Introducción a **Rad Studio XE Delphi - C++ builder** - Firemonkey

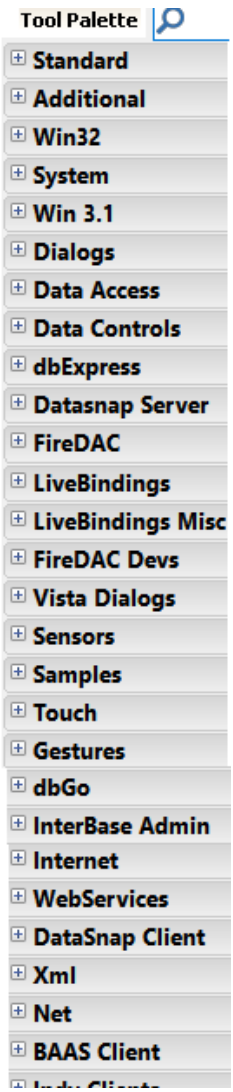
Embarcadero **Rad Studio** es un IDE Interface de Desarrollo del tipo RAD (Diseño rápido de aplicaciones) que permite utilizar la programación orientada a objetos utilizando como base dos lenguajes:

- El **lenguaje C++** en modo visual denominado **C++ Builder**. Aunque también permite hacer aplicaciones C++ de consola.
- El **lenguaje Pascal** orientado a objetos denominado aquí **Delphi.**

Librerías y entorno:

- Este IDE comparte las mismas funciones y librerías básicas creadas exlusivamente para el sistema operativo Windows (32 y 64), llamadas *VCL* tanto para Delphi como para C Builder.
- Las versiones **XE** incorporan la variante ***Firemonkey*** o entorno *multi-device* que permiten compilar en multiplataforma OS – Android e incluso dispositivos móviles.

Par más información véase la página: http://docs.embarcadero.com/products/rad_studio/



Conceptos básicos de programación.

- **Proyecto**. Un proyecto es un conjunto de módulos de formulario, de códigos y de archivos que componen una aplicación. La ventana View ▸ ***Project Manager*** muestra todos los archivos que componen una aplicación.
- **Formulario**. Ventana con el diseño y el código asociado a dicho formulario.
- **Componentes**. Los componentes son objetos como cuadros, botones, etiquetas… que se disponen en un formulario o ventana para manipular o mostrar datos. Para poner los componentes en un formulario usar la *Tool Palette*.
- **Propiedades**. Usando la ventana de Object inspector (F11) se definen las propiedades de formularios y controles. Las propiedades especifican los valores iniciales de las características, tales como tamaño, nombre y posición.
- **Eventos**: Es la respuesta a las acciones del usuario que desencadena una acción en el programa.

Concepto de código:

Es el texto escrito en un programa. El programa se divide en distintos apartados o subrutinas generalmente como respuesta a un evento y se divide en procedimientos (subrutinas) o funciones.

- ***Acciones o subrutinas***: Trozos de código como respuesta a un evento o acción. No recoge ni devuelve ningún dato.
- ***Procedimientos***: Trozos de código como respuesta a un evento o acción. No devuelven ningún valor pero sí puede recoger uno o varios parámetros (especificados entre paréntesis).
- ***Funciones***: Son procedimientos que además devuelven un valor al procedimiento desde donde se le llamó.
- ***Diagrama de Flujo***: Representa la esquematización gráfica de un algoritmo, el cual muestra gráficamente los pasos o procesos a seguir para estructurar bien un programa.

Eventos y acciones (sobre un componente)**:**

Lista de eventos (Desencadena la acción de un objeto)

Evento	Acción
Click	Al hacer clic con el mouse (o el teclado)
DragDrop	Al soltar un control arrastrado con el mouse
DragOver	Al Arrastrar un control con el mouse.
KeyDown	Presionar una tecla mientras el objeto tiene el foco.
KeyPress	Presionar una tecla y devolver su valor ASCII.
KeyUp	Liberar una tecla mientras el objeto tiene el foco.
MouseDown	Presionar el botón del mouse sobre el objeto.
MouseMove	Mover el puntero del mouse por encima del objeto
MouseUp	Liberar el botón del mouse en el objeto.

Propiedades básicas de los objetos:

Alignment	- Alineación
BevelInner	- borde interior
BorderStyle	- estilo del borde
Caption	- rótulo
TabStop	- Salto tabulación
Color, Cursor	- Color, puntero
Enabled	- Activo
Font	- Tipo letra
Height	- Altura del objeto
Hint	- Etiqueta ayuda
Left	- Posición izquierda
Name	- Nombre del objeto
PopupMenu	- Menú contextual
ShowHint	- Mostrar etiquetas
Top	- Posición vertical
Visible	- Visible
Width	- Ancho del objeto

Acceso al programa RAD Studio

En el menú de Inicio de Windows - Todas las Aplicaciones – Embarcadero. Dispones de un acceso directo para crear sólo proyectos en *Delphi* o sólo *C++ Builder* o el acceso a ***RAD Studio*** para crear un proyecto en cualquiera de los dos lenguajes.

