

Curso de Free Pascal - Básico 1

Tema: Introducción a la herramienta Lazarus

Prof.: Lic. Ricardo De Castro Aquino (Otoño de 2015)

Actualizado al: 16/04/2015 19:57



Tópicos principales:

I – Introducción

II – Tutorial

1. [Creando el proyecto inicial – Formulario con un botón](#)
2. [Creando el proyecto con dos botones](#)
3. [Modificando nombres y propiedades de componentes](#)
4. [Aplicación con Edit y Check Box](#)
5. [Utilización del evento OnChange de la clase TEdit](#)
6. [Utilización de procedimientos y escritura rápida](#)
7. [Navegación por el código y formulario/código](#)
8. [Aplicación con RadioGroup y Memo](#)
9. [Aplicación con ComboBox](#)
10. [Temporizador](#)
11. [Menú de la Aplicación](#)
 - [Cómo cambiar el idioma de las constantes de mensajes de Lazarus](#)
12. [Eventos con Parámetros](#)
13. [Confirmar salida de la aplicación – Close Query](#)
14. [El Depurador de código – Debugger](#)
15. [Comentarios finales](#)

III – Ejercicios de Lazarus

1. ["Hola mundo" en un botón que se mueve](#)
2. [Edit, Botón y Memo](#)
3. [Cálculo de operación elegida por RadioGroup \(resultado en Edit Box\)](#)
4. [Cálculo de operación elegida por RadioGroup \(descrita en Memo Box\)](#)
5. [Diseño básico \(canvas\)](#)
6. [Aplicación multi-ventana](#)

IV - Conexión de Lazarus a una Base de Datos PGSQL

1. [Diferentes formas de uso](#)
2. [Preparación](#)
3. [Acceso utilizando DBGrid](#)

4. [Ejecución directa de código usando el TConnection](#)
5. [Utilizar las consultas SQL \(Fields\)](#)
6. [Dejando al SQLQuery la tarea de manejar los datos](#)
7. [Ejercicio libre](#)
8. [Mantenimiento práctico enlazado por controles visuales](#)
 - [Usando el TDBEdit para editar campos individuales](#)
 - [Implementando una barra de navegación funcional](#)
 - [Solucionando el problema de los Hints de TDBNavigator](#)
 - [Usando el TDBLookupCombobox para ver y editar claves foráneas](#)

[V – Algunos comandos SQL \(micro resumen\)](#)

1. [Manejo de tablas](#)
2. [Manejo de datos](#)
3. [Consultas de datos \(queries\)](#)

[VI – FAQ básica de Lazarus sobre temas afines](#)

1. [¿Porqué Lazarus?](#)
2. [Lazarus y POE](#)
3. [Lazarus debugger, inestabilidades y errores raros](#)
4. [Lazarus, PostgreSQL y errores relacionados a lippq.xxx](#)
5. [Error "Circular Unit Reference"](#)
6. [Cadenas dinámicas \(‘ comillas dentro de “comillas” ’\)](#)
7. [Consultas que no funcionan y Esquemas y/o Relaciones \(tablas\) que no se pueden acceder](#)
8. [Passwords \(contraseñas\)](#)
9. [¿Cómo crear informes con Lazarus?](#)
10. [¿Qué es un paquete Lazarus?](#)
11. [¿Cómo reducir el tamaño de los ejecutables?](#)
12. [Archivos que pueden ser borrados](#)

[VII – Listado de teclas y funciones](#)

I - INTRODUCCIÓN

"*Lazarus*" es una herramienta de programación tipo RAD (Rapid Application Development) potente y compleja, que utilizaremos como ambiente integrado de desarrollo (IDE) para el lenguaje **Free Pascal (FPC)**. Esta herramienta es completamente "cross platform" y puede ser instalada en cualquier máquina y en casi todos los sistemas operativos existentes en la actualidad. Es decir, se trata de un **Compilador (L)GPL** que se ejecuta en Linux, Win32/64, OS/2, 68K, MacOS y más.

Este tutorial introductorio abordará apenas el IDE de Lazarus y la introducción a la "Programación Visual", Programación Orientada a Eventos (POE). Es decir, no entraremos en mayores detalles sobre el lenguaje de programación en sí (FPC). Esto deberá ser visto en un módulo aparte.

Debido a que en sus últimas versiones el instalador de *Lazarus* es bastante intuitivo, aquí no se ha incluido información de instalación y además se considera que el alumno ya dispone de una versión instalada para trabajar. Sin embargo, si usted desea obtener el instalador de la herramienta, puede acceder al sitio oficial de descargas: <http://sourceforge.net/projects/lazarus/> y descargar la última versión para el sistema operativo que desee.

Este documento fue confeccionado como un instructivo de apoyo a la presentación del IDE y de las ideas claves que permitirán al alumno posteriormente continuar su aprendizaje con mayor autonomía.

La estructura del mismo está basada principalmente en el tutorial llamado "Introdução à ferramenta Lazarus", creado en Mayo del año 2010, por Armando de Sousa para la "Faculdade de Engenharia da Universidade do Porto - FEUP". Muchas gracias a él por dicho trabajo. Sin embargo hay otras partes sacadas de varias fuentes de la web, principalmente de las [wiki de FreePascal/Lazarus](#).

Para la confección de este tutorial, se usó la versión **1.2.6** de Lazarus sobre Windows 7, así que, si usted está usando otra versión o Sistema Operativo, la apariencia de las figuras que ilustran las acciones y resultados puede diferir un poco (no mucho) de lo que obtengas.

A lo largo de este documento, he utilizado siempre que posible los términos en castellano para varios eventos muchas veces referidos y conocidos por su terminología en inglés. Por ejemplo: tiempo de diseño (design-time), tiempo de ejecución (run-time), casilla de verificación (check box), cadena (string), etc.

Es importante que, cuando se lea por primera vez este tutorial, no se omita los pasos ni se aparte demasiado de los pasos mencionados. El alumno puede y debe repetir la "hoja de ruta" una vez que haya obtenido una mejor percepción del funcionamiento en conjunto de todos los elementos involucrados y, en esa fase, dedicarse a una exploración más amplia de las posibilidades ofrecidas por la herramienta.

Siempre que necesario, no olvide consultar el capítulo final "[Listado de teclas y funciones](#)".

¡Buena suerte!

Lic. Ricardo De Castro Aquino
Asunción - Paraguay

Sitio oficial de Lazarus FPC: <http://www.lazarus.freepascal.org/>

II - TUTORIAL

1. Creando el proyecto inicial – Formulario con un botón.

En la mayoría de los IDEs de programación visual, los objetos que se crean para las ventanas de la aplicación son llamados de formularios. Aquí, crearemos un formulario simple con un botón, utilizando el *Lazarus*.

1.1 – Arranque su PC con el Sistema Operativo (S.O.) de su preferencia y luego ejecute el *Lazarus*.

Ejemplo Windows: Botón de inicio -> Todos los programas -> Lazarus -> Lazarus

1.2 – Si lo estamos ejecutando por primera vez, *Lazarus* ya nos creará un proyecto en blanco llamado temporalmente de "project1". Si no, nos abrirá el último proyecto en el que habíamos trabajado. Como sea, dentro de *Lazarus* seleccione la opción de menú Proyecto -> Nuevo Proyecto... y aparecerá la ventana de diálogo que vemos en la **Figura 1**.

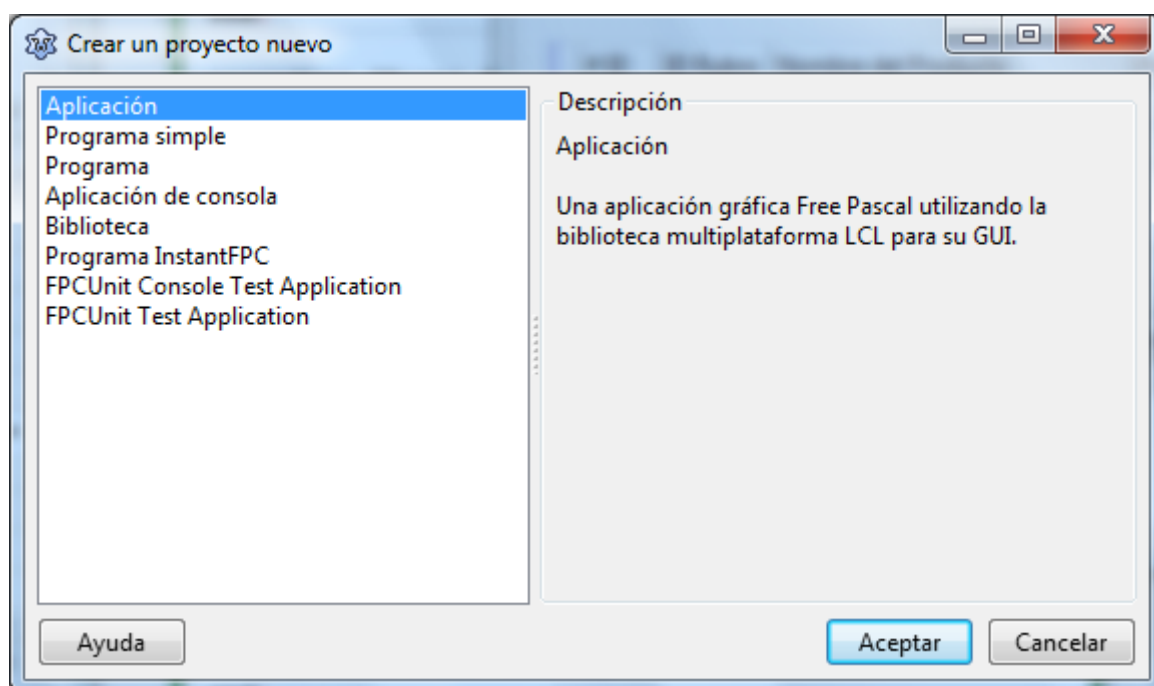


Figura 1: La ventana para creación de un Nuevo Proyecto.

