriormente planteado, tendremos que la solución sería de este modo:

```
case a of
1: c := 10;
2: c := 15;
3: c := 20;
```

```
Else c:= 0;
End;
```

## BUCLES

En algunos programas, es necesario repetir la misma acción un número determinado de veces, hasta que se cumpla una acción determinada.

En Delphi existen 3 tipos distintos de Blucles:

**Bucle "Para"**.- Se utiliza para efectuar un número concreto de ocasiones la misma secuencia de acciones. Su sintaxis es:

```
For variable := inicio to fin do
Begin
<código a repetir>
End;
```

Para repetir una suma 10 veces, se utilizará la siguiente instrucción:

```
a := 1
For i := to 10 do
Begin
a := a + 1;
End;
```

**Bucle "Mientras"**.- Se utiliza para repetir las acciones mientras se cumpla una condición. Su sintaxis es:

```
While condición do
Begin
<código a repetir>
End;
```

En el momento que la condición deje de cumplirse, el bucle terminará y seguirá con la siguiente instrucción del código. Ejemplo:

```
a := 1;
j := 1;
While (j <=10) do
Begin
a := a +j;
j := j +1;
End;
```

**Bucle "Repetir"**.- Se utiliza para repetir las mismas acciones hasta que se cumpla una condición determinada.

```
Repeat
<Código a repetir>
Until condición;
```

En el momento en que se cumpla la condición, terminará el bucle. Ejemplo:

```
a := 1;
j := 1;
Repeat
a := a * j;
j := j+1;
Until j=10;
```

En este ejemplo, se obtendrá el factorial de 10.
***La diferencia entre el Bloque While y el Bloque Repeat es que en el primero, se evalúa la condición antes de ejecutar las acciones, por lo que es posible que no se ejecute ni una sola vez. Mientras que Repeat, evalúa la condición después de ejecutarla una vez.***

## PROCEDIMIENTOS Y FUNCIONES

Delphi, es un lenguaje estructurado, lo cual indica que los programas escritos en este lenguaje, pueden descomponerse en pequeños módulos que pueden ser llamados cuando los necesitemos. Estos módulos en Pascal se llaman *funciones* y se identifican mediante un nombre. Se declaran del siguiente modo:

```
function nombre (argumento1: tipo; argumento2: tipo;...) : tipo;
Begin
<Acciones a realizar>
End;
```

Donde *nombre* es el nombre que se asignará a la función, *argumento1* y *argumento2* son valores que pasan a la función y *tipo* es el tipo de valor que retornará como resultado. Por ejemplo, si queremos implementar en Pascal la función matemática: *f(x, y) = (x * x + y) / x* siendo *x* e *y* números enteros, se realizaría de este modo:

```
Function f (x: integer; y: integer) : integer;
```

```
Begin
F = (x * x + y) / x;
End;
```

Dicha función, se definirá una sola ocasión durante el programa, y únicamente será necesario hacer llamada a esta función mediante su nombre y sus valores dados entre paréntesis.

```
a := f(1, 2); {llamada a f con x=1 e y=2}
b := f(a, 3); {llamada a f con x=a e y=3}
c := f(a, b) + f(1, 4);
```

Los valores colocados entre paréntesis de la función, reciben el nombre de argumentos. Estos se comportan dentro de la función, como si se tratase de variables.

## PROCEDIMIENTOS

Toda función que no retorna valores ningún valor, o que no realiza operaciones matemáticas o genera resultados numéricos, recibe el nombre de procedimiento. La programación estructurada, se basa en dividirse en pequeñas partes autosuficientes que realizan una tarea muy concreta, y que pueden ser invocados cuando se desee. Se puede realizar alguna función o tarea especifica que arroje información de modo no numérico, o sin realizar operaciones, es entonces cuando se utilizan los procedimientos.
Un procedimiento se declara del mismo modo que una función, a diferencia que la palabra reservada *Function* cambia por *Procedure* y al final de la lista de argumentos no se pone ningún tipo de valor de respuesta, pues no arroja ningún resultado. Ejemplo:

```
Program Ejemplo2;
Procedure Di (mensaje: String);
Begin
Showmessage(mensaje);
End;
Begin
Di ('Mi nombre es: ');
Di ('David Osornio Fernández');
End;
```

También es posible construir funciones o procedimientos que no tengas argumentos, pues pueden realizar una tarea concreta, no importando alguna otra cosa. Ejemplo:

```
Program Ejemplo3;
Procedure Saluda;
Begin
Showmessage('Hola a todos');
End;
Procedure Nombre;
```

```
Begin
Showmessage('Mi nombre es David Osornio Fernández');
End;
Procedure Despidete;
Begin
Showmessage('Adiós a todos');
End;
Saluda;
Nombre;
Despidete;
End;
```

## PARAMETROS POR VALOR Y POR REFERENCIA

¿Que es lo que sucede cuando enviamos una función matemática a un procedimiento y nos debe enviar un resultado?. Por ejemplo:

```
Program Ejemplo;
Procedure cuadrado (a: byte);
Begin
a := a * a;
End;
Var c, d :byte;
Begin
c := 3;
cuadrado (c);
d := c;
End;
```

La pregunta del millón de Dólares, es... ¿Cuánto vale *d* al final del programa?, se pensara que *d* vale 9 pero, en realidad vale 3, pues al pasar un valor numérico a un procedimiento y no a una función, no se rescata ningún resultado, pues no tiene declaración del tipo de variable que arrojará. Si se desea realizar esto se deberá declarar en la sección correspondiente, del siguiente modo:

```
Program Ejemplo;
Procedure cuadrado (Var a: byte);
Begin
a := a * a;
End;
Var c, d :byte;
Begin
c := 3;
cuadrado (c);
d := c;
End;
```

***Es recomendable que cuando utilices funciones y procedimientos, estos sean lo mas cortos posibles y si algún procedimiento o función es demasiado extenso, puede ser dividido en procedimientos mas pequeños, de no mas de 30 líneas de código.***

## OBJETOS Y CLASES

Una clase es un registro en el que sus campos pueden ser tanto variables como procedimientos, y un objeto es una variable que tiene como tipo una clase.
La declaración de una clase es similar a la de un registro, sólo que en lugar de utilizar la palabra *record* se utiliza *class*, y que podremos introducir procedimientos y funcione en el. Ejemplo:

```
Type variable_clase = class
                    Subvariable1 : tipo;
                    Subvariable2 : tipo;
                    Subvariable3 : tipo;
                    ...
                    Procedimientos <opcionales>
                    Funciones <opcionales>
End;
```

La clase es tomada como un objeto con propiedades, las cuales serán visualizadas o modificadas, mediante la colocación de un punto entre la clase y las variables que esta contenga. Ejemplo:

```
Type persona = class
                    Nombre : String;
                    Dni : longinit;
                    Edad : byte;
End;
Var Pablo, Javier : persona;
```

Para asignar valores a las nuevas clases, por ejemplo colocarle el nombre de Pablo Díaz a la Subvariable nombre de la variable Pablo, es necesario lo siguiente:

```
Pablo.nombre := 'Pablo Díaz';
Pablo.Edad := 24;
```

A los campos que son variables, se les denomina *atributos* y a los procedimientos y funciones se les llama *métodos*.

## ENCAPSULAMIENTO

Cuando se genera un registro o una clase, es posible acceder a los métodos y a los atributos, sin embargo, si deseamos que sea imposible acceder a una parte de la clase, como por ejemplo a los atributos, se deberá declarar a que parte es posible acceder mediante la palabra reservada P*ublic* y las secciones a las que será imposible acceder desde fuera del objeto *Private*. Ejemplo:

```
Type persona = Class
                Private
                Nombre : String;
                Dni : Longinit;
                Edad : Byte;
                Public
                Function comosellama : String;
                Function QueEdad : byte;
End;
```

Mediante este ejemplo, hemos declarado como secciones inaccesibles *nombre, Dni, y edad* por lo cual no sería posible acceder desde fuera del código. Con esta definición, por ejemplo no sería posible hacer lo siguiente:

```
Javi.Nombre := 'Javi García';
```

## ¿QUÉ ES UN COMPONENTE?

Son los objetos que se colocan dentro del formulario, y forman la interfaz del usuario. Estos se encuentran en la caja de herramientas ubicada en la parte superior de la pantalla. Pueden ser agrupados en 3 distintos tipos:

Controles ***normales:*** Son los que vienen por defecto en Delphi, los que están en la caja de herramientas. Delphi los clasifica en varios apartados. Estos son:

Standard: Posee la mayoría de los componentes básicos de una aplicación.

Additional: Contiene controles para aplicaciones un poco mas avanzadas.

Win32: En el se encuentran los controles comunes de los entornos de 32-bits.

System: Contiene los controles para manejar el sistema.

Internet: Contiene controles para trabajar con Internet.

Data Access: Controles que permiten acceder a las tablas de datos, realizar consultas, etc.

Data Controls: Contiene controles para manejar los datos contenidos en Tablas. Se necesitan algunos controles de la pestaña Data Access.

Decision Cube: Controles que permiten manejar datos multidimencionales.