Crear y guardar un proyecto:

Prepara la carpeta de trabajo: En primer lugar, antes de trabajar con un proyecto, ya sea Delphi o C++ , es aconsejable que crees una carpeta (subdirectorio) con el nombre del proyecto para separar el conjunto de archivos que contendrá cada uno de ellos. RadStudio XE? genera en la carpeta de documentos personales, una carpeta Projects para guardar los proyectos:

Ruta de muestra: *C:\Users\Ofimega\Documents\Embarcadero\Studio\Projects*

y en los documentos públicos pone las carpetas compartidas como los ejemplos:

Ruta de muestra: *C:\Users\Public\Documents\Embarcadero\Studio\17.0\Samples*

Crear un Proyecto en Delphi:

- Para hacer aplicaciones de escritorio ***Windows***:
  Escoge del menú de Rad Studio la opción: File ▸ New – VCL Forms Application Delphi
  (ó File ▸ New – *Application Delphi*, según las versiones)
- Para hacer aplicaciones multidispositivo ***Firemonkey***:
  Escoge del menú de Rad Studio la opción: File ▸ New – Multi-device Application - Delphi
  (ó File ▸ New – Firemonkey desktop application Delphi, según las versiones)

Crear Proyecto en C++ Builder:

- Para hacer aplicaciones de escritorio ***Windows***:
  Según las versiones, escoge del menú de *Rad Studio* la opción: *File ▸ New – Application C++ Builder*
  ó *File ▸ New – VCL Forms Application C++Builder*
- Para hacer aplicaciones multidispositivo ***Firemonkey***:
  Según las versiones, escoge del menú de Rad Studio la opción: File ▸ New – Multi-device Application - C++ Builder ó File ▸ New – Firemonkey desktop application C++ Builder

Guardar un proyecto en Delphi

- Primero hay que guardar la unidad principal de la aplicación, ej: **Hola1.pas** donde se encuentra el código fuente en **Pas**cal. Se genera por defecto el archivo del formulario asociado: **Hola1.dfm** que contiene el diseño de la ventana.
- Finalmente guardamos el archivo del proyecto conjunto **Hola.dproj** que también genera **Hola.dpr** para versiones anteriores de Delphi.

Guardar un proyecto en C++Builder

Primero guardamos el archivo fuente **Hola1.cpp** que genera por defecto el archivo del formulario **Hola1.dfm** y el archivo cabecera de unidades internas: **Hola1.h**. En segundo lugar, guardamos el archivo cabecera: **Hola.h** que incluye las unidades externas y finalmente guardamos el archivo del proyecto conjunto **Hola.cbproj**.

Reglas de nomenclatura al declarar procedimientos o variables:

- Deben comenzar por una letra
- No pueden contener puntos
- No puedes superar los 255 caracteres
- Nombres de formularios no sobrepasar 40 caracteres
- No pueden tener el mismo nombre que una palabra clave

Comentarios en el código

- En una línea: Se inician con: // tanto para Delphi como para C++
- En varias líneas: Delphi van entre { . . . } C++ van entre los signos /* . . . */

Normas para escribir el código:

- Para referirmos o cambiar una propiedad de un objeto en tiempo de ejecución:
  En Delphi: Escribiremos el nombre del objeto, un punto y su propiedad:
  Ejemplo: `Button1.Caption='Vale';` si asignamos una cadena de texto irá entre apóstrofes
  En C++: Escribiremos el nombre del objeto, una flecha formada por los caracteres -> y su propiedad:
  Ejemplo: `Button1->Caption="Vale";` si asignamos una cadena de texto irá entre dobles comillas
- Atención a las mayúsculas en C++: Importante escribir "Button" con la 1ª letra en mayúsculas, ya que C++ distingue mayúsculas de minúsculas.
- Para <u>asignar</u> un valor a un objeto o variable
  En Delphi: Se indica con **:=** Ejemplo: `ProgressBar1.position :=100;`
  En C++: Se indica con **=** Ejemplo: `ProgressBar1->Position = 100;`
- Para <u>comparar</u> el valor de un objeto o variable
  En Delphi: Se indica con el signo igual **=** Ejemplo: `if ProgressBar1.position=100 then ...`
  En C++: Se indica con el doble signo igual **==** Ejemplo: `if (ProgressBar1.Position == 100)`

Lenguaje base

Estructuras de control

- if (comparador si-entonces)
- if...else *(comparador si...sino)*
- for *(bucle con contador)*
- switch / case *(comparador múltiple)*
- while *(bucle por comparación booleana)*
- break *(salida de bloque de código)*
- continue *(continuación en bloque de código)*
- return *(devuelve valor a programa*

Tipos

- boolean (booleano)
- char (carácter)
- byte (entero hasta 226)
- int (entero)
- unsigned int (entero sin signo)
- long (entero 32b)
- unsigned long (entero 32b sin signo)
- float (decimal de coma flotante de 16b)
- double (decimal coma flotante de 32b)
- string (cadena de caracteres)
- array (cadena matriz)
- void (vacío)

Operadores

- asignación C = Delphi :=
- igual a C == Delphi =
- suma + resta -
- multiplicación *
- división /
- resto %
- distinto de C != Delphi <>
- menor que <
- mayor que >
- menor o igual que <=
- mayor o igual que >=
- y && o || Delphi And - Or
- negación ! Not
- incremento / decremento ++ --

Funciones matemáticas

- min () (mínimo)
- max () (máximo)
- abs () (valor absoluto)
- sq () (eleva al cuadrado)
- sqrt () (raíz cuadrada)
- sin () (seno) cos () (coseno) tan () (tangente)