1.3 – Seleccione **Aplicación** y oprima [Aceptar]. Como teníamos un proyecto abierto por default al arrancar Lazarus, la herramienta nos pedirá confirmación sobre qué debe hacer con el mismo, por medio de la ventana de diálogo mostrada en la **Figura 2**. Como lo que queremos es crear un nuevo proyecto, haga clic en el botón [No]

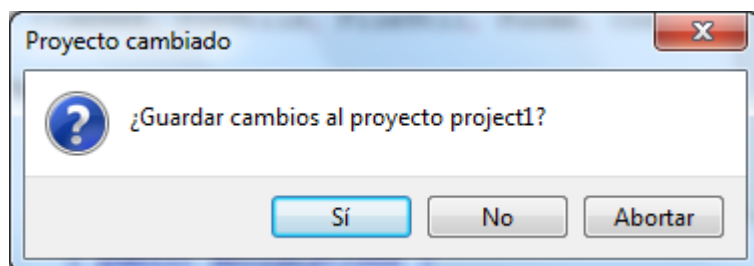


Figura 2: Diálogo de confirmación sobre guardar cambios al proyecto abierto.

Nada más que con estos pasos hemos creado un nuevo proyecto en blanco para desarrollar una nueva aplicación gráfica.

1.4 – El ambiente integrado de desarrollo (IDE) que aparece, está constituido por diversos componentes, cada cual en su ventana, como se muestra en la **Figura 3**.

- Arriba, la barra de menús y la **Paleta de Componentes**.
- A la izquierda, el **Inspector de Objetos** que se compone de un cuadro de lista del tipo árbol y de un marco de páginas con las Propiedades, Eventos, etc.
- Al centro, el **Editor de Código Fuente** que es donde se escribe el código, cuando necesario.
- En la **Figura 3** se muestra a la derecha el formulario "Form1" en tiempo de diseño (design-time).
- Bien abajo, aparece la ventana de mensajes que es dónde van a aparecer los errores de compilación, etc.
- Inicialmente, las ventanas no tendrán el aspecto mostrado en la Figura 3. Modifique las posiciones y dimensiones de las mismas para reflejar un aspecto similar al mostrado. Sin embargo, puede ajustar el aspecto de cada una para adaptarlas a su gusto particular. Cuando vuelva a abrir este proyecto, *Lazarus* recordará de sus preferencias y las traerá como se las dejaste por última vez.

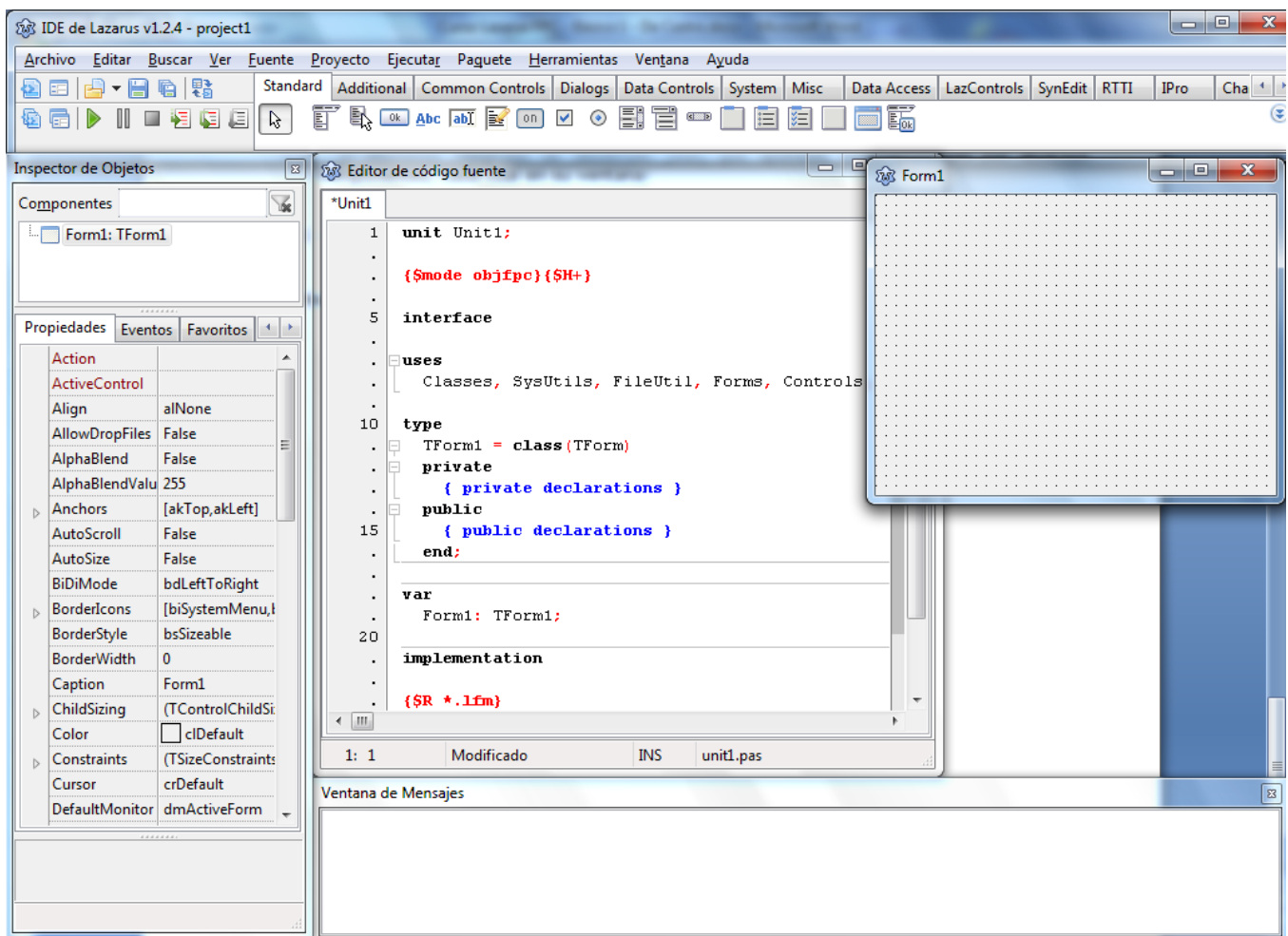


Figura 3: Entorno de desarrollo integrado de *Lazarus*.

1.5 – **Agrega un botón al formulario.** Para esto:

1.5.1 – Haga clic en el componente TButton, que está situado en la pestaña “Standard” de la Paleta de Componentes.

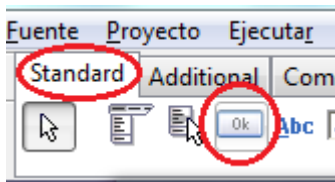


Figura 4: Seleccionando el componente TButton

1.5.2 – Haga clic en el formulario. Aquí, si deseamos, podemos utilizar la tecla de arrastre del mouse para dibujar el botón en un tamaño y forma específico.

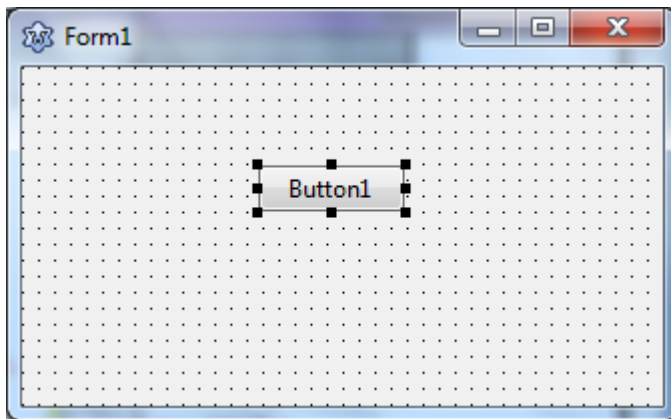


Figura 5: Agregando un botón al formulario.

1.6 – El botón que acabas de crear fue automáticamente nombrado como *Button1*. Haga un doble-clic sobre él y será transportado al código a ejecutar cuando se pulsa el botón (dentro del *Editor de Código Fuente*).

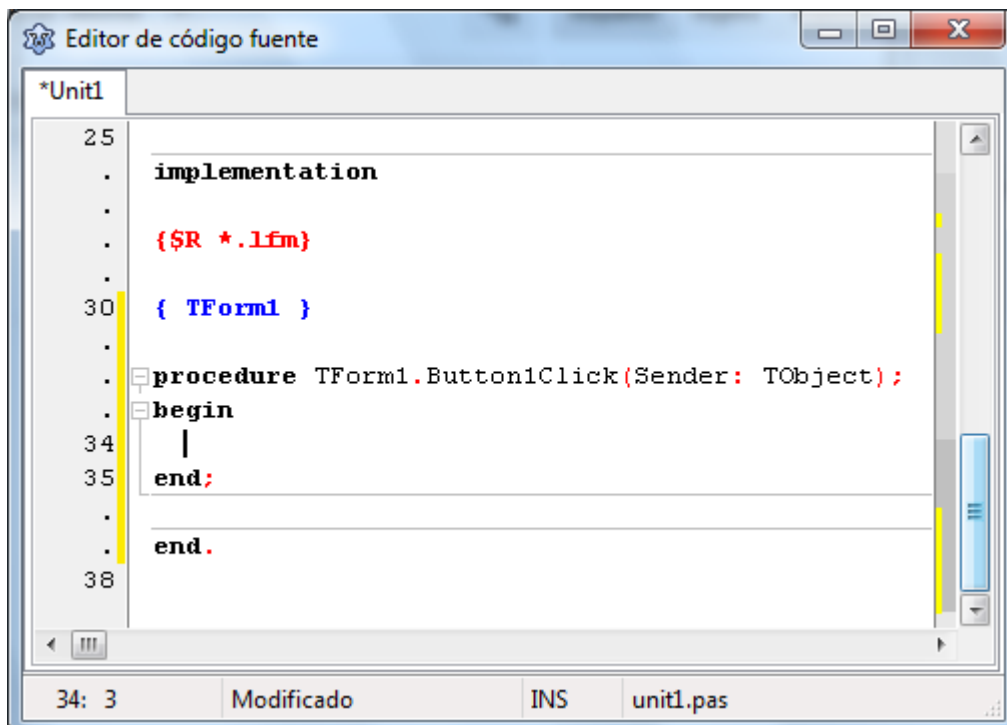


Figura 6: Editando el evento *OnClick* del botón *Button1* recién creado.