Qreport: Controles que permiten realizar informes de forma rápida.

Dialog: Controles que permiten a nuestras aplicaciones mostrar ventanas de dialogo comunes.

Win 3.1: Componentes ya antiguos que se incluyen en esta versión para permitir compatibilidad con las aplicaciones de Windows 3.11.

Samples: Aquí se depositan componentes de ejemplo y los que serán creados.

ActiveX: componentes ActiveX.

Controles ***instalables***: Son aquellos que pueden ser agregados a Delphi, estos se pueden agregar en una pestaña personalizada y pueden ser utilizados cuando se desee, estos se encuentran en el mercado o incluso en Internet, y si se tiene suficiente conocimiento del tema, pueden ser creados los controles propios.

Objetos ***insertables***: No son controles, pero funcionan como si lo fueran, pues se basan en la tecnología OLE para presentarse en el formulario, de este modo, se puede utilizar el editor de gráficos preferido en alguna aplicación.

## PROPIEDADES DE LOS CONTROLES

Cada componente tiene atributos específicos que permiten el desarrollo de una aplicación. Estos son las *propiedades*, los *eventos* y los *métodos*:

Las *propiedades* son características del control, y recogen cualidades como altura, anchura, color...

Los *eventos* son las acciones que pueden realizarse sobre dicho control estas pueden ser el pasar el Mouse por encima del control, dar click con el Mouse, etc.

Los *métodos* son las acciones que realizará el control al ejecutar algún evento, este es en si el código del control.

Las propiedades básicas de los controles son:

Cursor: Por defecto, cuando ponemos el cursor del Mouse encima de un componente, el cursor permanece presentando una flecha, sin embargo es posible cambiarlo mediante esta propiedad.

Enabled: Esta propiedad indica si un objeto esta activado o desactivado, esto es si puede ser utilizado o no.

Font: Determina el tipo de fuente de letra que tendrá el objeto, cada objeto puede tener distinto tipo de letra, sin embargo se sugiere siempre respetar el mismo estilo.

Height: Altura del objeto dada en Pixeles.

Hint: Es una pequeña etiqueta de ayuda que se puede emplear para ayudar al usuario a saber el uso de dicho botón. Para que la etiqueta sea visible, la propiedad *ShowHint* deberá estar en *True*.

Left: Distancia que existe entre el limite izquierdo de un componente y el limite izquierdo del formulario.

Top: Distancia que existe entre el limite superior del objeto y el limite superior del formulario.

Visible: Es una propiedad Booleana, que indica si el componente será mostrado o no.

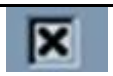
Width: Ancho del objeto dado en pixeles.

Text: En el caso del componente TextEdit, indica el texto que contendrá en su interior.

Lines: En el componente Memo, es un array que contiene todas las líneas introducidas por el usuario.

## COMPONENTE BÁSICOS

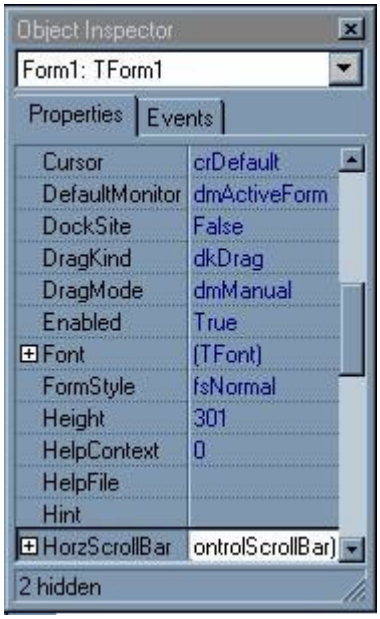
Los componentes básicos, principalmente se encuentran en la pestaña Standard, y son encontrados en todos los programas, pues permiten llevar un control de la ejecución del mismo programa. Los componentes básicos son:

| Botón | Control | Nombre | Descripción |
|---|---|---|---|
|  | Main Menu | Menú Principal | Permite insertar menús dentro de la aplicación. En este menú se puede poner el control de la aplicación. |
|  | Label | Etiqueta de Texto | Permite colocar texto en los formularios. Se utilizan para indicar al usuario lo que debe hacer. |
|  | Edit | Caja de Texto | Permite al usuario introducir datos para ser tratados. |
|  | Memo | Cuadro de líneas | Permite al usuario introducir grandes cantidades de texto en múltiples líneas. |
|  | Button | Botón de orden | Son los que permiten el control del programa. |
|  | CheckBox | Casilla de Verificación | Estas permiten seleccionar varios elementos de un listado, la selección puede ser desde 0 hasta el número de elementos que se tengan. |
|  | RadioButton | Casilla de Opción | Permite seleccionar uno solo de los elementos de una lista. |
|  | ListBox | Cuadro de lista | Muestra una lista con elementos que se pueden seleccionar. |
|  | ComboBox | Cuadro Combinado | Muestra el objeto seleccionado de una lista. Combina una |

| | | | |
|---|---|---|---|
| | | | caja de texto con una persiana. |
|  | ScrollBar | Barra de Desplazamiento | Es la representación visual del valor actual de una variable. |
|  | GroupBox | Marco | Permite agrupar opciones o botones. |
|  | RadioGroup | Marco de selección | Permite agrupar botones de opción de un modo lógico en el formulario. |
|  | Panel | Panel | Es un subformulario dentro del formulario. |
|  | BitBtn | Botón Gráfico | Son botones normales, a diferencia que permiten la introducción de un pequeño gráfico. |
|  | SpeedButton | Botón rápido | Son los que se localizan debajo del menú principal y realizan instrucciones comunes. |
|  | Image | Imagen | Permite insertar imágenes en el formulario. |
|  | StaticText | Texto estático | Es similar a una caja de texto, a diferencia que si el texto no cabe en la caja, se pasa a la siguiente línea. |
|  | Shape | Forma | Dibuja una figura en el formulario. |

**EL INSPECTOR DE OBJETOS**

Los objetos, tienen propiedades y estas reflejan cualidades del mismo, como ancho, color, altura, etc. Pero ¿Cómo se cambian dichas propiedades? Una forma de hacerlo es mediante el inspector de objetos.
El inspector de objetos es la ventanita que se encuentra en l aparte izquierda de la pantalla. Esta es indispensable para la creación de las aplicaciones. Se compone de 2 partes: La parte superior que permite seleccionar el objeto que se desea modificar, y la parte inferior, que es donde se modifican las propiedades de los objetos.




# FORMULARIOS

Un formulario es la interfaz que presentan las aplicaciones de Windows. El formulario, es la caja de color gris que tiene una barrita de color azul en la parte superior. Todos los lenguajes de programación que sean para Windows, o estén orientados a objetos, crean de modo automático la ventana y es ese el punto de partida de toda aplicación.

## CARACTERISTICAS DE UN FORMULARIO

Un formulario, no es mas que un objeto mas dentro de la programación, por lo tanto, cuenta también con métodos, propiedades y eventos. Los formularios poseen muchas propiedades, algunas de las mas importantes son:

*ActiveControl*: En este se pone el nombre del objeto que tendrá el enfoque cuando se ejecute la aplicación, de no poner nada, tendrá el enfoque el objeto que se inserto primero.

*AutoScrooll*: Si tiene el valor True, el formulario presentará barras de desplazamiento que permitan acceder a todas las partes del formulario.

*AutoSize*: Si tiene el valor True, el ancho y alto del formulario se ajustará automáticamente para que ocupe el menor espacio posible conteniendo a todos los componentes.

*BorderIcons*: Propiedad compuesta que determina si los botones de maximizar, minimizar, salir... estarán visibles en el formulario.

*BorderStyle*: Determina como será el borde del formulario.

*BorderWidth*: Contiene el tamaño en pixeles del borde de la ventana.

*Caption*: Texto que aparece en la parte superior del formulario.

*Color*: Color de fondo del formulario.

*Name*: Nombre del formulario en código.

*Position:* Posición que ocupará en la pantalla el formulario.

**MÉTODOS DE UN FORMULARIO**

Los métodos, son las acciones que se pueden realizar sobre un formulario, entre las mas comunes encontramos a la siguientes:

*Close*: Invocando a este procedimiento, el formulario se cierra.

*Show*: Este procedimiento hace que el formulario aparezca en pantalla.

*ShowModal*: Esta función hace aparecer el formulario, y le da el control de la aplicación, para que el usuario pueda acceder a sus botones, cuadros de texto. Mientras el usuario no cierre este formulario, no puede seguir con la aplicación.

*Hide:* Oculta el formulario. Es el procedimiento contrario a *Show*.

| Valor | Descripción |
|---|---|
| mrNone | El formulario terminó pero no se pulso ningún botón. |
| mrOk | El usuario pulsó el botón Ok. |
| mrCancel | El usuario pulsó el botón Cancel. |
| mrAbort | El usuario pulsó el botón Abort. |
| mrRetry | El usuario pulsó el botón Retry (Reintentar). |
| mrIgnore | El usuario pulsó el botón Ignore (Ignorar) |
| mrYes | El usuario pulsó Yes. |

| | |
|---|---|
| mrNo | El usuario pulsó No. |
| mrAll | El usuario pulsó otro botón o pulsó varios. |

## PRACTICA #1

Realizar una aplicación de "Hola mundo".