- **Comparadores:**

Comparador	Delphi	C++	ejemplo
Igual	=	==	if ProgressBar1.position=100 then showmessage ('Has llegado a 100');
Mayor	>	>	if ProgressBar1.position>100 then showmessage ('Te has pasado de 100');
Menor	<	<	if ProgressBar1.position>100 then showmessage ('Aún no has llegado a 100);
Mayor o igual	>=	>=	if edad>=100 then showmessage ('Has llegado o te has pasado de 100');
Menor o igual	<=	<=	if edad<=100 then showmessage ('Aún no has llegado a 100);
Distinto	< >	! =	if edad < > 100 then showmessage ('No es 100');

Conversión de tipos:

IntToStr (Integer to String) ▸ Cambia del tipo de número entero a cadena de texto
StrToInt (String to Integer) ▸ Cambia del tipo cadena de texto a número entero
Otros:

StrToFloat (texto a decimal flotante)
FloatToStr (Decimal flotante a texto))
FloatToDecimal (Decimal flotante a decimal
Formatfloat (Decimal flotante a texto con formato)
DateTimeToStr (Fecha/hora a texto)
DateToStr (Fecha a Texto)
StrToTime (Texto a hora)
StrToDate (Texto a fecha)

Ventanas de mensaje. (3 métodos para mostrar o pedir mensajes al usuario):

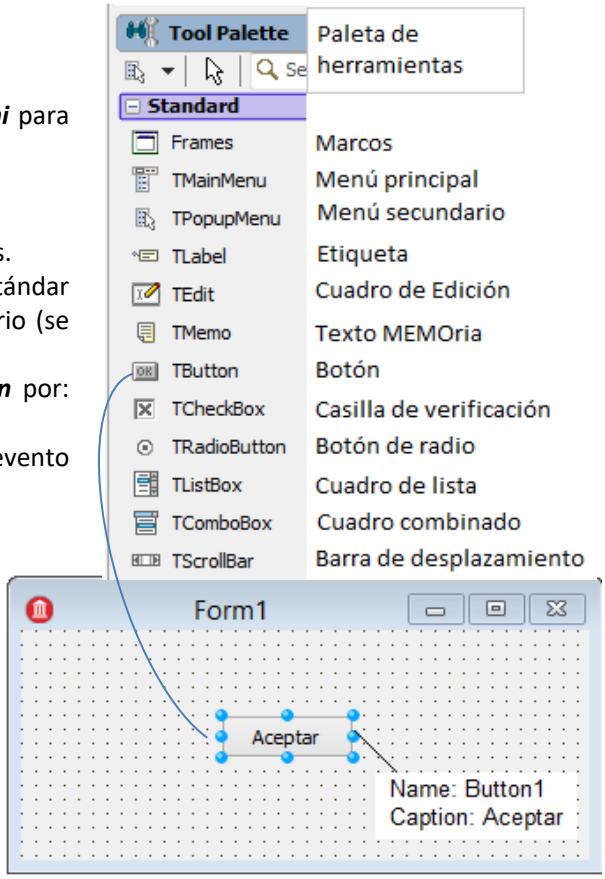
1. Usando las funciones VCL internas que incorpora C++Builder:
   - ***ShowMessage***(): Muestra un mensaje de texto en una ventana
   - ***MessageDlg***(): Muestra una ventana con mensaje y botones de respuesta
     Los botones de acción pueden ser: mbOK – mbCancel – mbYes – mbNo – mbAbort – mbRetry – mbClose
     Y sus repuestas son: mrOk - mrCancel – mrYes – mrNo – mrAbort – mrRetry - mrClose
   - ***MessageDlgPos***() ídem pero con posibilidades extras
2. Usando las funciones de la API de Windows
   - **Application->*MessageBox***(): Muestra un mensaje de texto en una ventana de Windows estándar con o sin botones de respuesta que pueden ser: MB_OK - MB_OKCANCEL- MB_ABORTRETRYIGNORE - MB_YESNOCANCEL - MB_YESNO - MB_RETRYCANCEL - MB_HELP
3. Creando nuestros propios formularios/ventanas:
   - Primero: crear un nuevo formulario secundario: Flie – New – Form donde se pondrá el mensaje y los botones
   - Segundo: incluir la unidad del formulario en la principal: #include "secundario.h" o USES
   - Tercero: Mostrar la ventana utilizando las funciones show() o showmodal();

Pedir datos al usuario en ventanas:

1. Utilizando ventanas con entradas de texto:
   - ***InputBox***('Nombre_ventana', 'Texto_Dato', 'Respuesta_predeterminada'); → Devuelve Var String
   - ***InputQuery***('Identificate!', 'Pon tu nombre, por favor...', Nombre); → Devuelve Var Bolean
2. Añadiendo componentes de diálogo estándar al formulario desde la paleta *Dialogs*:
   - ***Opendialog*** – Savedialog: Componentes para abrir y guardar archivos
   - ***PrintDialog*** – PrinterSetupDialog - PageSetupDialog: Componentes para imprimir y configurar impresión
   - ***FontDialog*** – ColorDialog - FindDialog: Cambiar la fuente, el color o buscar texto.

Ejercicios básicos. Crear una ventana “hola mundo”.