Como lo que estamos haciendo aquí es, en realidad, editar el evento **OnClick** de dicho botón, otra forma de acceder al mismo punto es seleccionando ese evento desde el "**Inspector de Proyectos**", en la pestaña "**Eventos**".

1.7 – Después del **begin** y antes del **end**; escriba el código:

```
Button1.Caption := 'Hola mundo...'
```

Este código modifica el texto asociado al botón para que este pase a ser "Hola mundo...". El botón se llama *Button1* y el texto dentro del mismo le pertenece y es accesible por medio de la propiedad *Caption* del mismo. Para asignar una cadena de caracteres al texto dentro del botón, se utiliza el código mencionado.

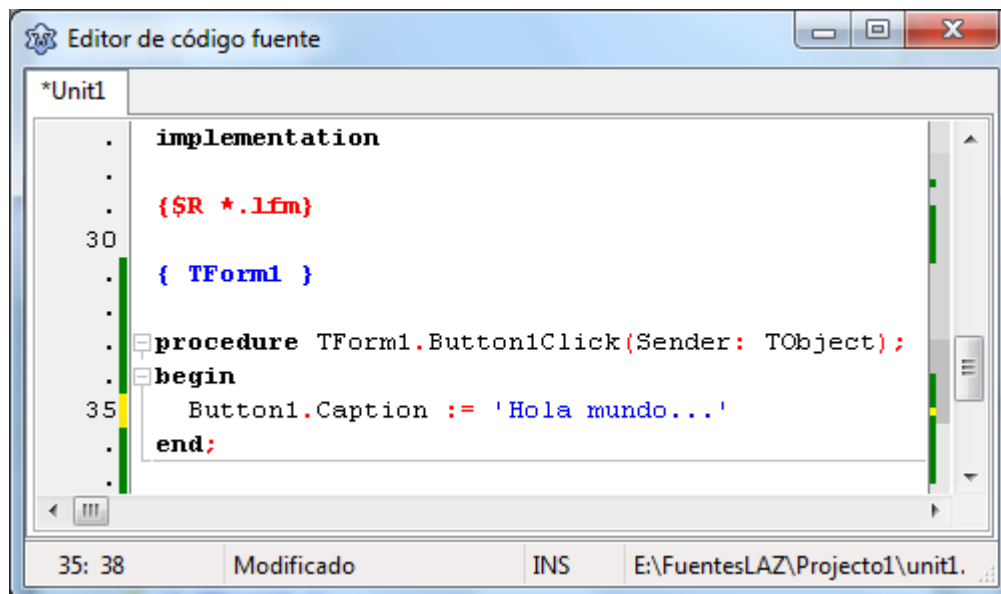


Figura 7: Código de asignación de un texto al *Caption* de *Button1*.

La asignación se hace por medio del operador **:=** y la cadena de caracteres es identificada por estar entre 'comillas simples' (atención, no usar "comillas dobles"). En el código, se puede utilizar mayúsculas o minúsculas sin problemas. El aspecto en la IDE será aproximadamente el que se muestra en la **figura 7**.

El código que acabas de escribir quedará guardado dentro de una "Unit" del FPC.

1.8 – Elija en el menú Archivo -> Guardar todo para guardar todos los archivos en una carpeta creada para tal efecto, en cualquier localización del disco local (evite utilizar unidades de red pues la creación del ejecutable se vuelve muy lenta). Acepte los nombres predeterminados (default) en minúsculas.

1.9 – Ejecute el programa oprimiendo la tecla **F9** o por medio del menú Ejecutar -> Ejecutar y observe las emisiones de mensajes en la *Ventana de mensajes*: primero viene la llamada al compilador del FPC; si no hay errores, entonces vendrá el mensaje "linking" señalando el proceso de construcción del ejecutable y por último el mensaje de suceso:

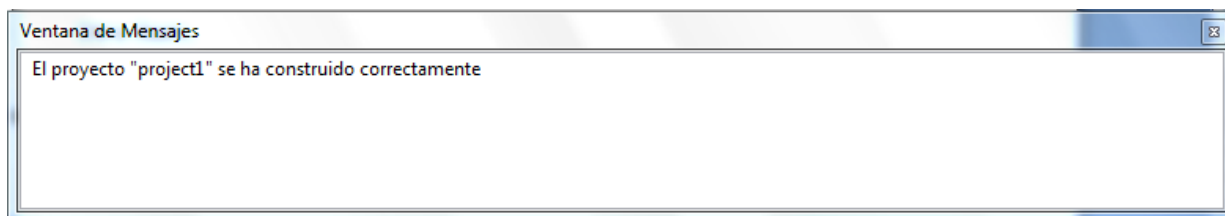


Figura 8: Ventana de mensajes luego de crear el ejecutable.

- 1.10 – El programa que acaba de ser creado es entonces ejecutado por el sistema operativo y aparece la ventana de su primer programa hecho con *Lazarus*.

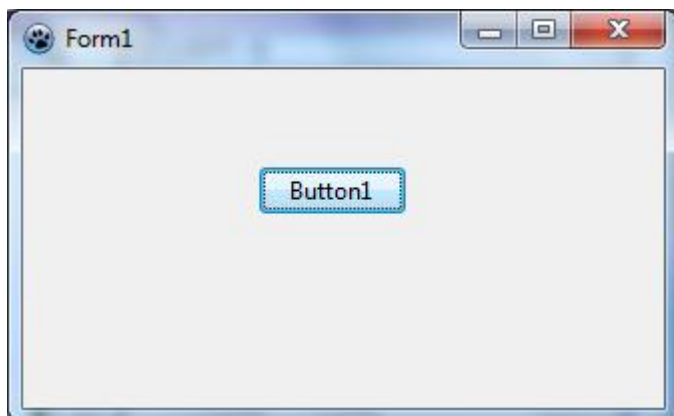


Figura 9: el programa en ejecución.

- 1.11 – Oprima el botón con el mouse y observe que el texto asociado a dicho botón se cambia para:

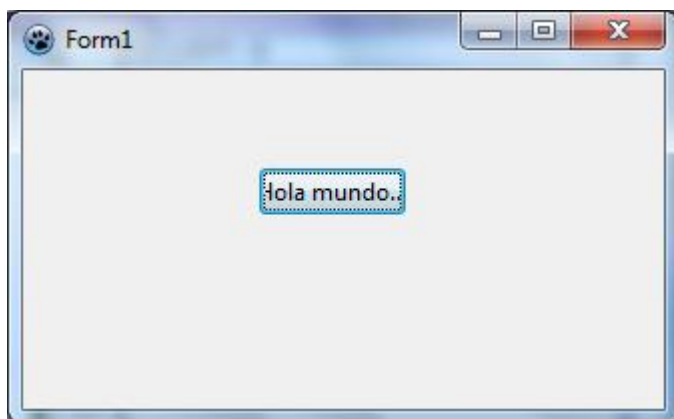


Figura 10: apariencia del botón luego de ser oprimido.

- 1.12 – Cambia el tamaño y mueva la ventana del programa para asegurarse de que esta ventana del programa se comporta como cualquier otra aplicación.
- 1.13 – Cierra la aplicación oprimiendo el botón [X] de cierre en el rincón superior derecho de la ventana del programa que está siendo ejecutado.
- 1.14 – Si nos fijamos en la **Figura 10**, podemos notar que el nuevo texto no cabe completo en el ancho del botón. ¿Cómo podemos solucionar esto? Muy simples:
- 1.14.1 - Haga clic sobre el botón en el Formulario.
- 1.14.2 - Manteniendo oprimida la tecla de Mayúsculas (Shift), utilice las teclas de cursor para redimensionar el botón a un tamaño adecuado.

1.14.3 - Vuelva a ejecutar los pasos desde el 1.8 hasta el 1.11. Si el tamaño sigue siendo inadecuado, repita los pasos de este tópico (**1.14**).

1.15 – Ideas para retener:

- *El Lazarus permite crear aplicaciones gráficas para Sistemas Operativos gráficos y orientados a eventos.*
- *Este proyecto incluye una parte gráfica (el formulario) y una parte de código escrito en FPC (la "Unit").*
- *La programación incluye diseñar el formulario que después será ejecutado y el código que será ejecutado cuando se oprima el botón durante la ejecución del programa.*
- *La ejecución del programa hace aparecer la ventana previamente diseñada en el IDE del Lazarus. Cuando oprimimos el botón, es llamado el código que escribimos en el programa.*
- *La ventana que es diseñada en el IDE del Lazarus, antes de su ejecución por el sistema operativo, es llamada de formulario en tiempo de diseño. Este mismo formulario aparece en tiempo de ejecución (comparar la **Figura 5** con la **Figura 9**).*
- *La aplicación creada es una de las muchas que están siendo ejecutadas por el Sistema Operativo.*
- *Con relación al código FPC, conviene retener:*
 - *Las asignaciones se hacen con el operador ":=*
 - *Para acceder a propiedades, utilizar un punto ".", por ejemplo "Button1.Caption".*
 - *El FPC no es sensible a mayúsculas y minúsculas.*
 - *Las cadenas de caracteres van entre 'comillas simples' y no se debe cambiar de línea dentro de una cadena de caracteres.*

1.16 – Ejercicio libre – Crea y prueba una aplicación con un botón que ejecute el código ShowMessage('¡Hola Mundo!').

