Programación con Delphi (V)

© Francisco Charte Ojeda - http://www.fcharte.com

Sumario

Una de las características de Delphi que más gusta a sus usuarios, los desarrolladores, es el lenguaje en que está basado: Object Pascal. En esta penúltima entrega conocemos las bases de este lenguaje.

Desde que Niklaus Wirth lo crease a finales de los sesenta, Pascal ha sido uno de los lenguajes más importantes en el ámbito informático, tanto en número de usuarios como en las influencias que dicho lenguaje ha tenido sobre otros. Como se apuntaba al inicio de la primera entrega de este curso, aunque Pascal fue creado, inicialmente, como una herramienta académica, la aparición de herramientas como Turbo Pascal le pusieron al mismo nivel que C. Lo mejor de Object Pascal, una versión avanzada y orientada a objetos del Pascal original, es que ofrece una claridad y elegancia en el código que otros lenguajes, como C o C++, no tienen. Al tiempo, Object Pascal tiene prácticamente las mismas posibilidades que dichos lenguajes, y el código objeto generado por Delphi a partir de código fuente Object Pascal es comparable en eficiencia.

En las entregas previas ha tenido ocasión de conocer algunos de los elementos fundamentales de Object Pascal, sobre todo los que tienen que ver con las clases de objetos y su uso. Para poder crear aplicaciones con Delphi, no obstante, eso no es suficiente. Es necesario, además, saber cómo podemos manipular los datos, almacenándolos en variables y efectuando operaciones. También es preciso conocer las estructuras de control básicas para poder tomar decisiones, repetir procesos, etc.

Lógicamente, en esta entrega lo único que se persigue es efectuar una introducción a Object Pascal de manera totalmente informal. Este lenguaje es tan rico que harían falta muchas más páginas para poder estudiarlo y, de hecho, hay libros completos dedicados exclusivamente a Pascal y Object Pascal.

Estructura general de una aplicación

Las aplicaciones escritas con Object Pascal pueden componerse de uno o varios módulos de código que, básicamente, siempre tienen la misma estructura general. Uno de esos módulos, el único imprescindible, contendrá el código de entrada a la aplicación, mientras que los demás pueden almacenar definiciones de funciones y procedimientos, clases de objetos y otros elementos.

Los módulos de código Object Pascal cuentan con una cabecera que se inicia con la palabra `program` o `unit`, seguida del nombre del módulo. Al trabajar con Delphi, el nombre de cada módulo es establecido automáticamente, en el momento en que éste es guardado. Tan sólo puede existir un módulo con el encabezado `program`, mientras que puede haber ninguno, uno o varios módulos con el encabezado `unit`.

Cuando se ejecuta un programa escrito con Object Pascal, el código al que se transfiere el control es aquél delimitado por las palabras `begin` y `end` en el módulo con el encabezado `program`. Dicho módulo es, como se decía antes, el punto de entrada a toda aplicación. Su estructura general será la mostrada en el Listado 1. Observe que tras la cabecera aparece una cláusula `uses`. Ésta sirve para referenciar a otros módulos que componen el proyecto, concretamente aquellos módulos cuyo contenido va a ser usado directamente al inicio del programa.

La estructura del resto de los módulos será similar a la que puede ver en el Listado 2. Cada módulo se divide en dos grandes partes: la interfaz y la implementación, delimitadas por las palabras `interface` e `implementation`. La interfaz contiene los elementos que podríamos considerar públicos y, por tanto, pueden ser usados desde el módulo principal o cualquier otro. En esta sección suelen definirse tipos de datos, clases de objetos y variables. La sección de implementación, como de su propio nombre puede deducirse, contiene los detalles de implementación, es decir, métodos de objetos, funciones, etc. Todos los elementos de esta sección quedan ocultos, no siendo accesibles para el resto de módulos del proyecto.

Los tipos, clases y variables también pueden aparecer en la sección de implementación, caso éste en que sólo serán visibles para ese módulo. Serán, por tanto, elementos con un ámbito no público. Por último, también pueden incluirse algunos de esos elementos, por ejemplo tipos y variables, en el interior del cuerpo de funciones y procedimientos. Éste es el ámbito más reducido, conocido como local, ya que esos elementos sólo pueden usarse desde el interior de dichas funciones.