Ejercicio 1:



Crear la aplicación en Delphi:

- Escoge del menú: File ▸ New ▸ VCL Form Applicaton - ***Delphi*** para crear un nuevo proyecto en Delphi.
  Te aparecerá una ventana o formulario vacío (*Form*).
  En el panel *Inspector de objetos (Object Inspector),* verás las propiedades de la ventana o de los objetos que en ella pongas.
- Desde la barra de herramientas ***Tool palette***, En la sección estándar busca el componente T***Button*** y arrástralo sobre el formulario (se mostrará el botón *Button1).*
- Desde el panel de ***Properties***, cambia su propiedad ***Caption*** por: *Aceptar*
- Pulsa *doble clic* sobre el botón e introduce el código (en el evento ***OnClic*** del botón *Button1*):

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Button1.Caption:='Vale';
  ShowMessage ('Hola mundo');
end;
```

- Compilar y probar el programa: Pulsa el botón **Run** ▷ o elige del menú: Run - run.
  Comprueba su funcionamiento correcto cerrando la ventana.
- Guardar el proyecto y los archivos fuentes en *Delphi*
  1º. Abre la carpeta de tus projectos y crea la subcarpeta: Holamundo
  2º. Para guardar la ventana del formulario y su código asociado elige del menú: *File* ▸ ***Save all***
  - El archivo que contiene el código en pascal se llamará ***Hola1.pas***
  - El archivo que contiene el proyecto se llamará ***Hola*** *(*.dpr o *.pdroj)*
- Cerra todo el proyecto: Elige del menú: ***File*** ▸ ***Close all...***

Crear la aplicación en C++ Builder:

- Escoge del menú: File ▸ New ▸ *VCL Form Applicaton* – ***C++ Builder*** para crear un nuevo proyecto en C++ Visual.
- Repite los mismos pasos para añadir desde la ***Tool palette*** un botón T***Button*** sobre el formulario (*Button1).*
- Desde el panel de ***Properties***, cambia su propiedad ***Caption*** por: *Aceptar*
- Pulsa *doble clic* sobre el botón e introduce el código (en el evento ***OnClic*** del botón *Button1*):

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{                                          //Cuidaddo:  C++ distingue mayúsculas
  Button1->Caption="Vale";                 //cambia el texto del botón
  ShowMessage ("Hola mundo");              //muestra un mensaje
}
```

- Prueba la aplicación pulsando en el botón ***Run*** ▷ o la tecla: F9
- Si funciona correctamente, guarda la aplicación: *File* ▸ ***Save all***
  - El archivo que contiene el código se llamará ***Hola1.cpp***
  - El archivo cabecera: ***Hola1H.h***
  - El archivo del proyecto: ***Hola.cbproj***

Comparartiva con Microsoft Visual Estudio

Mismo ejercicio en Microsoft C#:

```
private void button1_Click(object sender, EventArgs e)
{
  button1.Text = "Vale";
}
```

Mismo ejercicio en Microsoft Visual C++ :

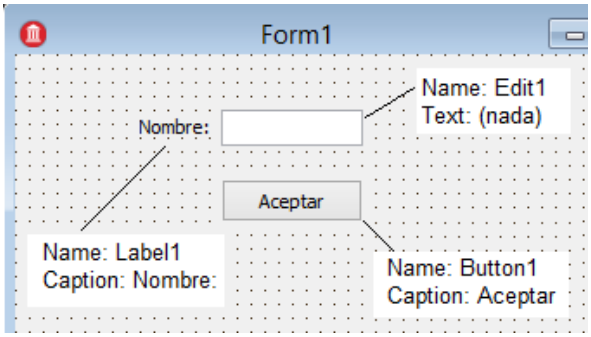
```
private void button1_Click(object sender, EventArgs e)
{
  button1->Text = "Vale";
}
```

Mismo ejercicio en Microsoft Visual Basic:

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
   Button1.Text = "Vale"
End Sub
```

Ejercicio 2:

- Abre el proyecto anterior. En Delphi o C++ usando la opción del menú: *File* - ***Open Project…***
- Desde la ***Tool palette***, añade al ejercicio anterior un componente T***Edit*** y un componente T***Label*** con las propiedades de la imagen:



En Delphi	***En C++***

Añadir al OnClick del botón Button1:
`procedure TForm1.Button1Click(Sender: TObject);`
`begin`
`ShowMessage('Hola, ' + Edit1.Text + '.');`
`end;`
	Añadir al OnClick del botón Button1:
	`void __fastcall TForm1::Button1Click(TObject *Sender)`
	`{`
	`ShowMessage("Hola, " + Edit1->Text + ".");`
	`}`

Guardar los proyectos:

Al guardar el proyecto, se deben guardar antes varios archivos que contengan el código y el diseño. Por eso es importante guardarlo todo en una carpeta.

- En la red: Escoje y abre tu carpeta de alumno de la red
- En el equipo: Abre la carpeta: Documentos\Embarcadero\Studio\Projects
  crea dentro una subcarpeta con tu nombre

Crea la carpeta: *SaludoNombre*

En Delphi:

- Primero te pedirá la unidad del tipo (*.pas): Pon el nombre: *DSaludoNombre1*.pas
- Luego te pedirá el nombre del proyecto (*.dproj): Pon el nombre: *SaludoNombre*

Se generarán otros archivos como *SaludoNombre1*.obj que es el archivo objeto con el código máquina y *SaludoNombre1*.dfm que es el archivo de diseño del formulario.

En C++ Builder:

- Primero te pedirá la unidad del tipo (*.cpp): Pon el nombre: *SaludoNombreC1*.cpp
- Luego te pedirá el nombre del archivo cabecera: Pon el nombre: *SaludoNombreC*.h
- Luego te pedirá el nombre del proyecto (*.cbproj): Pon el nombre: *SaludoNombreC*

Se generarán otros archivos como *SaludoNombreC1*.h que es el archivo cabecera de la unidad y *SaludoNombreC1*.dfm que es el archivo de diseño del formulario.

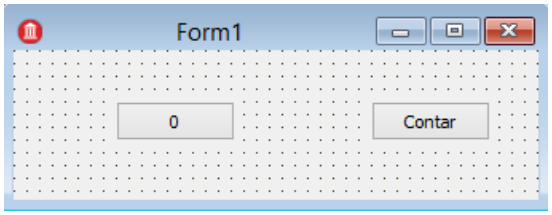
Esta técnica te servirá para guardar los siguientes proyectos y ejercicios.

Ejercicio 3. Contador: *Objetivo*: practicar con la conversión de datos y las propiedades

Recuerda la conversión de tipos de datos:

IntToStr (Integer to String) ▸ Función que cambia un valor del tipo numérico entero a cadena de texto
StrToInt (String to Integer) ▸ Función que cambia un valor del tipo cadena de texto a número entero

- Recupera el ejercicio anterior.
- Cambia el texto (propiedad caption) del botón por: **0**
- Añade un segundo botón a la ventana formulario desde la paleta ***Tool palette*** – sección Standard: Tbutton
  Al botón 2 cambia el texto (propiedad caption) por Contar
  Pulsa doble clic encima de él, para abrir la ventana de código.



En Delphi	*En C++*

Añadir al OnClick del botón Button2:
`procedure TForm1.Button2Click(Sender: TObject);`
`begin`
`  Button1.Caption:=InttoStr(StrToInt(Button1.Capt`
`  ion) +1);`
`end;`
	Añadir al OnClick del botón Button2:
	`void __fastcall TForm1::Button2Click(TObject *Sender)`
	`{`
	`  Button1->Caption = StrToInt(Button1->Caption) +1;`
	`}`

Guarda el proyecto con el nombre: *ContadorDelphi*	Guarda el proyecto con el nombre: *ContadorC*

Variables y constantes:

- ***Una variable*** es un registro o porción de memoria que contendrá un dato o valor y al que le asignaremos un nombre cualquiera. Al crear esta variable le tendremos que especificar qué tipo de dato o valor va a guardar. Este valor puede cambiar a nuestro antojo, en cualquier momento del programa (durante su desarrollo o en ejecución).
- Una **constante** se crea y declara como una variable, pero normalmente, el valor que le asignemos inicialmente no cambiará durante toda la ejecución del programa.
- ***Variables públicas y privadas.*** Si una variable o constante se crea o declara dentro de un procedimiento o función se considera privada de esa función porque sólo servirá o existirá para esa función. Al salir de la misma ya no permanecerá en la memoria (liberamos memoria) y su nombre se podrá emplear en otro lugar, pero no podrá ser utilizada en otro procedimiento o función.
  Si por el contrario la variable o constante es declarada al inicio del programa, antes que los procedimientos o funciones, se considera *pública*. Existirá durante todo el programa y podrá ser utilizada por todas las funciones del programa.

Ejemplo en Delphi:
La variable hay que declararla o crearla antes de ser utilizada con un nombre y su tipo de dato.

```
var
  NumPeque: byte;     //entero hasta 256
  NumEnte: integer;   //entero
  NumDeci: single;    //decimal simple
  NumDeci2: double;   //decimal doble
  Correcto: boolean; //lógico
  Letra: char;        //caracter
const
  Pi = 3.1416;  //supone que es float
```

Ejemplo en C++ Builder:
Se indica el tipo de variable y luego su nombre. Se suele aprovechar la declaración para asignarle ya un valor.

```
byte numPeque=3;   //declara y asigna num hasta 256
int numEnte = 500; // número entero
float numero=3.5;  // declara un numero decimal
boolean correcto=true;
char letra = 'A';  // Declara y asigna el carácter A