2. Creando el proyecto con dos botones.

2.1 – Vuelva al IDE de *Lazarus* para modificar el proyecto anterior.

2.2 – Si el Form1 no está visible, elija Proyecto -> Formularios -> Form1 -> Aceptar.

2.3 – Agrega otro botón a nuestra ventana. Si necesario, muévelo y redimensiona hasta que se alinee con el botón anterior:

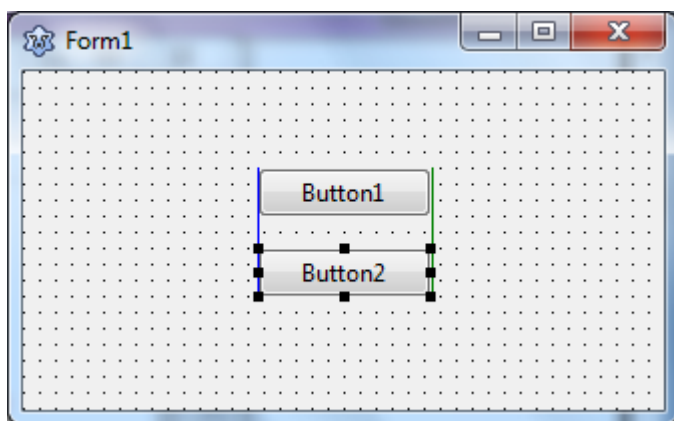


Figura 11: Formulario con los dos botones.

- 2.4 – Observe el Inspector de Objetos y confirme en la lista de Componentes que: nuestra ventana, el “formulario”, tiene el nombre de Form1, el primer botón se llama *Button1* y el segundo *Button2*. Por supuesto que ambos botones están dentro del Form1.

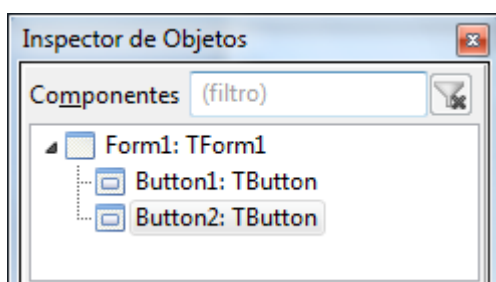


Figura 12: El Inspector de Objetos mostrando los componentes actuales.

- 2.5 – Haga un doble clic en el botón 2. El cursor aparecerá en el *Editor de Código Fuente*, dentro del procedimiento *Button2Click*, procedimiento que será llamado cuando se oprima el botón 2. Allí agrega el siguiente código:
`ShowMessage('¡Hola Mundo!')` ;

La ventana del editor debe parecerse con la de la figura 13 abajo.

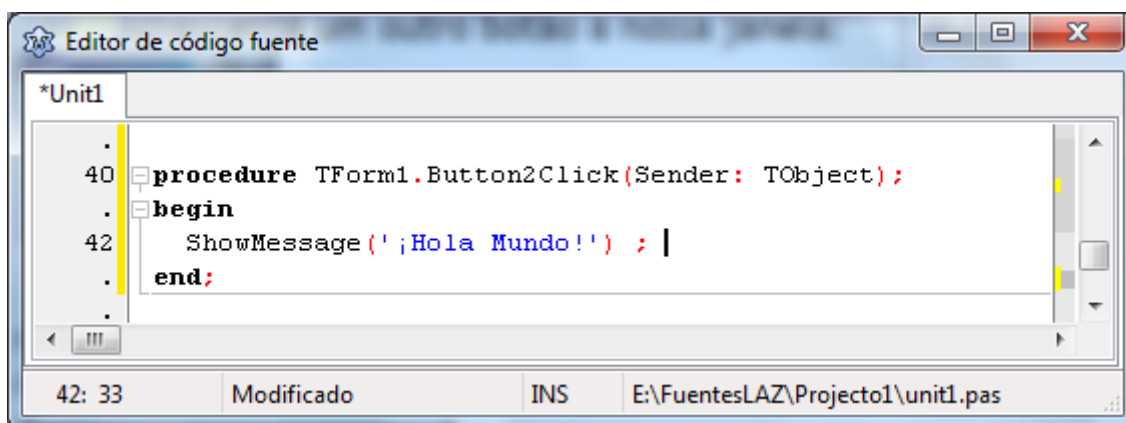


Figura 13: Código para el evento clic del segundo botón.

- 2.6 – Oprima [F9] para compilar y ejecutar el proyecto y, si no hay errores, el mismo será ejecutado, apareciendo el formulario en tiempo de ejecución (run time):

Nota: en versiones anteriores de Lazarus era necesario guardar los cambios antes de compilar. En las últimas versiones esto ya no es tan necesario, ya que al compilar la aplicación (F9) automáticamente se guardan últimos los cambios efectuados. Sin embargo, guardar antes de ejecutar sigue siendo una buena práctica.

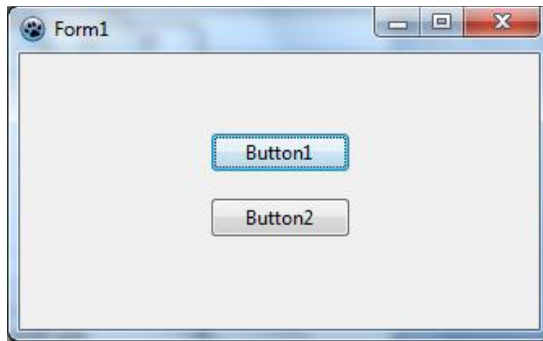


Figura 14: Formulario con dos botones, en tiempo de ejecución.

2.7 – Oprima el botón 1 y confirme que todo sigue funcionando.

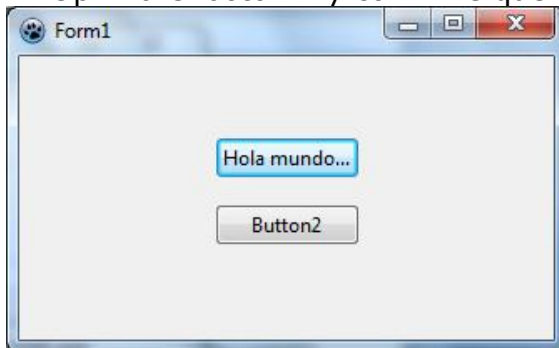


Figura 15: Comprobando el funcionamiento del botón 1.

2.8 – Oprima el botón 2 y vea que aparece el siguiente cuadro de mensajes.

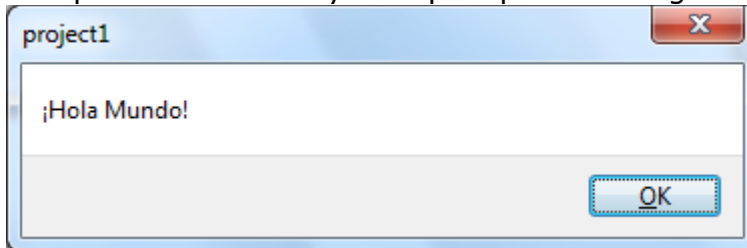


Figura 16: Cuadro de mensajes mostrado por el botón 2.

2.9 – Oprima Ok y luego cierra la aplicación con el botón **[X]**.

2.10 – Regresa al IDE de Lazarus y oprima F12 hasta ver el formulario en tiempo de diseño.

2.11 – Haga un doble clic en el Button1 y el cursor aparecerá en el Editor de código, dentro del procedimiento `Button1Click`; oprima **F12** para alternar entre el editor y el formulario, y luego haga un doble clic en el Button2 para que el cursor aparezca dentro del procedimiento `Button2Click`. Navega en el editor de código hasta encontrar el procedimiento `Button1Click`:

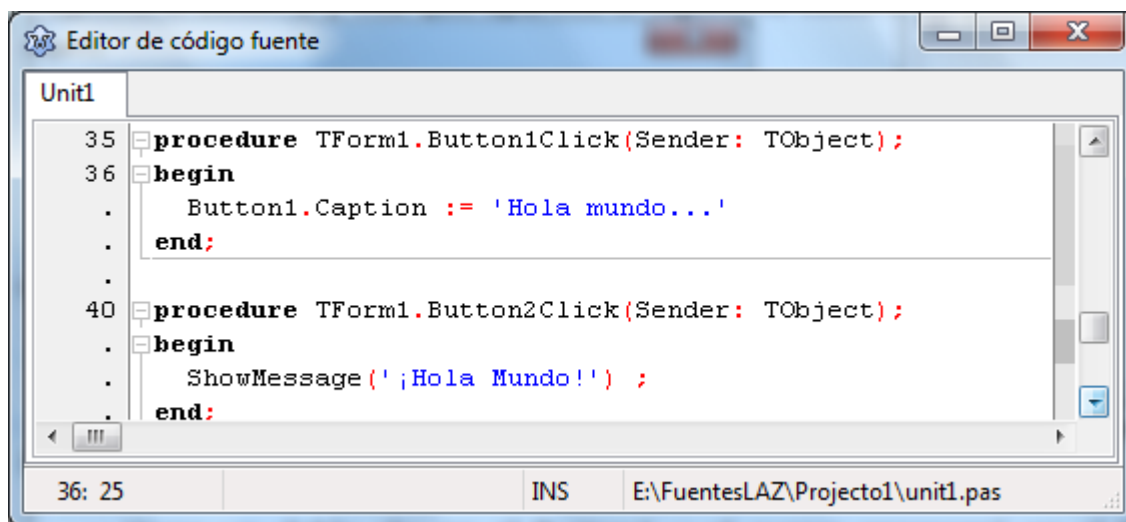


Figura 17: El editor de código fuente con los procedimientos para ambos botones.