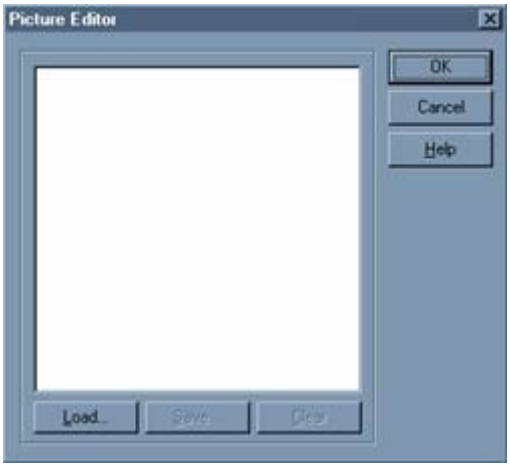
Cambiando el tamaño del formulario.
Cambiando el Titulo del formulario.
Insertando un botón que tenga "Hola" en la propiedad Caption.

*Practica #1*:Cuando se desea colocar el código al botón, es necesario seleccionar el evento en el que deberá aparecer en la persiana *Events* del inspector de objetos.
El código que se colocó en el evento "*On Click*" es el siguiente:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
close; {Esta orden cierra el formulario}
end;
```

Una vez creada la primera aplicación, esta tomó el siguiente aspecto:


## PONER ICONO AL FORMULARIO



En Windows, cada formulario puede dotarse de un icono propio, que será visualizado en la propia aplicación. El icono que se debe poner, deberá de identificar a la aplicación. Para asignar un icono al formulario, únicamente es necesario pulsar sobre la propiedad *Icon* del formulario. Esta propiedad envía a una ventana algo compleja que permite la visualización del icono seleccionado.

Esta ventana dispone de 6 botones y de un área blanca. Los botones son los siguientes:

*Help*: Sirve para mostrar una ventana con ayuda acerca de este editor.

*OK*: Se utiliza para indicar a Delphi que el icono seleccionado es el que se desea utilizar.

*Cancelar*: Se utiliza para ignorar la selección realizada.

*Load:* Sirve para buscar un icono en el disco. Presenta el cuadro abrir.

*Save*: Sirve para guardar en el disco el icono que se ha seleccionado. Se utiliza cuando se desea cambiar de nombre el icono.

*Clear*: Se utiliza para borrar el icono y no asignar ninguno al formulario.

### PONER ICONO AL PROYECTO

Cada proyecto o programa cuenta con su icono correspondiente, el cual puede ser distinto al de los formularios, este debe ser especifico y relacionado al funcionamiento del programa.
Cada proyecto tiene por defecto un icono de una torre con un rayo y un circulo verde. Sin embargo este se puede modificar dando click en el menú *Project* seguido de la instrucción *Options* y en la ventana que se presenta, se selecciona la pestaña *Application* y se pulsa el botón *Load Icon*.

# PROYECTOS

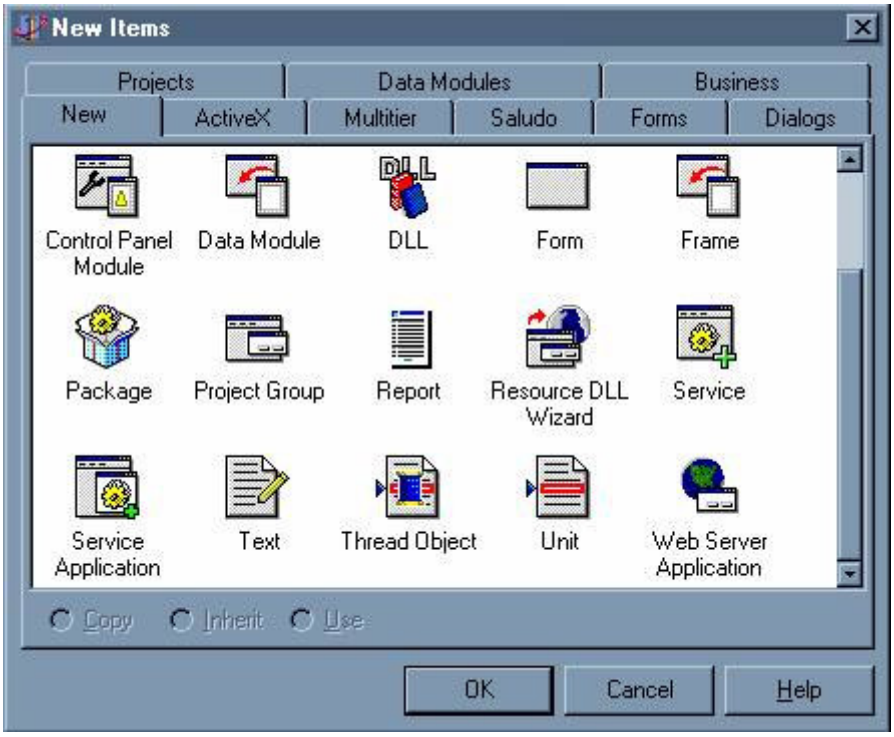
### ¿QUÉ ES UN PROYECTO?

Los programas se componen de instrucciones, las cuales se agrupan en bloques y en funciones. Además existe un nivel superior de agrupación de código: Las unidades y los proyectos.

El lenguaje Delphi, es un nivel estructurado de Pascal, conocido como *Object Pascal,* que nos permite la utilización del mismo código tantas veces como sea necesario, sin la necesidad de repetir todas las sentencias.
En este caso, podemos realizar una de 2 alternativas: repetir en la nueva aplicación las mismas sentencias e instrucciones, lo que supone mas trabajo; o bien, poner los módulos compartidos en un fichero aparte, de modo que todas las aplicaciones puedan utilizar estas rutinas sin necesidad de repetir todo el código.
A estos ficheros se les llama *unidades*, de este modo, una unidad es una colección de funciones y procedimientos que tienen una característica común en cuanto a las tareas que realizan. Por ejemplo, si deseamos generar un programa que realice algunas funciones matemáticas, podríamos tener en una unidad donde se almacenen las instrucciones para realizar los cálculos aritméticos, otra que sea la representación visual de los resultados, otra para que interactue con el usuario, etc.
En Delphi, los procedimientos y funciones que forman una unidad, se almacenan en un fichero independiente, de manera que cada una de ellas se encuentra en su propio archivo.
Al construir una aplicación, debemos especificar mediante indicaciones, que unidades utilizaremos. Estas indicaciones es lo que se llama *proyecto*.
En realidad, un proyecto es una lista de los archivos que utilizaremos para construir una aplicación.

## CREACIÓN DE UNIDADES

Cuando se desee crear una unidad, será necesario seleccionar el icono *Unit* en la ventana que aparece al elegir la opción New del menú File.




En una unidad, se deben tener 2 partes claramente definidas:

> *La Parte de Interface*: en la que únicamente se definen las cabeceras de funciones, procedimientos y las declaraciones de los tipos y variables que posee la unidad. Esta parte va precedida por la palabra reservada *Interface*. Su utilidad es permitir que las aplicaciones sepan que contiene la unidad. Es la parte publica y a la que se puede tener acceso.
> *La Parte de Implementación*: Es en la que se escribe el código de las funciones y procedimientos que se desea realice la unidad. Esta parte va precedida de la palabra reservada *Implementation*, y termina con la palabra seguida de un punto. Es la parte privada, a la cual no podemos acceder.

Ejemplo de una unidad que muestre en pantalla un mensaje y pregunte el nombre mediante una caja de mensaje:

```
(*************************************
     Ejemplo de declaración de una unidad
************************************* *)
Unit Mensajes; {El nombre del fichero, pero sin .PAS}
Interface
//Solo cabeceras de las funciones y proc.
```

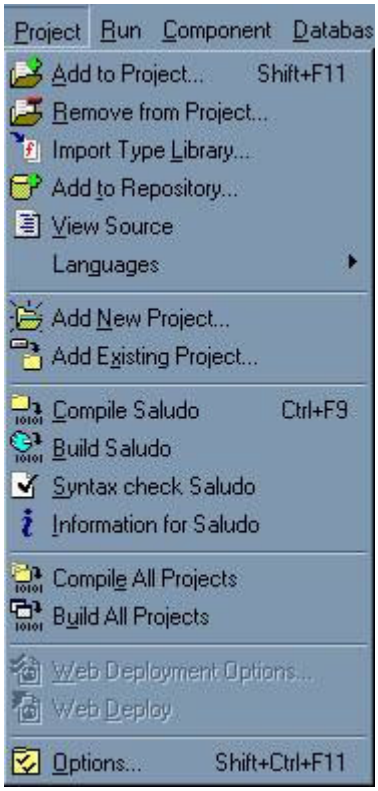
```
procedure Di (mensaje:String);
function PreguntaNombre:String;
Implementation
          //Implementación de las rutinas
procedure Di(mensaje:String);
Begin  //Muestra en pantalla el mensaje que se pasa por
parámetro
Showmessage(mensaje);
End;
Function PreguntaNombre:String;
Begin
PreguntaNombre := Inputbox('Identificación',
                    'Teclee su nombre',
                    '(Desconocido)');
End;
end.
```

Cuando deseemos utilizar este código desde el programa principal, deberemos poner al principio de este la palabra reservada *Uses* seguido de los nombres de unidades que deseamos utilizar, separados por comas. Ejemplo:

```
Program Ejemplo;
Uses Mensajes, Calculos;
Var  a : Word;
Begin
Di ('Hola');
a := logaritmo(10);
End;
```

En este ejemplo, se emplea un procedimiento (Di) que se encuentra en la unidad Mensajes. Por ello, antes de utilizar dicho procedimiento, se debe declarar el uso de dicha unidad, mediante la palabra *Uses* seguida de los nombres de unidades que se emplearan separadas por comas.

## EL MENÚ PROJECT




Este menú contiene la mayoria de las acciones que se pueden realizar sobre un proyecto.

*Add To Project*: Permite incluir una unidad ya existente al proyecto actual.
*Remove From Project*: Borra una unidad de las existentes en el proyecto.
*Import Type Library*: Permite incluir en nuestra aplicación librerías que son utilizadas por otras aplicaciones.
*Add To Repostory*: Permite exportar nuestras librerías para que puedan ser utilizadas por otras aplicaciones.
*View Source*: Muestra el código fuente del proyecto actual. Este código no es mas que una lista de las unidades del proyecto.
*Add New Project*: Permite crear un nuevo proyecto sin cerrar el abierto.
*Add Existing Project*: Permite abrir un proyecto existente sin cerrar los actuales.
*Compile Project*: Verifica que las instrucciones sean correctas al transformarlas a un lenguaje comprensible por la maquina.

*Build Project*: Crea un proyecto ejecutable que puede ser utilizado por el usuario.
*Syntaxis Check Project:* Verifica la sintaxis del código introducido.
*Information for Project:* Muestra una ventana con los datos importantes del proyecto, como número de líneas, tamaño del ejecutable, etc.
*Compile All Project:* Verifica que el código de todas los proyectos abiertos sea comprensible por la maquina.
*Build All Project:* Crea un archivo ejecutable por cada proyecto que se encuentre abierto.
*Options:* Presenta una ventana que permite configurar el proyecto: Icono de la aplicación, formularios disponibles, etc.

## OPERACIONES CON PROYECTOS

Sobre un proyecto, se pueden realizar las siguientes operaciones:

***Crear un nuevo proyecto***: Para crear un nuevo proyecto, únicamente será necesario dar click en el menú *File* seguido de la orden *New*. Este aparecerá con una única unidad, la cual podrá ser modificada mediante el formulario al que esta relacionada. Es recomendable que al crear un proyecto, se guarde inmediatamente.

***Abrir un Proyecto Existente***: Para seguir trabajando sobre un proyecto grabado previamente, se tiene varias opciones:
Utilizar la opción *Open* del menú *File*.
Utilizar la opción *ReOpen*, del menú *File*.

***Añadir una unidad nueva al proyecto***: Si lo que deseamos es crear una unidad nueva en el proyecto, en la que escribamos el código de un nuevo modulo, tenemos 2 opciones:
Si deseamos insertar una unidad que tenga relacionado un formulario, seleccionamos la opción *New Form* del menú *File*.
Si deseamos generas una unidad que no tenga relacionado ningún formulario, pues realizara funciones especificas, entonces se deberá dar click en la instrucción *New* del menú *File*, se mostrará una ventana en donde debemos seleccionar lo que se desea crear.

***Añadir una unidad existente:*** Si deseamos agregar una unidad existente al proyecto que se está elaborando, es necesario seleccionar la opción *Add To Project* del menú *Project*. En la ventana que se presenta, es necesario seleccionar el archivo que se desea agregar. Para quitar la unidad del proyecto, es necesario dar click en la opción *Remove From Project* del menú *Project*.

***Ver las unidades del proyecto***: Mediante este se puede saber que unidades se encuentran en el proyecto y que unidades aún no son agregadas en el mismo. Para utilizar esta orden, es necesario dar click en el menú *View* y seleccionar la instrucción *Units*.

***Cerrar un proyecto***: Cuando se desea cerrar un proyecto, ya sea porque no se desea seguir trabajando con Delphi o porque se trabajará sobre otro proyecto, es necesario dar click en la orden *Close All* del menú *File*.

*Código de un proyecto*: El código del proyecto puede ser visualizado mediante esta orden. Este aparecerá precedido de la palabra reservada *Program* y a continuación la lista que asocia cada unidad con el nombre del fichero en el que se encuentra.

# DEPURACIÓN Y CONTROL DE ERRORES

Cuando se construyen las aplicaciones, existen muchos fallos que pueden hacer que una aplicación no funcione como debe. Esto es algo normal y ocurre en las primeras pruebas realizadas sobre un programa.
El proceso de generación del código ejecutable pasa por una serie de estados con nombres específicos, que resulta conveniente conocer antes de continuar el resto de la aplicación:

> *Código Fuente*: Generalmente, se denomina de este modo al fichero que contiene las instrucciones en Pascal y que implementa las funciones de nuestra aplicación.
> *Código Ejecutable*: Se denomina de este fichero que contiene el programa escrito en el lenguaje base que entiende la maquina.
> *Compilación*: Consiste en el análisis y traducción del código fuente a código ejecutable. En Delphi consiste en verificar la sintaxis de las instrucciones y checar que se encuentre correctamente escrito.
> *Depuración*: Es el proceso de localización y eliminación de errores. Consiste en ejecutar el programa parte por parte para verificar errores.
> *Optimización*: Proceso para reorganizar el código ejecutable, de modo que se ejecute lo mas rápidamente posible y son el menor consumo de memoria.

## EL MENÚ PROJECT

Este menú contiene las acciones que se pueden realizar para checar un proyecto. Dentro de el existen 3 instrucciones básicas para la depuración y compilación del proyecto.

*Compile Project*: Realiza la compilación de las unidades del proyecto. Delphi checa únicamente las unidades que han sido modificadas desde la ultima compilación ahorrando así bastante tiempo.

*Build Project*: Similar al anterior, a diferencia de que checa todo el proyecto, aun las unidades que no han sido modificadas.

*Syntaxis Check*: Verifica que el código introducido este correctamente escrito en Pascal, es una precompilación.

## OPCIONES DEL COMPILADOR

La ultima opción del menú *Project* es *Options* que, entre otras cosas permite seleccionar como queremos realizar la compilación.

Esta ventana está dividida en 5 secciones:

*Code Genration*: Permite especificar como deseamos el código ejecutable del programa.
*Runtime Error*: Permite incluir rutinas que controlen errores como fallos de rangos.
*Syntaxis Options*: Especifica parámetros de compilación.
*Debugging*: Aquí se indican los parametros de compilación, deben estar siempre activados.
*Messages*: Indica a Delphi si debe mostrar mensajes cuando se ocasione un error.

## EL MENÚ RUN

Contiene las instrucciones que permiten ejecutar el programa y realizar depuraciones. Posee las siguientes opciones:

*Run*: Realiza una compilación y ejecuta el proyecto.
*Parameters*: Las aplicaciones pueden recibir parámetros a la hora de ser invocadas.
*Register ActiveX, Unregister ActiveX*: Estas opciones sirven para dar de alta y de baja respectivamente aplicaciones ActiveX.
*Step Over*: Permite realizar una ejecución paso a paso. Se usa para depuraciones.
*Trace Into*: Permite realizar una ejecución instrucción por instrucción.
*Trace To Next Source Line*: Ejecuta el programa y lo detiene cuando se llega a la siguiente linea de código.

*Run to Cursor*: Delphi ejecuta la aplicación hasta la línea donde se encuentra el cursor.
*Show Execution Point*: Sitúa el cursor en la línea que se esta ejecutando.
*Program Pause*: Detiene de forma temporal el programa.
*Program Reset*: Finaliza el programa en el punto en que se encuentre.
*Inpect*: Abre la ventana de inspección para ver valores de variables.
*Evaluate/Modify*: Permite analizar y cambiar variables, y comprobar el resultado de operaciones booleanas.
*Add Watch*: Existe una ventana llamada Watch que permite ver en todo momento el valor de una variable.
*Add Breakpoint*: Esto es que marca previamente el lugar donde deberá de pausar el programa.

## LOS ERRORES

En Delphi se generan 2 tipos de errores:

*Errores de compilación*: Estos errores son producidos al compilar el código fuente. Son causados por errores al escribir una función, no declarar variables, falta de punto y coma... Delphi no continua ejecutando el programa hasta que estos son corregidos.

*Errores de Ejecución*: Se producen cuando se ejecuta un programa, son debidos a fallos al abrir ficheros, bucles infinitos. Son difíciles de encontrar y corregir.

Para corregir estos errores Delphi ofrece herramientas de corrección.

## ERRORES DE COMPILACIÓN

Este tipo de errores, aparece cuando el código se encuentra mal escrito o no se siguen las reglas de sintaxis: Son los mas fáciles de localizar. Delphi, de modo automático nos señala el lugar del error y nos dice que tipo de error se ha cometido.

**Ventana de compilación:** En esta, se nos presenta el mensaje indicándonos el lugar del error, además del tipo de error. Los mensajes que en ella aparecen, se componen de 2 partes:

*Nombre de unidad y número de línea*: Presenta el nombre de la unidad en la que fue encontrado el error y el número de línea en el que se encuentra.
*Explicación*: Es un pequeño texto explicativo sobre la causa que ha provocado este error.

Se debe hacer doble click sobre e mensaje de error para que automáticamente nos lleve a la línea de código que se debe corregir.

Los errores de compilación mas comunes son:

*Missing operator or semicolon*: Este error se presenta cuando se olvida poner punto y coma al final de una línea de código. En Pascal, cada línea de código, debe llevar punto y coma. Un ejemplo de este error es:

```
Procedure Form.Ejemplo;
Begin
Showmessage('Hola')       //aquí no hay punto y coma
Showmessage('Hola otra vez');    //← aquí da el
error
End;
```

Al compilarlo, aparecería el siguiente mensaje:

```
(Error)Ejemplo.pas(40): Missing operator or
semicolon in Form.Ejemplo
```

*Incompatible types: 'Tipo1' and 'Tipo2'*: Se esta tratando de asignar a una variable un valor que no puede almacenar en ella. Ejemplo:

```
Procedure Form.Ejemplo;
Var  v:byte;
Begin
v := 'h';    //v solo puede almacenar números, no
caracteres
Showmessage('Hola');
End;
```

Al compilar, aparecería el mensaje:

```
(Error)Ejemplo.pas(50): Incompatible types: 'Byte'
and 'Char'
```

*Undeclared Indentifier: 'x'*: Este error aparece cuando al compilar se ha encontrado una variable que no ha sido declarada. Puede ser causa de que se encuentre mal escrita dicha variable. Ejemplo:

```
Procedure Form.Ejemplo;
Var MiNombre: string;
     Direccion:sting;    //"string" mal escrito
Begin
MiNombe := 'David Osornio Fernández';  //"MiNombre"
mal escrito
```

```
Telefono := '755-01-64';      //Variable no definida
End;
```

Al compilar aparecerían los siguientes mensajes:

```
(Error)Ejemplo.pas(40): Undeclared identifier:
'sting'
(Error)Ejemplo.pas(42): Undeclared identifier:
'MiNombe'
(Error)Ejemplo.pas(43): Undeclared identifier:
'Telefono'
```

*';' expected but found PROCEDURE*: Este error aparece cuando se declara una función y se olvida de poner punto y coma al final del end del procedimiento. Ejemplo:

```
Procedure Form.Ejemplo;
Begin
Showmessage('Hola');
End          //← aquí falta el unto y coma

Procedure Form.Ejemplo2;    //← aquí da el error
Begin
Showmessage('Hola otra vez');
End;
```

Al compilarlo aparecería el mensaje:

```
(Error)Ejemplo.pas(45): ';' expected but found
PROCEDURE
```

*';' expected but '.' found*: Este error aparece sobre todo cuando se esta definiendo una clase y se ha escrito mal el nombre. Ejemplo:

```
//estamos definiendo los métodos del objeto
MiObjeto
Procedure MiObjeto.Ejemplo2;    //← nombre de
clase mal escrito
Begin
Showmessage('Hola');
End;
```

Al compilarlo aparecería el mensaje:

```
(Error)Ejemplo.pas(50): ';' expected but ',' found
```

*File not found: 'x'* : Este error aparece cuando en *uses* se escribe mal el nombre de un archivo o unidad. O se coloca el nombre de un archivo inexistente. Ejemplo:

```
//tenemos las unidades unidad1 y unidad2
Uses Crt, unidad1, uidad2, unidad3;
     //← Unidad2 mal escrita
     // ← Unidad3 no existe
Procedure MiObjeto.Ejemplo2;
Begin
...
```

Al compilarlo aparecería el mensaje:

```
(Error)Ejemplo.pas(10): File not found 'uidad2';
(Error)Ejemplo.pas(10): File not found 'unidad3';
```

*Declaration expected but end file found: 'x'* : Indica que al final de la unidad se ha olvidado poner un *END* seguido de punto.

## Los avisos (Hints)

Además de los mensajes de error, Delphi nos muestra en la ventana de compilación mensajes que nos pueden ayudar a mejorar nuestro código. Podríamos decir que son advertencias de situaciones extrañas que Delphi ha detectado en nuestro programa.

*Variable 'X' is declared but never used*: Este aviso aparece cuando dentro de un procedimiento o función se ha declarado una variable que no se usa.

```
Procedure Form.Ejemplo;
Var v : byte;       //← Definimos v, pero no la
usamos
Begin
Showmessage('Hola');
End;
```

Al compilarlo aparecería el mensaje:

```
(Hint)Ejemplo.pas(40): Variable 'v' is declared but
never used in Form.Ejemplo
```

*Value assigned to 'X'*: Este aviso aparece cuando dentro de un procedimiento o función se ha declarado una variable, y

después le dimos un valor, sin embargo jamas la volvimos a utilizar.

```
Procedure Form.Ejemplo;
Var v : byte;
Begin
v := 1;  //damos un valor pero no volvemos a usar v
Showmessage('Hola');
End;
```

Al compilarlo aparecería el mensaje:

```
(Hint)Ejemplo.pas(40): Value assigned to 'v' never
used in Form.Ejemplo
```

*Private symbol 'x' declared but never used*: Este aviso aparece cuando dentro de la parte *private* de un formulario se declara una variable que no se usa en toda la unidad.

## ERRORES EN TIEMPO DE EJECUCIÓN

Este tipo de errores, son muy comunes en las primeras partes de prueba de un programa, y aunque son molestos nos ayudan a definir con exactitud las acciones de dicho programa. Estos son ocasionados por accesos a ficheros que no se han abierto, utilización de formularios que no han sido creados, operaciones aritméticas incorrectas, etc.

## ERRORES DE FUNCIONAMIENTO

Estos errores son los mas difíciles de detectar, pues el programa funciona correctamente, sin embargo los resultados que arroja son incorrectos y deben ser analizados. Siempre se debe tener en cuenta los elementos que se manejan y como se manejan, estos errores son ocasionados por mala lógica del programador.

## EJECUCIÓN PASO A PASO

Esta herramienta permite ejecutar el código de nuestra aplicación línea a línea, pudiendo ver en todo momento el estado de nuestro programa: como valores de variables, procedimiento que se está ejecutando, resultado de comparaciones booleanas.
Este tipo de ejecución permite al sistema la ejecución del programa paso a paso, y línea por línea.

## LA VENTANA WATCH

Esta se utiliza para visualizar constantemente el valor de una variable. Para visualizarla se debe dar click en el menú *Run* seguido de *Add Watch*. En el apartado *Expression* se debe colocar el nombre de la variable que se desea analizar y pulsar enter.

## LA VENTANA EVALUATE/MODIFY

Esta ventana es una especialización de la anterior, en la que no solo se nos permite ver el valor de la variable, sino que además se nos permite modificar dicho valor. También permite analizar las expresiones booleanas como el *if*.

Como puedes observar, tiene tres partes:

*Expression*: En este cuadro de texto se introduce la variable que se desea analizar.
*Result*: En este apartado será mostrado el resultado de la expresión que se ha introducido en el cuadro anterior.
*New Value*: Aquí pondremos el nuevo valor que se le asignara a la variable.

## EVITAR ERRORES

En algunas ocasiones será imposible el corregir los errores de un programa, debido a que estos se ocasionan por un error en los datos introducidos por el usuario, por ejemplo:

*Hacer divisiones entre 0*: Si una aplicación intenta efectuar una división entre 0, será detenida automáticamente.
*Acceder a ficheros que no existen*: Esto genera un error pues se trata de rastrear un archivo que no existe en la posición designada.
Evitar hacer bucles infinitos*:* Este esta formado por un bucle que jamas termina, lo que ocasiona es que el sistema colapse.

## TRY

Para evitar el uso del if y que se pudiera ocacionar un error de código, se utiliza la estructura *Try...Except*. Lo que realiza este bloque es detener la ejecutar la ejecución de alguna instrucción, si esta ocasiona error. La sintaxis es:

```
Try
Bloque "peligroso
Except
```

```
Bloque alternativo
End;
```

De este modo, si en el bloque peligroso se produce un error o fallo, automáticamente se ejecuta la instrucción alternativa. Por ejemplo, para evitar una división entre 0 se pondria:

```
Try
a := b/c;
Except
Showmessage('error al dividir');
End;
```

# TRABAJAR CON GRAFICOS

Los gráficos permiten agregar un toque profesional a nuestras aplicaciones. Y de este modo conseguir unas aplicaciones mucho mas atractivas para el usuario.

## GRAFICOS EN DELPHI

En Delphi es posible incluir gráficos mediante 6 distintos objetos:

*Image* en la página *Additional*: Este objeto permite insertar en los formulario imágenes contenidas en ficheros con extensión JPG, JPEG, BMP, ICO, EWF o WMF.
*Shape* en la página *Additional*: Permite generar figuras geométricas estáticas en el formulario.
*Bevel* en la página *Additional*: Permite dar un efecto tridimensional a las aplicaciones.
*Canvas* de *Image*: Permite crear figuras geométricas de forma dinámica en el formulario.
*PaintBox* de la página *System*: es una mejora del componente *Image*, pues utiliza menos memoria.

## GRAFICOS CON IMAGE

Para utilizar el objeto Image, únicamente es necesario insertarlo en el formulario y en la propiedad Picture, buscar la imagen que se desea colocar. Este objeto tiene 2 propiedades importantes, que son:

*Picture*: Que hace mención al archivo que será visualizado.
*Autosize*: Si se encuentra en True, si la imagen es mayor o menor, el objeto se acoplará al tamaño de la imagen, en caso contrario, se deberá acoplar al tamaño de dicho objeto.

## GRAFICOS CON SHAPE

Este componente, permite el dibujo de figuras en el formulario, este tipo de objeto puede ser cambiado en cualquier momento, ya sea en tiempo de diseño o en tiempo de ejecución. Este objeto es

rara vez usado, pues no recibe ningún tipo de datos del usuario, ni muestra datos o resultados.

Los valores de la propiedad *Shape* de dicho objeto, demarcarán el tipo de figura geométrica que se presentará:

| Constante | Figura Geométrica |
|---|---|
| StCircle | Círculo |
| StEllipse | Elipse |
| StRectangle | Rectángulo |
| StRoundSquare | Cuadrado con las esquinas redondeadas |
| StRoundRectangle | Rectángulo con las esquinas redondeadas |
| StSquare | Cuadrado |

Así pues *Shape* permite dibujar figuras geométricas, las cuales pueden tener propiedades como color de contorno, color de relleno, estilo de trazo, etc. Todas estas cualidades se encuentran en la propiedad *Pen*. Las subpropiedades que se encuentran en esta propiedad son:

*Color*: Color de contorno de la figura. Contiene una lista con los colores disponibles.
*Mode*: Es el modo como se dibuja el contorno, esta propiedad es algo complicada.
*Width*: Ancho del contorno.
*Style*: Tipo de trazo en el contorno, punteado, continuo, etc. En esta se encuentran los siguientes valores:

| Valor | Significado |
|---|---|
| PsSolid | Trazo continuo |
| PsDash | Trazo discontinuo |
| PsDot | Trazo punteado |
| PsDashDot | Trazo formado por una línea y un punto |
| PsDashDotDot | Trazo formado por una línea y dos puntos |
| PsClear | Sin borde |

La propiedad *Pen* se refiere al contorno de dicho objeto, y la propiedad *Brush* se refiere al relleno de dicho objeto. Las subpropiedades de la propiedad *Brush* son:

*Color*: Color de relleno de la figura.
*Style*: estilo de relleno de la figura. Esta subpropiedad tiene distintos valores, los cuales son:

| Valor | Significado |
|---|---|
| bsDiagonal | Relleno de líneas diagonales a la izquierda. |
| bsClear | Sin relleno. |
| bsCross | Relleno a cuadrículas. |
| bsDiagCross | Relleno a cuadrículas diagonales. |
| bsFDiagonal | Relleno de líneas diagonales a la derecha. |
| bsSolid | Con relleno sólido. |
| bsHorizontal | Relleno de líneas horizontales. |
| bsVertical | Relleno de líneas verticales. |

## EL COMPONENTE BEVEL

Este componente no es en si un objeto gráfico o colocador de imágenes, sino que únicamente da un efecto de 3D a nuestra aplicación. Tiene 2 propiedades principales, las cuales son:

*Shape*: la cual marca el tipo de forma que tomará dicho objeto. Esta puede ser:

| Valor | Significado |
|---|---|
| bsBox | Forma de rectángulo 3D (sobresale del formulario). |
| bsFrame | Forma un rectángulo incrustado en el formulario. |
| bsTopLine | Forma una línea horizontal superior incrustada. |
| bsBottonLine | Forma una línea horizontal inferior incrustada. |
| bsLeftLine | Forma una línea vertical izquierda incrustada. |
| bsRightLine | Forma una línea vertical derecha incrustada. |

*Style:* Que puede tomar 2 distintos valores:

| Valor | Significado |
|---|---|
| bsLowered | Las formas aparecen incrustadas en el formulario. |
| bsRaised | Las formas aparecen sobresalidas del formulario. |

## GRAFICOS CON CANVAS

La propiedad Canvas, permite que el programa por si mismo dibuje las figuras geométricas de modo automático, sin necesidad de dibujarlas por si mismo en tiempo de diseño. Este no es en si un componente de Delphi, sino que es una propiedad de los objetos. Tanto el formulario, como el componente *image* tienen esta propiedad. La propiedad Canvas tiene 5 subpropiedades importantes que son:

| Propiedad | Descripción |
|---|---|
| Brush | Brocha, se utiliza para especificar el relleno de las figuras. |
| Font | Permite indicar las propiedades del texto gráfico. |
| Pen | Permite determinar el tipo, color, estilo, etc. De la línea. |
| PenPos | Punto actual del lápiz gráfico. |
| Pixels | Matriz con los puntos gráficos del componente. |

Además, Canvas permite utilizar muchos métodos de dibujo sobre si mismo. Los más importantes son:

| Método | Descripción | Ejemplo |
|---|---|---|
| Ellipse | Dibuja una elipse. Recibe como parámetros, las coordenadas del rectángulo que bordea la elipse. | Ellipse(10,20,100,150); |
| Arc | Dibuja una porción de elipse. Recibe como parámetros las coordenadas del rectángulo que bordea la elipse, y las coordenadas de los puntos iniciales y finales del segmento de la elipse. | Arc(10,20,100,150,10,20,200,200); |
| MoveTo | Define la nueva posición actual. | MoveTo(5,5); |
| LineTo | Dibuja una línea que va desde la posición actual hasta las coordenadas indicadas. | LineTo(100,100); |
| Rectangle | Dibuja un rectángulo, toma como | Rectangle(10,10,80,50); |

| | | |
|---|---|---|
| | parámetros las coordenadas de las esquinas superior izquierda e inferior derecha. | |
| TextOut | Imprime texto en las coordenadas indicadas. | TextOut(10,10,'Hola'); |

**Practica De Canvas**

En un proyecto nuevo en blanco, se deberá realizar la siguiente secuencia de pasos.

Se insertará un componente *image* en el formulario.
Se modificarán las propiedades del objeto.

**Left:** *350*
**Name:** *Imagen*
**Top:** *50*
**Height:** *350*

En el inspector de objetos, en la ficha *events*, se deberá seleccionar el evento *OnPaint* del objeto formulario.
En este, se deberá colocar el siguiente código:

```
Procedure Tformulario.FormPaint(Sender:Tobject);
Var x:word;
Begin
Imagen.canvas.pen.color := clBlue;        {Color de
borde azul}
Imagen.canvas.ellipse(120,12,220,220);    {elipse}
Imagen.canvas.pen.color := clRed;         {Color de
borde rojo}
Imagen.canvas.rectangle(200,150,280,300);
{Rectángulo}
Imagen.canvas.pen.color := clPurple; {Color de
borde púrpura}
Imagen.canvas.brush.color := clPurple;    {Color de
relleno púrpura}
Imagen.canvas.brush.style := bsFDiagonal; {estilo
de relleno líneas diagonales}
Imagen.canvas.ellipse(300,100,600,150);   {Elipse}
```

```
Imagen.canvas.pen.width := 3;          {Figura de
borde ancho}
Imagen.canvas.brush.color := clRed;   {Color de
relleno rojo}
Imagen.canvas.brush.style := bsDiagCross;
{diagonales cruzadas}
Imagen.canvas.pen.color := clYellow; {Color de
borde amarillo}
Imagen.canvas.ellipse(10,200,150,300);    {Elipse}
Imagen.canvas.brush.style := bsSolid;     {Relleno
sólido}
Imagen.canvas.pen.color := clGreen;   {color de
borde verde}
Imagen.canvas.pen.width := 1;          {Borde fino}
For x:=1 to 100 do
begin
{línea que va de (x*10,10) a (300-x*10,80)}
imagen.canvas.MoveTo (x*10,10);
imagen.canvas.LineTo (300-x*10,80);
end;
Imagen.canvas.brush.color := clNone; {Sin color de
relleno}
imagen.canvas.Font.color := clBlack; {Color de
texto negro}
imagen.canvas.Font.size := 16;         {Tamaño de
texto 16}
imagen.canvas.TextOut(100,300,'Hola esta es una
demostración de Canvas');
End;
```

En este ejemplo, una porción de figura de la elipse púrpura quedara fuera del objeto *image*, esta porción de figura no será mostrada.

Para utilizar la propiedad Canvas del formulario, se utilizan las mismas instrucciones, a diferencia de que no se hace referencia a ningún objeto al principio de cada línea de código. Por ejemplo, se sustituiría la línea.