En un proyecto desarrollado con Delphi el módulo principal, con el encabezado `program`, es el que contiene el código de proyecto, en el que se referencian todos los módulos y se crean los formularios. Otros elementos, como los formularios y módulos de datos, tienen asociados módulos `unit` en los que se alojan las definiciones de tipos y el código asociado a los eventos.

Manipulación de datos

Para poder efectuar las funciones que se le encomiendan, la mayoría de las aplicaciones tienen que manipular datos obtenidos de fuentes externas: el teclado, archivos en disco, una red, etc. Para ello, dichos datos debe almacenarse temporalmente en memoria, usando variables, y a continuación actuar como operandos, generalmente en expresiones aritméticas, relacionales y condicionales.

Como se ha apuntado antes, las variables de un programa Object Pascal pueden tener ámbitos diferentes, según el lugar en que se declaren. Si el punto de declaración es el cuerpo de una función, por ejemplo, tendremos una variable local. En caso de que la variable deba ser compartida por varias funciones de un mismo módulo, lo normal es declararla en la sección de implementación. Por último, tenemos el ámbito público. Éste se obtiene al declarar la variable en la sección de interfaz, consiguiendo así que sea accesible desde cualquier punto del código del programa.

En cualquier caso, la sección de declaración de variables siempre se inicia con la palabra `var`. De igual forma, la sintaxis para esa declaración es también siempre idéntica, siguiendo el patrón `variable: tipo;` donde `tipo` determina la información que puede contener esa variable. En el Listado 3 puede ver diversas declaraciones de variables en varios ámbitos. La Figura 1 muestra una lista jerárquica en la que se clasifican los distintos tipos existentes en Object Pascal.

Las variables representan el medio por el cual un programa puede, al ejecutarse, almacenar en memoria los datos que precise. Estos datos, lógicamente, tendrán que manipularse, operando sobre ellos para obtener un resultado. Aquí es donde entran en escena los operadores, los elementos que permiten efectuar diversas acciones sobre los operandos que serían los valores almacenamos en las variables.

Mediante los operadores puede efectuar operaciones aritméticas con valores numéricos, concatenar cadenas, analizar la relación existente entre varios elementos, comprobar la existencia de un valor en un conjunto, etc.

Definición de nuevos tipos

Aparte de los tipos básicos o nativos del propio lenguaje, como los números en diferentes precisiones, las cadenas o los punteros, en Object Pascal el programador puede definir sus propios tipos. Es posible definir conjuntos, enumeraciones, subrangos, registros y clases de objetos. Tras la definición del nuevo tipo, pueden declararse variables de ese tipo tratándolo como a cualquier otro de los existentes en el lenguaje, sin diferencias aparentes.

Una enumeración, como su propio nombre indica, es un tipo de dato en el que son posibles sólo una lista determinada de valores. En el Listado 4 puede ver un ejemplo de definición. En este caso, el tipo `TDiaSemana` sería equivalente a `Integer` o `Boolean`, es decir, un tipo ordinal. La diferencia es que los límites de `TDiaSemana` no son `true` y `false` o 0 a 255, sino `Lunes` a `Domingo`. Una variable de tipo `TDiaSemana`, por tanto, sólo puede contener uno de esos valores. Por lo demás, es posible utilizar con una variable del nuevo tipo cualquier operador habitual de Object Pascal. En el Listado 4, por ejemplo, puede ver cómo se usa la función `succ()` para obtener el valor siguiente a uno dado.

Los conjuntos, por su parte, son tipos estructurados y no ordinales, como las enumeraciones. Un conjunto se define siempre a partir de un tipo ordinal, dando lugar a un nuevo tipo que puede contener ninguno, uno, varios o todos los valores de ese tipo base. Observe el código del Listado 5. En él se ha definido un conjunto, `Set Of`, partiendo de la enumeración `TDiaSemana` anterior. El resultado es un nuevo tipo, llamado `TDiasSemana`, que puede estar vacío o contener cualquier número de los elementos de `TDiaSemana`. En dicho ejemplo puede ver, además, cómo se asigna a

una variable un conjunto con tres días, y cómo se usa a continuación el operador `in` para comprobar la existencia de un determinado elemento en ese conjunto.