2.12 – Ejercicio Libre – Agrega un tercer botón que cuando sea oprimido muestre un diálogo con el mensaje: “Buen día, Brasil.”. Donde “Brasil” puede ser reemplazado por el nombre de tu país de origen.

2.13 – Ideas a retener:

- El formulario del proyecto, llamado *Form1*, pasó a tener dos botones: *Button1* y *Button2*.
- La “*Unit1*” pasó a tener, entre otros elementos de código, los procedimientos *Button1Click* y *Button2Click*.
- Los dos botones son **instancias** de objetos del tipo *TButton*. Se dice que *TButton* es una **clase**. Generalmente, las clases y tipos son identificados empezando por la letra “*T*”.
- Cada botón tiene su respectivo procedimiento de respuesta al clic, en tiempo de ejecución: *Button1Click* y *Button2Click*. Que son diferentes y no se confunden.
- Es el S.O. quien se encarga de llamar a la rutina correcta cada vez que, en tiempo de ejecución, se oprima un determinado botón. Es decir, el S.O. es quien genera los eventos y llama a las rutinas que vamos creando – de ahí que este tipo de programación sea conocida como **Programación Orientada a Eventos** (POE).

3. Modificando nombres y propiedades de componentes

3.1 – En el *Inspector de Objetos*, seleccione el *Button1* y después la pestaña *Propiedades*. En esta pestaña, se puede ver y modificar todas las propiedades (atributos) del objeto seleccionado actualmente. Estas propiedades son listadas en orden alfabético. La figura 18 muestra algunas propiedades del *Button1*.

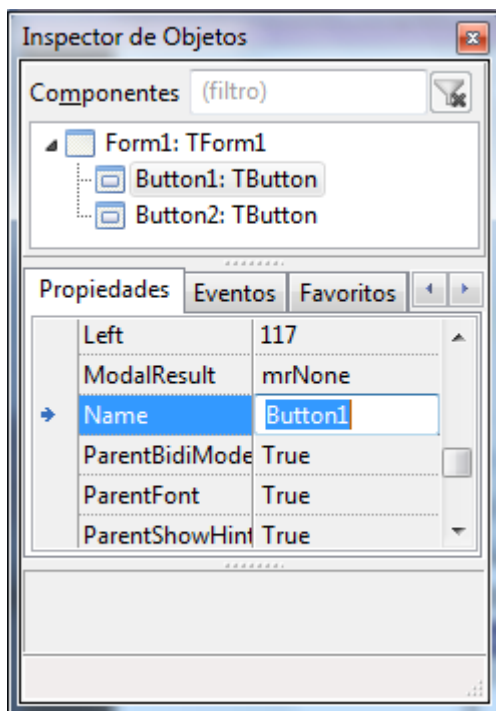


Figura 18: Inspector de Objetos mostrando propiedades de Button1.

3.2 – Seleccione ahora la pestaña Eventos. Esta pestaña muestra los eventos disponibles para los botones TButton, es decir, todos los sucesos relacionados con las instancias. El único campo con contenido es el Button1Click para el evento OnClick. Confirme lo mismo para los eventos del Button2 – el único evento listado es el Button2Click.

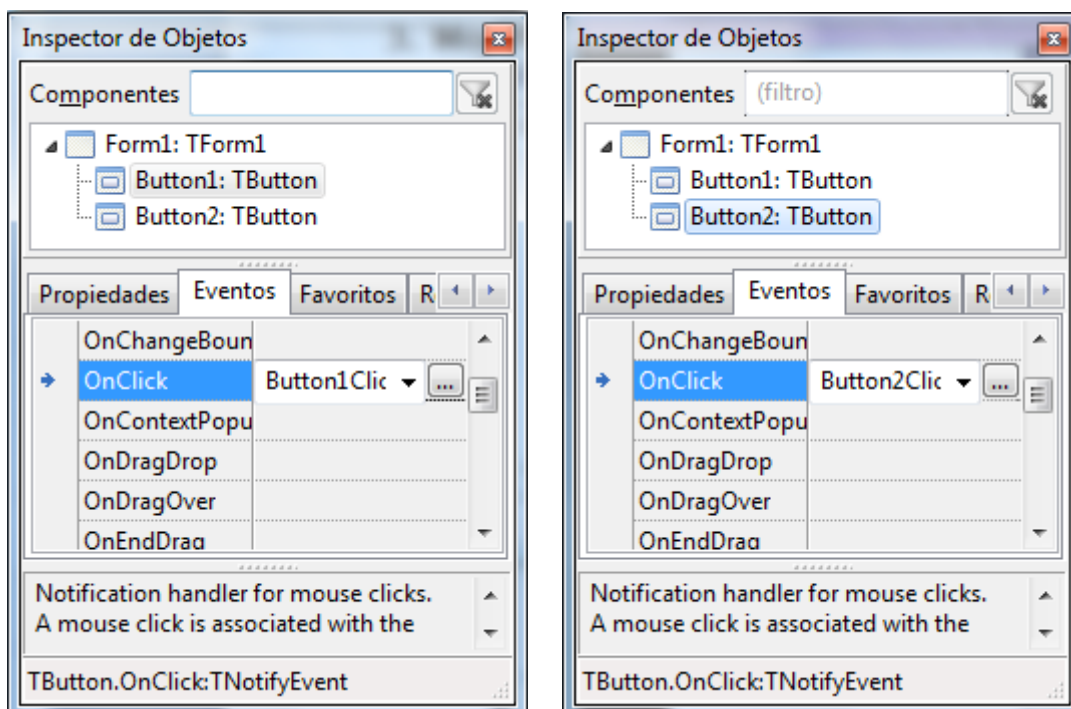


Figura 19: Comparación de eventos entre los botones.

Fíjese que, al seleccionar el evento OnClick de alguno de los botones, aparece el botón de edición [...]. Al clicar en dicho botón, seremos enviados a la ventana del *Editor de código fuente*, posicionados en el código respectivo, tal y cual sucedió cuando hicimos doble clic en el objeto.

3.3 – Seleccione ahora la pestaña Favoritos. Esta pestaña tiene el “resumen” de las propiedades y eventos más visitados y frecuentemente más utilizados.

3.4 – Los resúmenes de las propiedades de los botones son mostrados en la figura abajo.

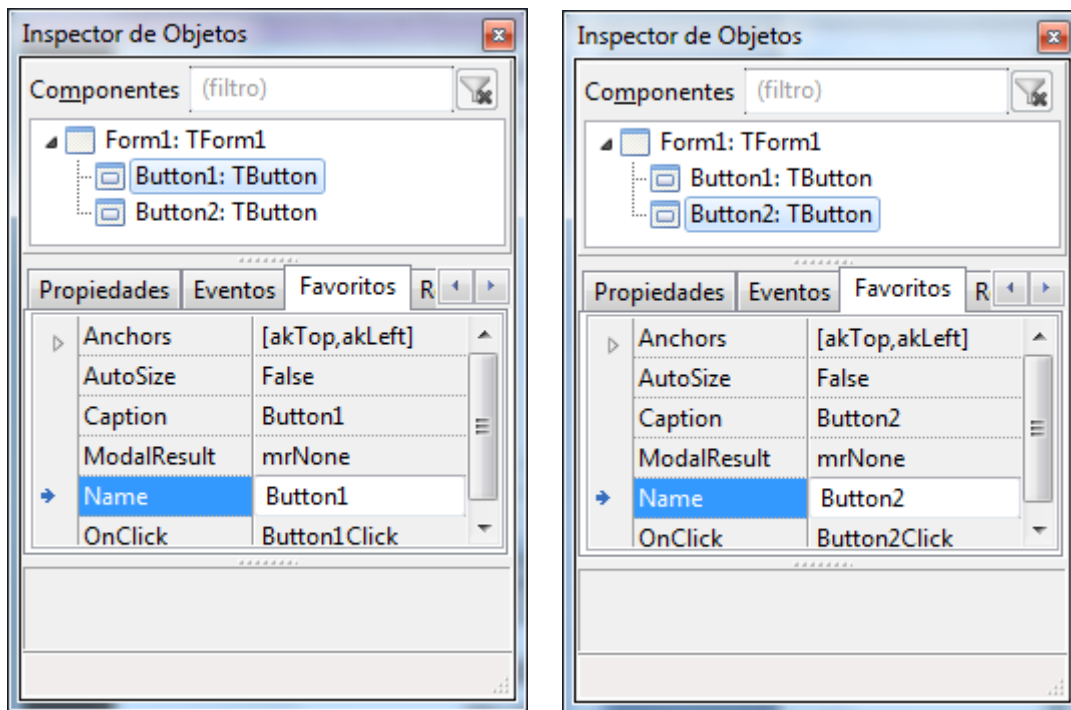


Figura 20: Pestaña Favoritos de ambos botones.

Cambia el nombre del Button1 para BCambiaCaption y del botón 2 para BShowBox. Cambia solamente la propiedad "Name" del Button1 y del Button2. Después de los cambios el Inspector de Objetos mostrará una apariencia semejante a la que vemos en la **Figura 21**.