```
Image1.canvas.ellipse(20,50,10,20)
```

Por la línea:

```
Canvas.ellipse(20,50,10,20)
```

**EL COMPONENTE Bitbtn**

En si no es tomado como un componente gráfico, sino que es un objeto que dibuja un componente de botón, que permite colocar

un objeto o imagen dentro de si mismo, para poder representar mas fácilmente la acción que este realiza.

Las propiedades principales que este objeto presenta son:

| Propiedad | Descripción |
|---|---|
| Caption | Es el texto dentro del botón. |
| Glyph | Gráfico que se presenta en el botón. |
| Margin | Separación en pixeles entre el gráfico y el texto. |
| Layout | Posición del gráfico en el botón. |
| Spacing | Separación en pixeles entre el borde del botón y el texto. |

Para facilitar el trabajo, Delphi tiene botones predeterminados dentro del Bitbtn, que son:

bkAbort
bkAll
bkCancel
bkClose
bkHelp
bkIgnore
bkNo
bkOk
bkRetry
bkYes

Al seleccionar alguno de estos botones predeterminados, Delphi automáticamente modifica las propiedades: Caption, Glyph, etc.

## UN EDITOR DE GRÁFICOS

Para verificar el conocimiento de los componentes de diseño gráfico, se realizará la siguiente practica:

Se cambiarán las siguientes propiedades del formulario:

- Caption: *Editor de gráfico.*
- Height: *500.*
- Name: *Editor.*
- Position: *poScreenCenter*
- Width: *600.*

Se insertará sobre el formulario, un panel, el cual servirá de base para el área de dibujo, a este se le cambiarán las siguientes propiedades.

- Caption: (vacío).
- Color: *ClWhite.*
- Left: *20*
- Name: *PanelDibujo.*
- Top: *20.*
- Height: *420.*
- Width: *420.*

Se colocará un componente *image* sobre el panel, y se cambiarán las siguientes propiedades.

- Left: *0.*
- Name: *AreaDibujo.*
- Top: *0.*
- Height: *420.*
- Width: *420.*

Ahora, se insertará un botón para salir de la aplicación, para ello, se colocará un componente *Bitbtn*, sobre el formulario, al cual se le cambiarán las siguientes propiedades.

- Kind: *bkClose.*
- Left: *450.*
- Name: *BotonSalir.*
- Top: *420.*
- Height: *40.*
- Width: *120.*

Se le colocará el siguiente código al botón que se acaba de crear.

```
Procedure Teditor.BotonSalirClic(Sender: Tobject);
Begin
Close;
End;
```

Se colocará un botón para dibujar un cuadrado en el área de dibujo. Y se le colocarán las siguientes propiedades.

- Caption: *Cuadrado.*
- Left: *450.*
- Name: *BotonCuadrado.*
- Top: *20.*
- Height: *40.*
- Width: *120.*

Se colocará otro botón para dibujar un círculo en área de dibujo, y se le cambiarán las siguientes propiedades.

- Caption: *Círculo.*
- Left: *450.*
- Name: *BotonCirculo.*
- Top: *80.*
- Height: *40.*
- Width: *120.*

Se colocará otro botón para dibujar una elipse en área de dibujo. Y se le colocarán las siguientes propiedades.

- Caption: *Elipse.*
- Left: *450.*
- Name: *BotonElipse.*
- Top: *140.*
- Height: *40.*
- Width: *120.*

Se añadirán 5 paneles más al formulario, quedando del lado derecho del mismo, y cada uno hará una alusión a un color para el fondo del área de dibujo. Deberán tener las siguientes propiedades.

| **Propiedad** | **Panel1** | **Panel2** | **Panel3** | **Panel4** | **Panel5** |
|---|---|---|---|---|---|
| Left | *450* | *470* | *490* | *510* | *530* |
| Name | *PanelAul* | *PanelRojo* | *PanelVerde* | *PanelNegro* | *PanelBlanc* |
| Caption | *(nada)* | *(nada)* | *(nada)* | *(nada)* | *(nada)* |
| Color | *clBlue* | *clRed* | *clGreen* | *clBlack* | *clWhite* |
| Top | *200* | *200* | *200* | *200* | *200* |
| Height | *20* | *20* | *20* | *20* | *20* |
| Width | *20* | *20* | *20* | *20* | *20* |

Después de la fase de diseño de interfaz, es necesario colocar algo de código, que servirá de base para el funcionamiento de nuestra aplicación. En la sección ***private*** del código se colocará lo siguiente:

```
Private
    ColorFigura:TColor;
    Figura:byte;
    XRaton, Yraton:word;
```

Mediante estas 4 variables conoceremos el color de la figura que se desea dibujar, el tipo de figura que se creará y las coordenadas del puntero del ratón, donde se deberá de dibujar dicha figura.

En el evento *MouseMove* del área de dibujo, se colocará lo siguiente:

```
Procedure Teditor.AreaDibujoMouseMove(Sender:
Tobject; Shift:
                        TShiftState; X, Y: Integer);
Begin
XRaton:=x;
YRaton:=y;
End;
```

Y cuando el usuario seleccione alguna figura, haremos que la variable del tipo de figura se actualice automáticamente. En el código del botón de figura cuadrada, se colocará el siguiente código.

```
Procedure Teditor.BotonCuadradoClic(Sender:
TObject);
Begin
Figura:=0;
End;
```

En el botón de Círculo se colocará el siguiente código:

```
Procedure Teditor.BotonCirculoClic(Sender:
TObject);
Begin
Figura:=1;
End;
```

Y en el de elipse, lo siguiente:

```
Procedure Teditor.BotonElipseClic(Sender: TObject);
Begin
Figura:=2;
End;
```

Así mismo tendremos que actualizar el color de la figura mediante la variable que se le a asignado. Esto se realizará mediante los 5 paneles de color que se colocaron en el formulario.

```
Procedure TEditor.PanelAzulClic(Sender: Tobject);
Begin
ColorFigura:=clBlue;
End;
```

Lo mismo se colocará en los demás paneles, pero cambiando la instrucción del color por el color correspondiente(clRed, clBlack, clWhite, etc.).

Después solo colocamos el código de dibujo de figuras y colores dentro del área de dibujo, para que cuando se pulse, se dibuje automáticamente una figura en el área.

```
Procedure TEditor.AreaDibujoClic(Sender: TObject);
Begin
AreaDibujo.canvas.pen.color:=colorfigura;
AreaDibujo.canvas.brush.color:=colorfigura;
Case figura of
     0: AreaDibujo.canvas.rectangle(xraton-
     50,yraton-50,
                              xraton+50,
                              yraton+50);
     1: AreaDibujo.canvas.ellipse(xraton-50,
     yraton-50,
                              xraton+50,
                              yraton+50);
     2: AreaDibujo.canvas.ellipse(xraton-100,
     yraton-20,
                              xraton+100, y
                              raton+100);
end;
End;
```

Finalmente se asigna una figura y un color por defecto, el cual se presentará cada vez que se cargue el formulario.