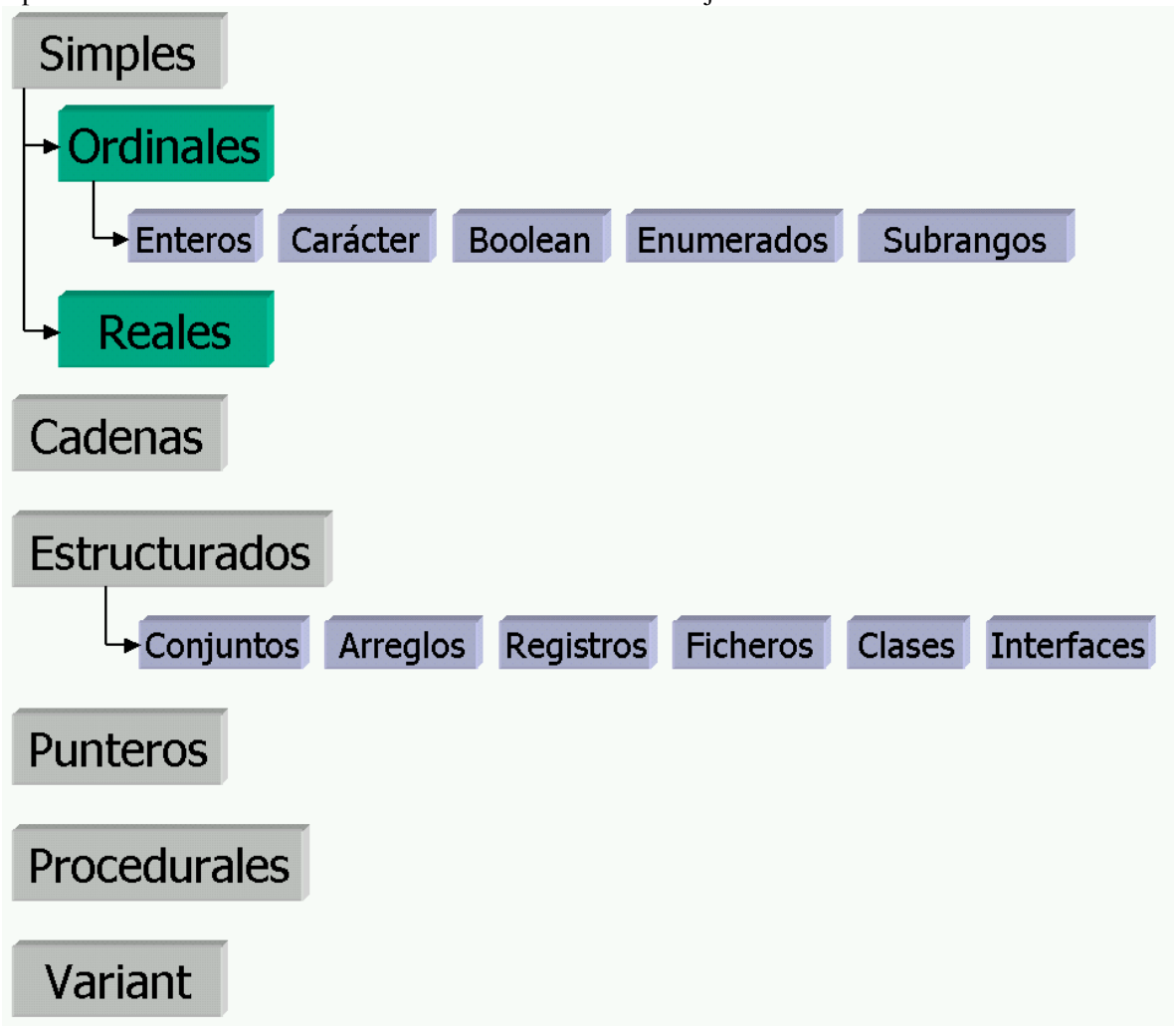

Figura 1. Los tipos de datos en Object Pascal se estructuran en varias categorías, dando lugar a esta pseudo-jerarquía.