const float PI  = 3.1416; //constante tipo decimal
```

Ejercicio 4. Contador con límite

Objetivo: practicar con el estado condicional if y variables

- Recupera el ejercicio anterior y cambia el código anterior por:

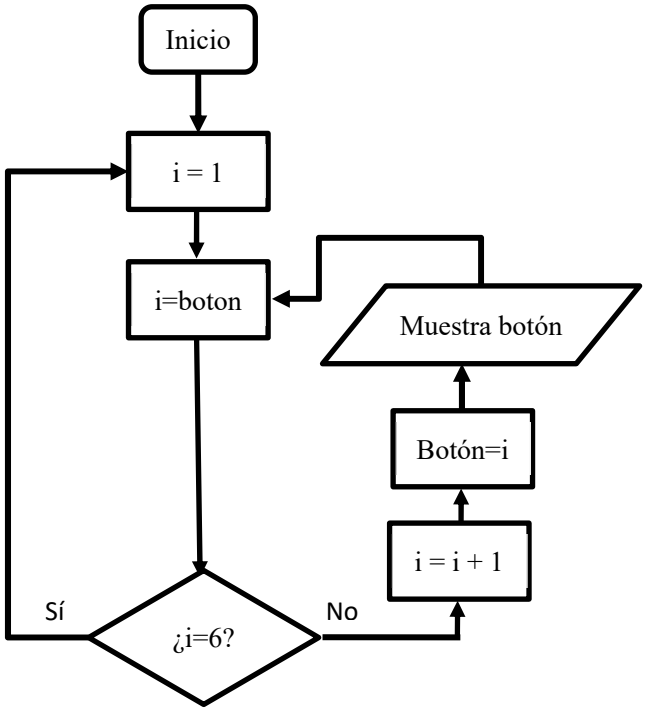
En Delphi/Pascal	***En C++***

`procedure TForm1.Button2Click(Sender: TObject);`
`var  //sección para variables`
`i:integer; //declaro la variable entera i`
`begin`
`i:=strtoint(Button1.Caption);  //pongo el valor`
`if i=6 then i:=1 else i:=i+1;  //si llega a 6…`
`Button1.Caption:=inttostr(i);  //mostramos i`
`end;`
	`void __fastcall TForm1::Button2Click(TObject *Sender)`
	`{`
	`int i;  //declaro la variable i para usar como contador`
	`i=StrToInt(Button1->Caption); //coge el valor del botón1`
	`if (i==6)`
	`{`
	`i=1;  //si ya vale 6 la ponemos a 1`
	`}`
	`else  // y si no….`
	`{`
	`i++;  // le incrementamos 1`
	`}`
	`Button1->Caption = StrToInt(i);  //ponemos i en el botón1`
	`}`

Teoría: Diagrama de flujo o flujorama

Es la represenación gráfica o esquemática de la secuencia que realiza un algoritmo o subrutina
No depende del lenguaje de programación.

Ejemplo de diagrama de flujo del ejercicio anterior ➔



Explicación del algoritmo:

Creamos una variable llamada ***i*** que contendrá un valor numérico entero y que usaremos como contador.

Al pulsar el botón2, la variable ***i*** cogerá el valor que está mostrando el botón1 (que debe ser un número) y este valor de ***i*** lo comaparemos con el 6 en la instrucción IF.

Si ***i*** aún no ha llegado a 6 le sumamos 1 (i=i+1) o le incrementamos 1 (i++).

Si ***i*** ya ha llegado a 6 le reasignamos el valor 1.

Guarda los proyectos
Guarda el proyecto *ContadorDelphi* y el proyecto *ContadorC*.

Ejercicio 5. Generar un número aleatorio

- Recupera el ejercicio anterior.
- Añade un tercer botón a la ventana formulario desde la paleta ***Tool palette*** – sección Standard: Tbutton
- Al botón 3 cambia el texto (propiedad caption) por Aleatorio y pulsa doble clic encima l, para poner el código.

En Delphi:

```
procedure TForm1.Button3Click(Sender: TObject);
begin
  randomize;  // llamada a procedimiento
  Button1.Caption:=inttostr(random(5)+1);
end;
```

En C++ Builder:

```
void __fastcall TForm1::Button3Click(TObject Sender)
{
randomize;
Button1->Caption=IntToStr(random(5)+1);  }
```

Guarda los proyectos: Guarda el proyecto *ContadorDelphi* y el proyecto *ContadorC*.

Intrucciones condicionales

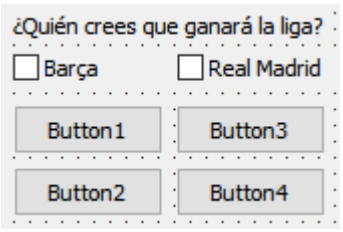
- ***If***

  La instrucción *if* puede usarse para ejecutar una instrucción sólo si se cumple una cierta condición (*if-then* ; si...entonces), o para elegir entre dos alternativas (*if-then-else*; si...entonces..., y, si no, ...)

- **Case - Swith**

  Case permite evaluar distintas condiciones según un valor ordinal, en una lista de posibles valores.

Ejercicio condicionales



- Crea una aplicación nueva (File ▸ New ▸ VCL Form Applicaton) y coloca los siguientes ingredientes:
  - Un Label – con el caption: ¿Quién crees que ganará la liga?
  - Dos casillas de selección: Con el caption: Barça y Real Madrid
  - Cuatro botones en el formulario.
- Haz doble clic en cada botón para añadir código al evento *OnClick*:

  Nota: La instrucción *if-then-else*, es una sola instrucción, así que no puede colocar un punto y coma en medio de ella.

En Delphi/Pascal:

```
procedure TForm1.Button1Click(Sender: TObject);
begin  // instrucción if
  if CheckBox1.Checked then
    ShowMessage ('Has activado Barça')
end;

procedure TForm1.Button2Click(Sender: TObject);
begin   // instrucción if-else

  if CheckBox2.Checked then
    ShowMessage (Has activado Real Madrid')
  else
    ShowMessage ('NO has activado Real Madrid);
end;

procedure TForm1.Button3Click(Sender: TObject);
begin
  // instrucción de doble condición
  if CheckBox1.Checked and CheckBox2.Checked
then
  ShowMessage (Has activado los dos equipos')
end;

procedure TForm1.Button4Click(Sender: TObject);
begin
  // instrucción if compuesta
  if CheckBox1.Checked then
    if CheckBox2.Checked then
    ShowMessage('Has activado los dos equipos')
    else
      ShowMessage('Sólo la casilla 1 está
activada')
  else
    ShowMessage (
      'La casilla 1 no está activada. ¿A quién
le importa la 2?')
end;
```

En C++ Builder:

```
 void __fastcall TForm1::Button1Click(TObject *Sender)
 {
   if (CheckBox1->Checked) {
   ShowMessage ("Has activado Barça");
   }
 }
 //----------------
 void __fastcall TForm1::Button2Click(TObject *Sender)
 {
   if (CheckBox2->Checked) {
       ShowMessage ("Has activado Real Madrid");
       }
   else
       {ShowMessage ("NO has activado Real Madrid");
       }
 }
 //--------------------
 void __fastcall TForm1::Button3Click(TObject *Sender)
 {
 if (CheckBox1->Checked && CheckBox2->Checked) {
       ShowMessage ("Has activado los dos equipos");
       }
 }
 //-------------------
 void __fastcall TForm1::Button4Click(TObject *Sender)
 {
   if (CheckBox1->Checked) {
       if (CheckBox2->Checked) {
         ShowMessage("Has activado los dos equipos");
       }
       else
       { ShowMessage("Sólo la casilla 1 está
       activada");
       }
       }
   else
       { ShowMessage("La casilla 1 no está activada.
       ¿A quién le importa la 2?");
  }
}
```

Guarda los proyectos

Guarda el proyecto CondicionalDelphi y el proyecto CondicionalC.

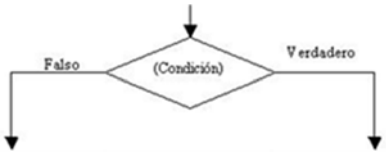
Estructura condicional. Toma de decisión

En Pascal:

```
if comparación then acción_verdadera
else accion_falsa ;
```

En C++:

```
if (comparación) { acción_verdadera; }
else { accion_falsa; }
```



Realizará una acción u otra según si cumple una comparación

Asignar o comparar un valor en Delphi/Pascal.

- Para asignar un valor a un objeto o variable se indica con **:=** Ejemplo: `ProgressBar1.position := 100;`
- Para comparar el valor de un objeto o variable se indica con **=** Ejemplo: `if ProgressBar1.position = 100..`

Asignar o comparar un valor en C++.

- Para asignar un valor a un objeto o variable se indica con **=**
- Para comparar el valor de un objeto o variable se indica con **==** Otros comparadores son:

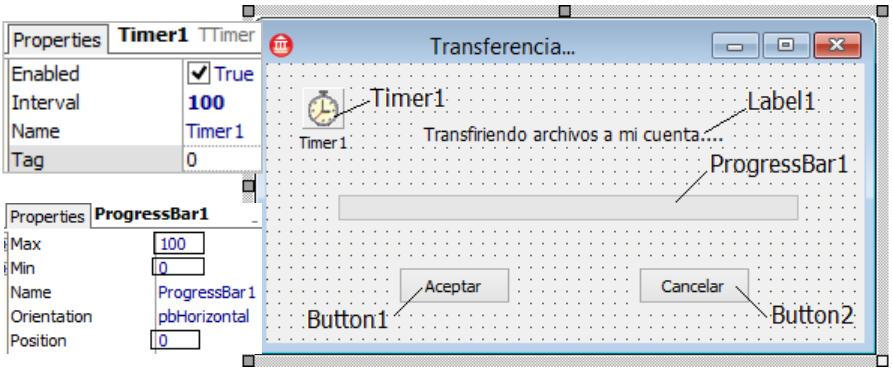
Comparador	*símbolo*	*ejemplo*
Mayor	>	if ProgressBar1.position>100 then Showmessage ('Te has pasado de 100');
Menor	<	if ProgressBar1.position>100 then Showmessage ('Aún no has llegado a 100);
Mayor o igual	>=	if edad>=100 then Showmessage ('Has llegado o te has pasado de 100');
Menor o igual	<=	if edad<=100 then Showmessage ('Aún no has llegado a 100');
Distinto	< > ó !=	if edad <> 100 then Showmessage ('No es 100');

Ejercicio ventana con barra de progreso (hacer en modo Windows VCL y modo Multidevice FMX)

- Crea un nuevo proyecto para Windows: File ▸ New ▸ Windows VCL Application (Delpphi o C++ Builder).

Diseño:

- Añade al formulario un objeto barra de progreso *tProgressBar* (de la paleta win32), dos botones (*tButtons*), una etiqueta *tLabel* (de la paleta *Standard*) y un componente *tTimer* (Paleta System)
- Distribuye y cambia las propiedades de la ventana-formulario como en la imagen.



- *Propiedad* ProgessBar1.Max = 100 ProgessBar1.Min = 0 y ProgessBar1.Position = 0
- *Propiedad* Timer1.Interval = 500 (medio segundo)

Objetivo del código:

Queremos que a cada impulso del Timer, mover la posición de la barra de 1 en 1:

→ `ProgressBar1.position := ProgressBar1.position+1;`

y cuando la barra llegue a 100 queremos que se pare:

→ `if ProgressBar1.position=100 then ...`

Pero… ¿Qué pasaría si los incrementos fuesen de 3 en 3?

Eventos: (Suceso que desencadena una acción)

- Pulsa *doble clic* sobre el Timer para activar el evento ***OnTimer*** del ***Timer1*** y escribe el código siguiente según la versión escogida:

Versión Delphi VCL:


```
procedure TForm1.Timer1Timer(Sender: TObject);
begin
   ProgressBar1.position := ProgressBar1.position+1; //-> incrementa la barra una posición
   if ProgressBar1.position=100 then                //-> si llega a 100…
    begin
      Timer1.Enabled:=False;                        // -> Para el temporizador
      ShowMessage('Transferencia realizada');       // -> muestra un mensaje
      Application.Terminate;                        //-> termina el programa
   end;
end;
```

Versión C++Builder VCL:

```
void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
ProgressBar1->Position++;                           //-> incrementa la posición de la barra
if (ProgressBar1->Position==100)                    //-> si llega a 100…
   {
     Timer1->Enabled=False;                         // -> Para el temporizador
     ShowMessage("La aplicación  " + Application->ExeName+ " ha finalizado");  // -> muestra un mensaje
     Application->Terminate;                        //-> termina el programa
   }
}
```

Versión Delphi MultiDevice (Firemonkey fmx):

```
procedure TForm1.Timer1Timer(Sender: TObject);
begin
ProgressBar1.Value := ProgressBar1.Value+1; //-> incrementa la barra una posición
if ProgressBar1.Value=100 then              //-> si llega a 100…
   begin
    Timer1.Enabled:=False;                  // -> Para el temporizador
    ShowMessage('Transferencia realizada'); // -> muestra un mensaje
    Application.Terminate;                  //-> termina el programa
  end;
end;
```

Versión C++ Builder MultiDevice (Firemonkey fmx):

```
void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
ProgressBar1->Value++;                      //incrementa la posición de la barra
if (ProgressBar1->Value == 100)             //-> si llega a 100…
 {
   Timer1->Enabled=False;                   // -> Para el temporizador
   ShowMessage("Transferencia realizada");  // -> muestra el mensaje
   Application->Terminate;                  //-> termina el programa
 }
}
```

Opcional: Puedes impedir cerrar la ventana si en el evento del form: Closequery pones la variable canClose:=false;
Ejemplo: ShowMessage('No puedes cerrar la ventana mientras se transfieren los archivos'); canClose:=false;

¿Qué pasaría si los incrementos fuesen de 3 en 3?

- ***Probar la aplicación***: Pulsa Run ▷ para comprobar su funcionamiento.
- ***Guardar el proyecto***: Crea una carpeta en tus *Projects* que se llame: *Transferencia*

Para Delphi VCL

- Escoge*: File ▸ Save as... Para guardar el archivo: Transferencia1.pas*
- Escoge*: File ▸ Save project as... Para guarda el proyecto Transferencia.dpr*

Para C++Builder VCL

- Escoge*: File ▸ Save as... Para guardar el archivo: Transferencia1.cpp* (Se crea Transferencia1.h)
- Escoge*: File ▸ Save project as... Para guarda el proyecto Transferenciah.h* y *Transferenciah.cproj*

Bucles:

Infinito:

While: Repite la ejecución infinitas veces hasta que se cumpla la condición.

```
while i<10  do i := i+1;
```

Finito (Contador):

For ... To, *o For ... Downto:*

Ejecuta una acción un número determinado de veces.

- *To*: Incrementa cada vez el bucle
- *Downto*: Decrementa cada vez el bucle.

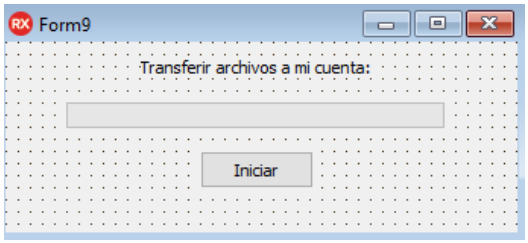
```
for i:=1 to 100 do
    Begin
      Label1. caption:='Paso número' + i;
      Beep; // tono sonoro
    End;
```

Ejercicio Progreso sin timer (con while)

- Crea un nuevo proyecto para Windows: File ▸ New ▸ Windows VCL Application (Delpphi o C++ Builder).

Diseño:

- Añade al formulario un objeto barra de progreso ***tProgressBar***
- Un botón (***tButton***) y una etiqueta ***tLabel*** (de la paleta *Standard*)
- Distribuye y cambia las propiedades de la ventana-formulario como en la imagen.



Eventos: (Suceso que desencadena una acción)

- Pulsa *doble clic* sobre el Botón y escribe el código siguiente según la versión escogida:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
   while ProgressBar1.Position<100 do
  begin
    ProgressBar1.Position := ProgressBar1.Position+1;
    sleep(200);
  end;
  ShowMessage('Transferencia realizada');   // -> muestra un mensaje
  Application.Terminate;     //-> termina el programa
end;
```

Probar la aplicación: Pulsa Run ▷ para comprobar su funcionamiento.

Ahora cambia el Código por este otro y comenta la diferencia:

```
procedure TForm1.Button1Click(Sender: TObject);
var
i:integer;
begin
   for i:=1 to 100 do
  begin
    ProgressBar1.Position := ProgressBar1.Position+1;
    sleep(200);
  end;
  ShowMessage('Transferencia realizada');   // -> muestra un mensaje
  Application.Terminate;     //-> termina el programa
end;
```