```
Procedure TEditor.FormCreate(Sender: TObject);
Begin
Figura:=0          {Figura por defecto, cuadrado}
Colorfigura :=clBlue;       {color por defecto,
azul}
End;
```

## APLICACIONES MULTIMEDIA

Las aplicaciones multimedia, son aquellas que además de gráficos, poseen sonidos y en ocasiones hasta animaciones. Se verá el modo de construir un pequeño juego con animaciones multimedia.

## MULTIMEDIA EN DELPHI

Delphi ofrece la posibilidad de incluir opciones multimedia en nuestras aplicación. El nombre de multimedia, nace de la posibilidad de dar a conocer información por diversos medios. Los principales componentes para añadir propiedades multimedia a nuestros formularios son:

El componente *Animate*, de la pestaña Win32. Con el podemos visualizar ficheros de imágenes animadas AVI.
El componente *MediaPlayer*, de la pestaña System. Con el podemos reproducir sonidos almacenados en ficheros wav y mid.

## EL COMPONENTE *ANIMATE*

Este permite visualizar películas en los formularios. Existen diversos tipos de formatos de película, pero Animate solo permite trabajar con los de extensión AVI.
Las películas son una secuencia de imágenes que mostradas en secuencia y de modo rápido, provocan una sensación de movimiento. Además contiene sonido para dar mayor efecto de realismo, la mayoría de estas son cintas de vídeo, volcadas al disco duro mediante un tratado especial. Las propiedades principales del objeto animate son:

*FileName*: Esta propiedad contiene el nombre del fichero AVI que será visualizado.
*Repetitions*: Número de veces que se repetirá la "película". Cuando el fichero termine, volverá al principio y se mostrará de nuevo.
*StartFrame*: Imagen inicial, es decir el primer fotograma de la película que será visualizado, si vale 1 la película se mostrara desde el principio.
*StopFrame*: Imagen final, es decir, el ultimo fotograma de la película que se mostrará, si vale 0 la película se mostrará hasta le final.
*Activate*: Si vale TRUE, inicia la animación, si vale FALSE la animación se detiene.
*Visible*: Si vale FALSE el componente no se muestra en el formulario, se utiliza para ocultar la animación cuando se termina.

Nota: En Windows, existen animaciones estándares, las cuales se muestran al copiar un archivo, al vaciar la papelera de reciclaje, al buscar archivos, etc. Delphi permite utilizar estas animaciones estándares, sin utilizar ningún fichero AVI. Para ello, Animate tiene la propiedad *CommonAVI*, que puede tomar alguno de estos valores:

| Valor | Descripción |
|---|---|
| aviCopyFile | Una hoja vuela de una carpeta a otra. |
| aviCopyFiles | Varias hojas vuelan de una carpeta a otra. |
| aviDeleteFile | Varias hojas salen de una carpeta y se destruyen en el aire. |
| aviEmptyRecycle | Varias hojas arrugadas salen de la papelera y desaparecen en el aire. |
| aviFindComputer | Una lupa inspeccionando un ordenador. |
| aviFindFile | Una lupa inspeccionando una hoja. |
| aviFindFolder | Una linterna dando vueltas. |
| aviRecicleFile | Varias hojas salen de una carpeta para ir a parar a la papelera de reciclaje. |

Si se desea que cuando la animación termine, automáticamente se deberá poner en la propiedad repetitions el valor 1, y para que la ventana de animación se desaparezca cuando termine su animación, unicamente es necesario poner el código correspondiente en el evento OnStop.

## EL COMPONENTE MEDIAPLAYER

Este componente es uno de los mas completos con los que cuenta Delphi, pues no solo permite la reproducción de ficheros de audio, sino que además permite reproducir CD's, ficheros wav, videos, grabar sonidos, controlar dispositivos MDI exteriores, etc., etc., etc.
Todos los dispositivos multimedia necesitan de un controlador, estos pueden ser de 2 tipos distintos:

> Controladores Básicos o de bajo nivel: Son controladores muy específicos para un dispositivo muy determinado.
> Controladores MCI (*Media Control Interface*): Controladores muy generales que permiten controlar dispositivos muy distintos. Poseen una interfaz estándar, lo que hace que cualquier dispositivo MCI sea fácil de controlar.

Mediante el componente MediaPlayer podemos acceder a cualquier dispositivo MCI. De hecho, este componente es una interfaz para manejar este tipo de controladores, por lo que se puede manejar un amplio rango de dispositivos.

Para especificar al control que tipo de dispositivo se manejara, se utiliza la propiedad DeviceType en la que se almacena el tipo de dispositivo que se esta controlando. Sus valores son:

| Valor | Descripción |
|---|---|
| dtWaveStudio | Audio digital en forma de onda (WAV). |
| dtSequencer | Secuenciador MDI. |
| dtAVIVideo | Video en formato AVI. |
| dtCDAudio | CD de audio (música) |
| dtMMMovie | Animación en formato MMM. |
| dtDigitalVideo | Vídeo Digital |
| dtVideoDisc | Vídeo almacenado en CD |
| dtVCR | Videocasete o magnetoscopio |
| dtDAT | Cinta digital |
| dtOverlay | Superposición de vídeo |
| dtScanner | Digitalizador de imágenes |
| dtOther | Otro dispositivo que no este en la lista |
| dtAutoSelect | Selección automática del dispositivo. |

Las posibilidades que ofrece este control, son tan amplias que se puede colocar un sonido de inicio a una aplicación, una cierta interacción con el usuario. Incluso permite la grabación de ficheros de sonido WAV o MID.
El objeto Media player contiene 9 botones, los cuales controlan la ejecución de cualuier elemento que se este reproduciendo en dicho dispositivo. Las propiedades principales de este dispositivo son:

| Propiedad | Descripción |
|---|---|
| AutoEnabled | Cuando se esta grabando, no tiene sentido pulsar sobre el botón Eject o Play. Por ello, si se pone en True reconoce cuales botones no pueden ser utilizados y los desactiva. |
| AutoOpen | Si esta en TRUE, cuando comienze el programa intentara abrir el dispositivo automáticamente. |
| AutoRewind | Si se pone en TRUE, el dispositivo rebobinara automáticamente. |
| ColoredButton | Permite indicar so los botones aparecen coloreados o en negro, se debe especificar botón a botón. |
| DeviceType | Especifica el tipo se dispositivo con el que actuara MediaPlayer según la tabla anterior. |
| Display | Permite indicar el sitio donde se mostrarán las animaciones, este debe ser un formulario o panel. |

| | |
|---|---|
| EnabledButtons | Permite indicar que botones están activados y cuales se encuentran atenuados. |
| FileName | Nombre y ubicación del archivo que será reproducido. |
| VisibleButtons | Permite indicar que botones estarán visibles al usuario y cuales no. Se deberá indicar botón a botón. |

Una característica de los dispositivos multimedia es que tienen que ser "abiertos" antes de poder ser utilizados, para ello, se utiliza la instrucción Open. Es por ello, que cuando se elige un dispositivo en *DeviceType*, los botones del objeto MediaPlayer aparecen desactivados. El mejor método para abrir un dispositivo, es cuando se crea el formulario, para ello se debe seleccionar el evento OnCreate del Formulario.
Existe una propiedad en el objeto MediaPlayer que hace que el dispositivo se abra automáticamente, esa propiedad es AutoOpen; se le debe dar el valor TRUE para que se abra automáticamente.
La característica mas importante que tiene el objeto MediaPlayer es la de reproducir CD's de audio. Este dispositivo, mide el tiempo en pistas, las cuales a su vez cntiene un cierto tiempo, el cual es la duración de una canción. En el reproductos de animaciones AVI mide los frames.
Algunos dispositivos pueden trabajar con distintos formatos de tiempo, mientras que otros solo admiten un formato especifico. Para saber el formato con el que trabaja el dispositivo, se utiliza la propiedad TimeFormat, que contiene la medida de tiempo que se esta utilizando.
Medidas de tiempo de TimeFormat:

| **Constante** | **Formato** |
|---|---|
| tfMiliseconds | Milisegundos |
| tfFrames | Cuadros |
| tfBytes | Mytes |
| tfSampes | Muestras |
| tfHMS | Horas, minutos y segundos |
| tfMSF | Minutos, segundos y cuadros |
| tfTMSF | Pistas, minutos, segundos y cuadros |
| tfSMPTE24 | SMPTE a 24 cuadros |
| tfSMPTE25 | SMPTE a 25 cuadros |
| tfSMPTE30 | SMPTE a 30 cuadros |
| tfSMPTE30Drop | SMPTE a 29 cuadros |

Los valores de tiempo como tfTMSF, tfMSF y tfHMS son tiempos empaquetados, pues contienen varios valores en una sola constante. Para poder desempaquetarlos y obtener los valores que contienen por separado, se utilizan las siguientes funciones.

| Función | Parámetro | Devuelve |
|---|---|---|
| MCI_HMS_HOUR | tfHMS | Extrae la hora de una constante tfHMS |
| MCI_HMS_MINUTE | tfHMS | Extrae los minutos de una constante tfHMS |
| MCI_HMS_SECOND | tfHMS | Extrae los segundos de una constante tfHMS |
| MCI_MSF_MINUTE | tfMSF | Extrae los minutos de una constante tfMSF |
| MCI_MSF_SECOND | tfMSF | Extrae los segundos de una constante tfMSF |
| MCI_MSF_FRAME | tfMSF | Extrae los cuadros de una constante tfMSF |
| MCI_TMSF_TRACK | tfTMSF | Extrae las pistas de una constante tfTMSF |
| MCI_TMSF_MINUTE | tfTMSF | Extrae los minutos de una constante tfTMSF |
| MCI_TMSF_SECOND | tfTMSF | Extrae los segundos de una constante tfTMSF |
| MCI_TMSF_FRAME | tfTMSF | Extrae los cuadros de una constante tfTMSF |

Una vez conocida la medida de tiempo que utiliza el dispositivo, tan solo nos falta conocer algunos datos de este, como la cantidad que contiene, y para el caso en que hayamos empezado a extraer información del dispositivo, saber cuanta información hemos usado ya y cuanta nos queda por usar.

Para conocer estas informaciones, se utiliza:

Propiedad *Length*: Contiene el tamaño de la información contenida en el dispositivo. Este tamaño viene dado en la unidad de medida que retorne la propiedad *TimeFormat*. Esta propiedad es de solo lectura y puede ser utilizada desde el momento en el que se abre el dispositivo.

Propiedad *Position*: Marca la posición actual del dispositivo, en medida de tiempo, por ejemplo el minuto y segundo actual de una canción, a diferencia de la propiedad Length esta si puede ser alterada para brincar en la melodía.

Propiedad *Tracks*: Permite conocer cuantas pistas existen en el dispositivo.

Propiedad *trackPosition*: Es una matriz que contiene los inicios de cada pista en el dispositivo. Así TrackPosition[0] es el inicio de la canción 0, TrackPosition[5] es el inicio de la canción 5.

Propiedad *TrackLength*: Es una matriz que contiene la longitudes de cada pista. Así por ejemplo TrackLength[2], contiene la longitud de la pista 2.