Al igual que los conjuntos, también los registros forman parte de los tipos estructurados. Un registro, conocido como estructura en otros lenguajes, está compuesto de varios miembros o elementos, que pueden ser, a su vez, de cualquier tipo. En el Listado 6 puede ver cómo se define un registro simple, compuesto de tres miembros de tipos diferentes: un `TDiaSemana`, un `TTime` y una cadena. Definido el nuevo tipo, declarar una variable es algo tan simple como en los casos anteriores. Por último, a la hora de acceder a esta variable, hay que tener en cuenta que contiene varios elementos, por lo que es preciso especificar su nombre.

Por último, y al igual que en la mayoría de los lenguajes actuales, Object Pascal permite declarar matrices de cualquier tipo de dato, facilitando la creación de listas, tablas y estructuras más complejas. Estas estructuras, además, pueden ajustar su tamaño dinámicamente durante la ejecución del programa.

Estructuras condicionales

El código de un programa no está, generalmente, pensado para ejecutarse secuencialmente, desde la primera sentencia hasta la última, con cada inicio de la aplicación por parte del usuario. Dependiendo de las acciones de éste y los datos a tratar, es habitual optar por la ejecución de un código u otro, o bien repetir el mismo código mientras se dé una cierta condición. Para poder

diseñar ese tipo de construcciones contamos con diversas estructuras de control, siendo las más importantes las estructuras condicionales y las de repetición.

Una estructura condicional permite optar por la ejecución de una determinada porción de código según unas condiciones que, dependiendo de los casos, pueden ser más o menos complejas. La estructura condicional más conocida, existente en la mayoría de los lenguajes imperativos, es la conocida `if-then-else`. Previamente, en el ejemplo del Listado 5, ha podido ver un ejemplo de uso de esta estructura, concretamente para mostrar un mensaje en pantalla sólo en caso de que se dé una condición: que un conjunto contenga el elemento `Miercoles`.

La Figura 2 representa la estructura típica de una sentencia condicional con todas sus partes. En un determinado punto, el flujo del programa llega a la sentencia `if` y analiza la expresión condicional que le acompaña. Si ésta se cumple, lo que equivale a decir que el resultado devuelto es `true`, se ejecutará la sentencia que sigue a la palabra `then`. En caso contrario, la sentencia ejecutada será la que sigue a la palabra `else`. En lugar de una sola sentencia, ambas partes pueden ser un bloque de código delimitado por las palabras `begin` y `end`.

Una alternativa a esta conocida construcción condicional, útil sobre todo cuando quieren comprobarse varias expresiones con un operando común, es la sentencia `case`. Ésta deberá ir seguida del operando común, al que se conoce generalmente como selector, y la palabra `of`. Entre esta apertura y el final del bloque, delimitado por la palabra `end`, podremos introducir cuantos valores deseemos comparar con el selector. El resultado es una construcción como la del Listado 7, en la que se ejecutan diferentes sentencias dependiendo del valor de la variable `Hoy`. Si el valor de dicha variable es `Lunes` se muestra un mensaje, si es cualquier día entre `Martes` y `Jueves` otro distinto, etc. Observe la sintaxis utilizada para crear un subrango de valores.

Estructuras repetitivas

Como se apuntaba al inicio del punto anterior, además de las condicionales también existen estructuras repetitivas que permiten ejecutar más de una vez el mismo código. A estas estructuras se las conoce habitualmente como bucles. Clásicamente han existido siempre dos categorías de bucles: por contador y por condición. En la primera categoría entran aquellos cuya ejecución se rige por el valor de una variable que, a cada ciclo, se incrementa o decrementa hasta llegar a un determinado límite. En realidad, dicha categoría es simplemente un caso particular de los bucles por condición que, como puede suponer, ejecutan el código mientras una expresión condicional sea cierta o falsa.

La Figura 3 es una representación genérica del funcionamiento de un bucle, independientemente de que esté controlado o no por un contador. En este caso, no obstante, la comprobación de la condición se efectúa al inicio, por lo que si ésta, ya desde principio, es falsa, no llegaría nunca a entrarse en el bucle. Es posible efectuar dicha comprobación al final del bucle, de tal forma que éste se ejecute al menos una vez.