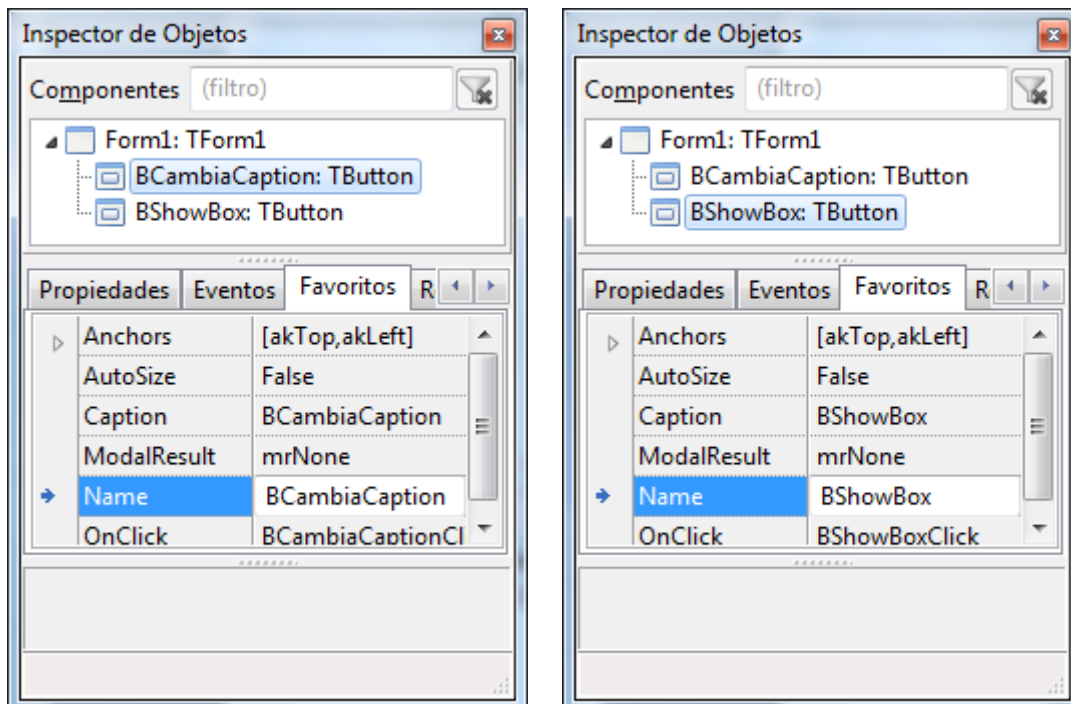


Figura 21: Propiedades y eventos, luego de cambiarse los nombres de los botones.

Note que la propiedad Caption y el evento OnClick fueron automáticamente renombrados. Además, el código respectivo también fue automáticamente modificado, como podemos ver en la **figura 22**.

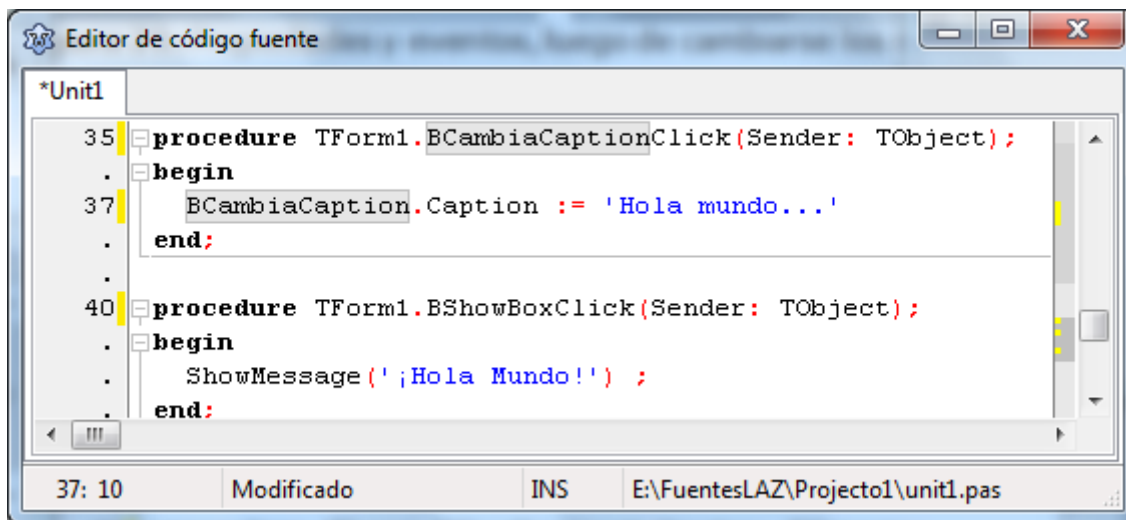


Figura 22: Código fuente modificado por la alteración del nombre de los objetos.

3.5 – Cambie ahora las propiedades Caption de ambos botones, tal y como se muestra en la **Figura 23**.

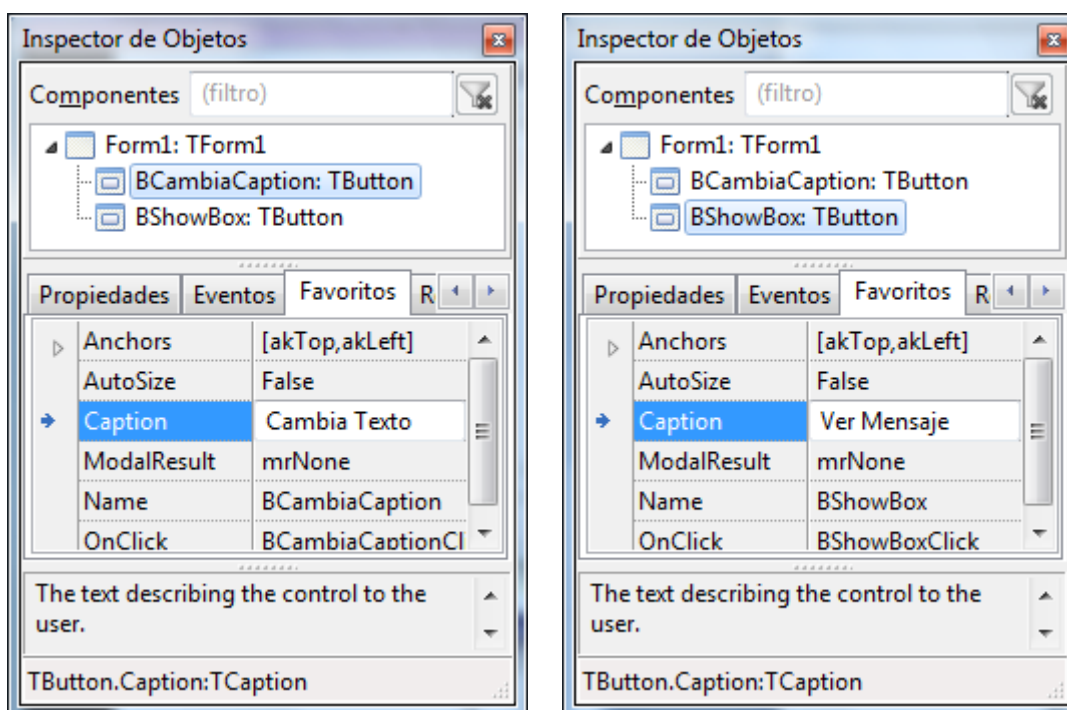


Figura 23: Cambiando los Captions de los botones.

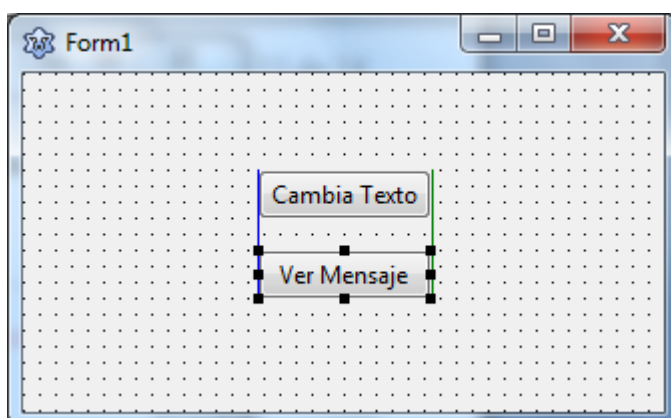


Figura 24: Los cambios a las propiedades reflejados en el formulario.

Observe que los cambios en las propiedades de los botones se vuelven efectivos inmediatamente y que el Formulario, en tiempo de diseño, es inmediatamente refrescado para reflejar los cambios ingresados. Tal y cual se muestra en la **Figura 24**.

3.6 – Seleccione el formulario en el Inspector de Objetos para ver sus propiedades.

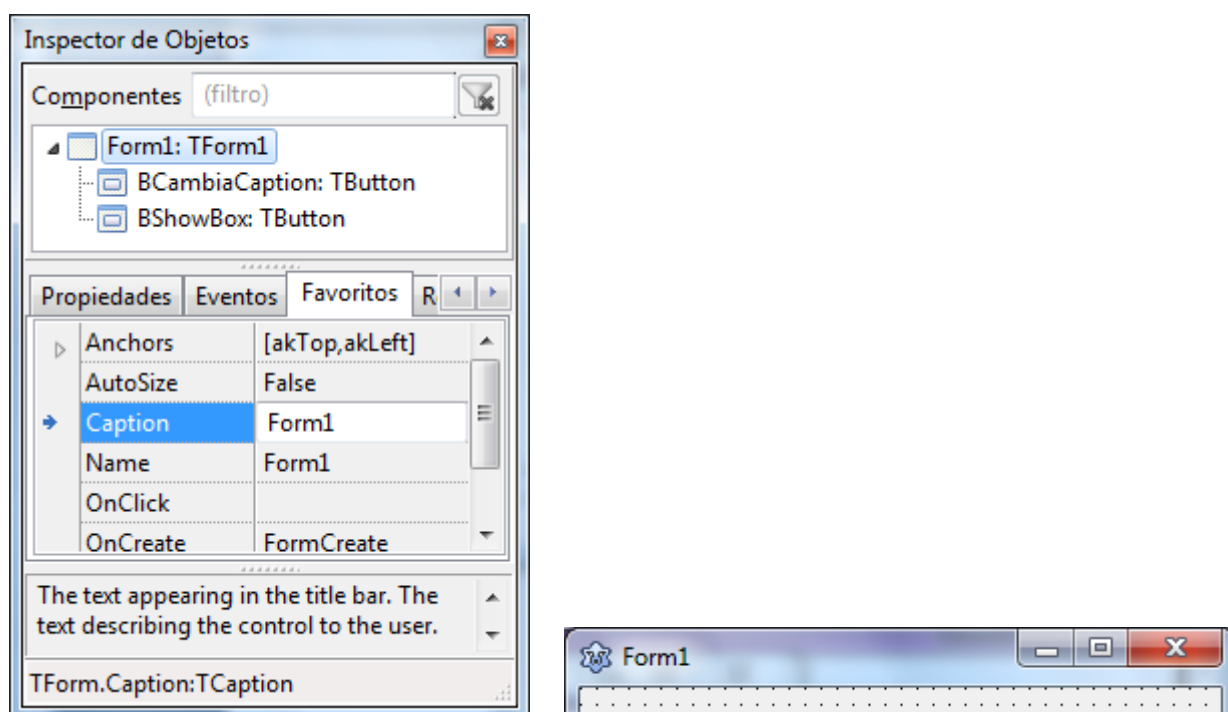


Figura 25: Propiedades iniciales del Formulario.