Los bucles por contador rigen su funcionamiento, como se ha dicho antes, sobre el valor de una variable. Este valor toma un valor inicial que, posteriormente, va incrementándose o decrementándose con cada ciclo. El valor puede ser cualquiera de tipo ordinal, por lo que es posible usar caracteres, enumeraciones o, más típicamente, números enteros. En el Listado 8 puede observarse la estructura típica de un bucle por contador. Observe cómo detrás de la sentencia `for` se indica la variable que servirá como contador, así como el valor inicial que tomará éste. Tras la palabra `to` se especifica el valor objetivo o límite, en este caso mayor que el de inicio. Si fuese a la inversa, caso en el que habría que decrementar el contador, cambiaríamos `to` por `downto`. Por último, tras la palabra `do`, hemos dispuesto las sentencias a ejecutar en cada ciclo. En este caso tan sólo una.

En caso de que el número de ciclos del bucle no pueda controlarse mediante un contador, porque no exista un incremento o decremento predeterminado, es posible usar las sentencias `while..do` y `repeat..until`. La única diferencia entre ambas es que el condicional que controla el bucle, una expresión que devolverá `true` o `false`, con `while` se dispone al inicio y causa la salida con el valor `false`, mientras que con `repeat` el condicional se codifica al final del bucle y éste finaliza con el valor `true`.

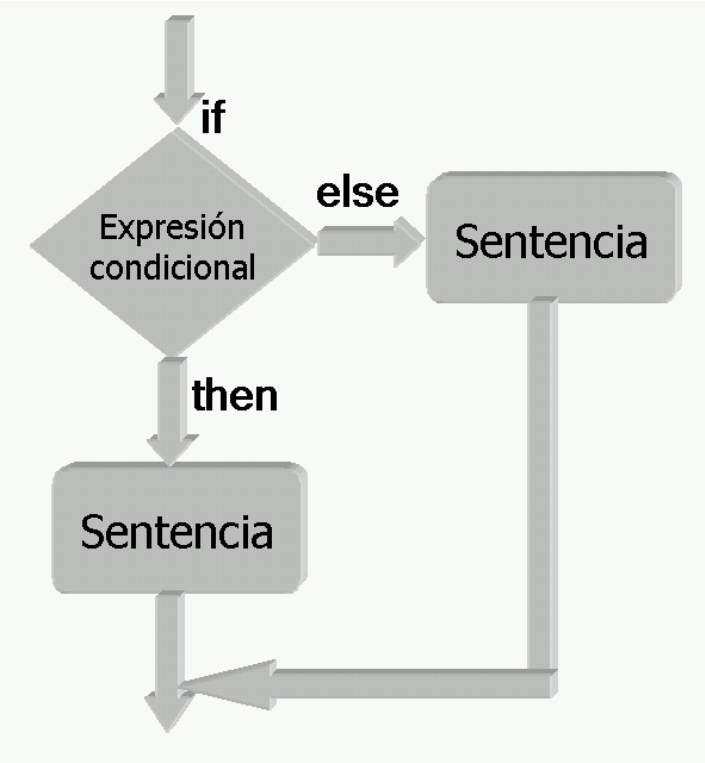

Figura 2. Representación gráfica de una estructura condicional clásica if-then-else.