Modifique el texto asociado a la barra de título del formulario, es decir, su propiedad "Caption", para "Mi primer Proyecto". Observe que el Caption del formulario no tiene nada que ver con el Caption de cada uno de los botones. El *Inspector de Objetos* reflejará los cambios y el formulario pasará a mostrar el nuevo título.

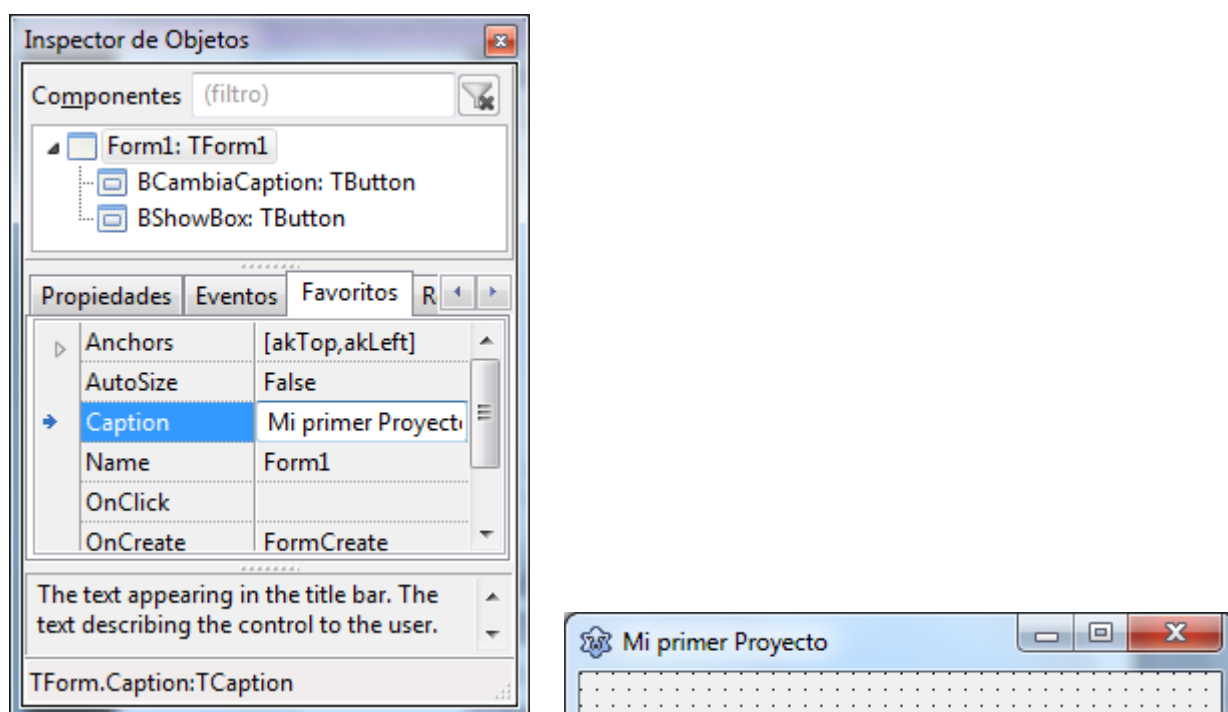


Figura 26: Caption del formulario debidamente modificado.

3.7 – Ejecute la nueva versión con [F9] para verificar las diferencias.

3.8 – Ideas a retener:

- Los nombres de los componentes, tales como los botones, deben ser descriptivos.
- Luego de cambiar los nombres de los componentes, el código es alterado automáticamente.
- Para una mejor localización de los controles dentro del formulario es recomendable armar una nomenclatura y utilizarla en todo el proyecto. Aquí, los nombres de todas las instancias de TButton (es decir, los botones) empezarán por la letra "B" – esto permitirá más adelante que todos los botones sean fácilmente localizables. Sin embargo, usted puede usar la nomenclatura que le resulte más cómoda.
La convención más utilizada es usar prefijos de tres letras minúsculas (normalmente sin vocales) representando el tipo de componente como: "btn" para botones, "edt" para cuadros de edición, "chk" para Check Boxes, etc. Estos prefijos son seguidos por un nombre descriptivo de la función del componente, e iniciará con una Mayúscula. Por ejemplo: btnSalir, edtNombre, etc.
Si la función del componente fuera un nombre compuesto, ese nombre debe ser escrito con los primeros nombres abreviados e iniciando en Mayúsculas. Por ejemplo: btnInfVentas (para botón Informe de Ventas) o btnInfVenProducto (para botón Informe de Ventas por Producto).
- Los cambios en las propiedades de los componentes visuales son visibles inmediatamente en tiempo de diseño – de ahí esta programación ser del tipo **Visual**.
- Tanto el formulario como cada uno de los botones tienen sus propiedades "Caption" – este concepto se llama **polimorfismo**; hay muchos componentes con propiedades iguales para funcionalidades similares – fíjese que Form.Caption y Button.Caption son conceptos similares para distintas clases.

4. Aplicación con Edit y Check Box

- 4.1 – Crea un nuevo proyecto de aplicación (Proyecto -> Nuevo Proyecto... -> Aplicación – para más detalles vea el ítem 1.2 y 1.3).
- 4.2 – Organiza las ventanas a su gusto (por ejemplo, tal como lo mostrado en la **Figura 3**).
- 4.3 – Elija en el menú Archivo -> Guardar Todo e guarde todos los archivos en una nueva carpeta diferente de la anterior. Recuerda no utilizar unidad de red para evitar lentitud en la generación del ejecutable. Utiliza los nombres "project2.lpi" y "myunit.pas" (utiliza minúsculas). De este modo, todos los archivos de este nuevo proyecto se quedan en la nueva carpeta. El Lazarus utiliza y crea varios archivos que son mantenidos sin intervención humana.
- 4.4 – Agrega un TEdit al formulario.

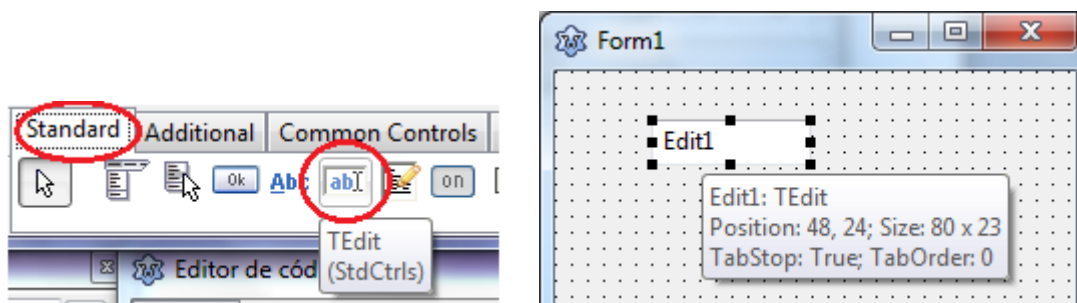


Figura 27: Agregando un TEdit al formulario

La clase TEdit provee la funcionalidad de un cuadro de edición de texto estándar del S.O. (texto genérico de una sola línea).

4.5 – Agrega un TCheckBox al formulario.

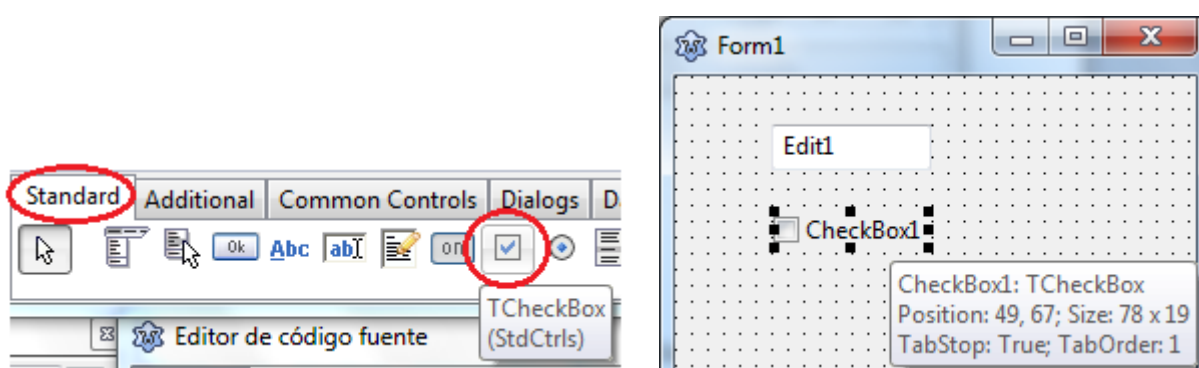


Figura 28: Agregando un TCheckBox al formulario.