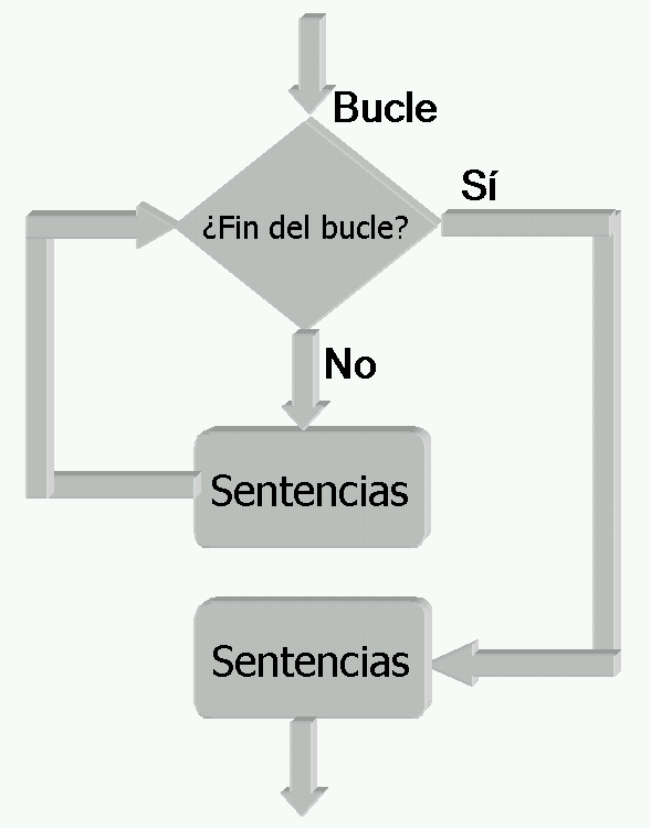

Figura 3. Representación gráfica de un bucle típico con comprobación condicional al inicio.

Visto y por ver

Si a lo cubierto en esta entrega, dedicada monográficamente a Object Pascal, sumamos las cuatro entregas anteriores, nos daremos cuenta que ya conocemos el entorno de Delphi; sabemos cómo crear un proyecto, compilarlo y ejecutarlo; podemos usar los controles más básicos de Delphi y, además, tenemos las bases de Object Pascal necesarias para iniciar la codificación. Para completar este curso, con el objetivo de que el lector adquiera las bases suficientes para comenzar a trabajar con Delphi, la última entrega la dedicaremos al proceso de depuración. Saber depurar una aplicación es, actualmente, casi tan importante como conocer los controles o las sentencias que debemos utilizar.

```
program Project1;

uses
  Unit1 in 'Unit1.pas' {Form1};

begin
  // Sentencias a ejecutar
  //…
end.
```

Listado 1. Estructura típica del módulo con el punto de entrada a la aplicación.

```
unit Unit1;

interface

uses
  // Otros módulos

type
  // Definición de tipos y clases

var
  // Definición de variables

implementation

// Código de implementación

end.
```

Listado 2. Estructura general de un módulo Object Pascal.

```
unit Unit2;

interface

var
  iContadorGlobal: Integer;

implementation

var
  iContadorDeModulo: Integer;

procedure ActualizaContador;
var
  iContadorLocal: Integer;
begin
   Inc(iContadorGlobal);
   Inc(iContadorDeModulo);
   Inc(iContadorLocal);
end;

end.
```

Listado 3. Módulo con tres variables declaradas en ámbitos distintos.

```
type
  TDiaSemana = (Lunes, Martes, Miercoles, Jueves, Viernes, Sabado, Domingo);

var
  Ayer, Hoy: TDiaSemana;

begin
  Ayer := Jueves;
  Hoy := succ(Ayer);
end;
```

Listado 4. Definición de una enumeración y declaración de dos variables del nuevo tipo.

```
type
  TDiasSemana = Set Of TDiaSemana;

var
  Citas: TDiasSemana;

begin
  Citas := [Martes, Miercoles, Viernes];
  if Miercoles in Citas then
     ShowMessage('Sí tienes cita');
end;
```

Listado 5. Definición de un conjunto y uso del operador In para comprobar la existencia de un elemento.

```
type
  TCita = record
    Dia: TDiaSemana;
    Hora: TTime;
    Nombre: String;
  end;

var
  Cita: TCita;

begin
   Cita.Dia := Martes;
   Cita.Hora := StrToTime('12:30');
   Cita.Nombre := 'Francisco Charte';
end;
```

Listado 6. Definición de un registro y posterior uso para declarar una variable.

```
var
  Hoy: TDiaSemana;

begin
   Hoy := Jueves;

   case Hoy of
     Lunes:
        ShowMessage('La semana está comenzando');
     Martes..Jueves:
        ShowMessage('Ya estás a mitad de semana');
     Viernes:
        ShowMessage('El fin de semana está a la vuelta');
     Sabado, Domingo:
        ShowMessage('Disfruta del descanso');
   end;
end;
```

Listado 7. Estructura condicional `case-of`, que permite evaluar varias expresiones contra un mismo operando.

```
type
  TDiaSemana = (Lunes, Martes, Miercoles, Jueves, Viernes, Sabado, Domingo);

var
  Dia: TDiaSemana;
  Anotaciones: array[TDiaSemana] of integer;

begin
  for Dia := Lunes to Domingo do
    Anotaciones[Dia] := 5;
end;
```

Listado 8. Construcción típica de un bucle por contador.