La clase TCheckBox provee la funcionalidad de una casilla de verificación que alterna entre Verdadero/Falso y es una funcionalidad estándar de los S.O..

4.6 – Modifique las propiedades del CheckBox de la siguiente manera:

- **Caption:** "&Modificar" – el "&", antes de una letra del Caption, define una tecla de atajo que puede ser utilizada en tiempo de ejecución.
- **Checked:** True – un doble clic en el *False* cambia para *True* e viceversa.
- **Name:** CBPermiteCambiar. Fíjese que aquí, para los CheckBox, usamos como convención anteponer las letras CB al nombre del objeto.

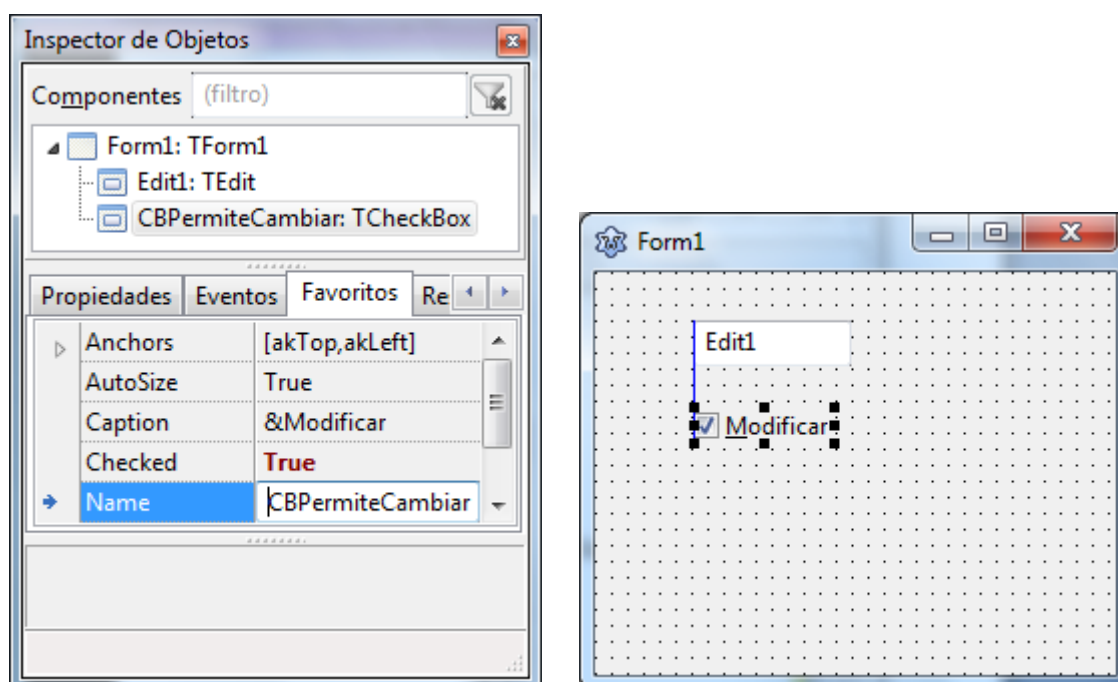


Figura 29: CheckBox configurado según lo indicado.

Observe que el Check Box es un control estándar del S.O. pero el formato exacto de la "verificación" o de la "cruz" que aparece para indicar que esta opción está marcada depende de la configuración del S.O..

4.7 – Modifica las propiedades del Edit1 tal y como se indica a seguir:

- **Name:** EditMiTexto
- **Text:** Proyecto2

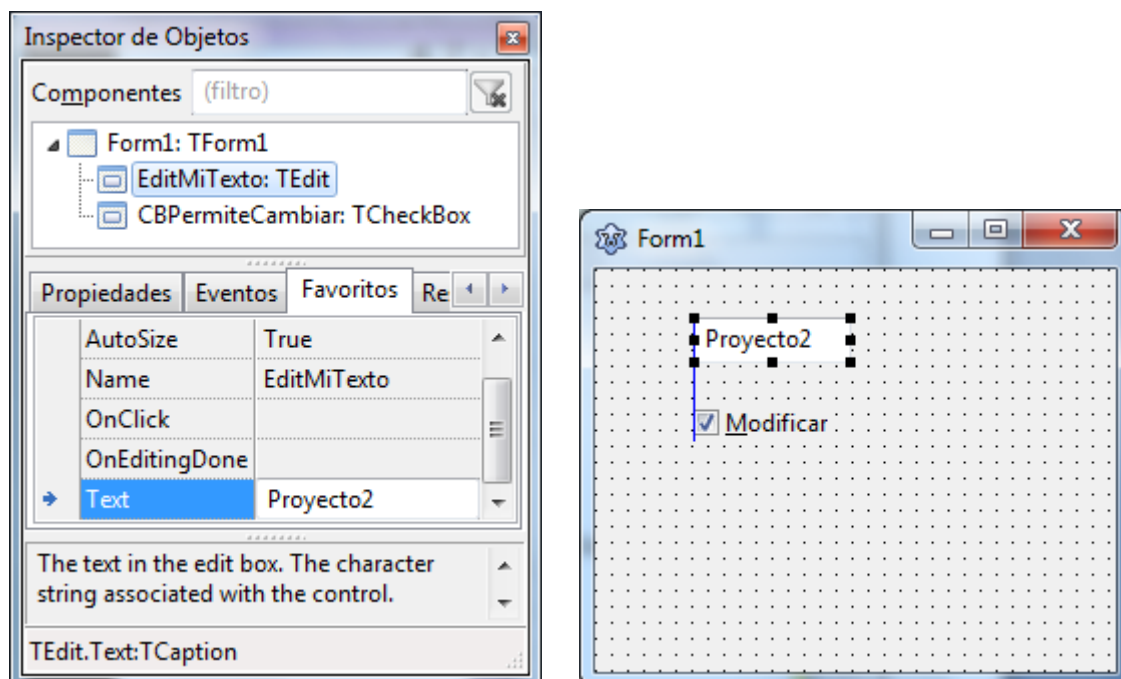


Figura 30: El TEdit configurado según lo especificado.

4.8 – Verifique en el listado de propiedades que existe una propiedad llamada `ReadOnly`, la cual puede presentar los valores verdadero y falso. Esta propiedad permite que en tiempo de ejecución sea posible (o no) alterar el texto de este componente (mantenga `ReadOnly` como `False`).

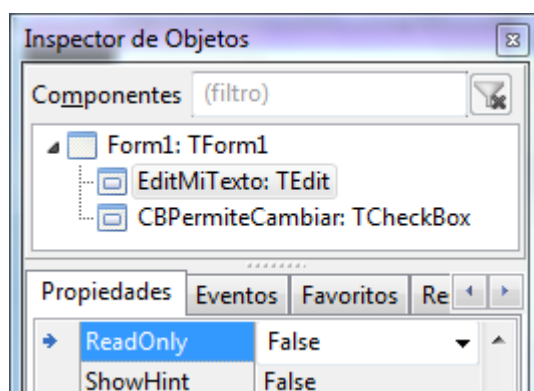


Figura 31: La propiedad `ReadOnly` del objeto `EditMiTexto`, en `False`.

4.9 - Modifique el Caption del formulario para que pase a decir "Proyecto 2". Algo semejante a lo que fue hecho en el ítem 3.6.

4.10 – Haga un doble clic sobre el `CheckBox` para crear y llamar el procedimiento `CBPermiteCambiarChange`. Este procedimiento será llamado siempre que se clicar sobre el `CheckBox`. Escriba "edit" y oprima `CTRL+SPACE` para activar la funcionalidad de "Completion", es decir, completar el código automáticamente. Esta opción abre una caja de selección de las posibilidades de escritura. Oprima `[Enter]`

para seleccionar la opción correcta (var EditMiTexto) e ver el texto EditMiTexto aparecer en el editor de código, isin tener que escribir más que las primeras letras!

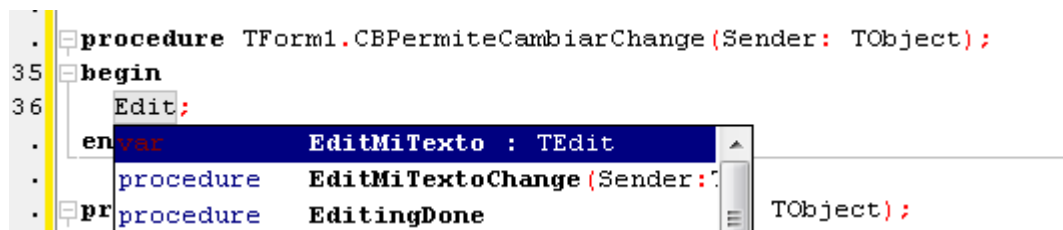


Figura 32: Cuadro de selección mostrando las opciones actuales para "edit...".

- 4.11 - Utiliza esta técnica para auxiliarle a escribir el texto indicado en la figura 33.

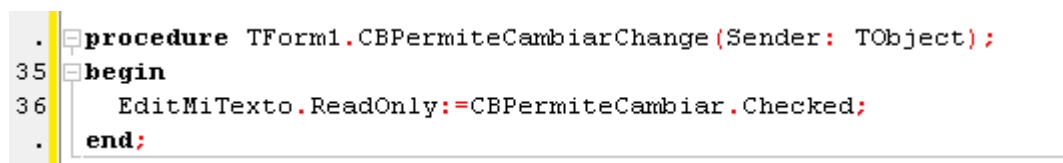




Figura 33: Código para el evento Change del CBPermiteCambiar.

- 4.12 - Guarde todos los cambios oprimiendo el botón  en la barra de herramientas principal y luego oprima el botón  para ejecutar la aplicación. Estos botones son lo mismo que elegir las opciones de menú Archivo -> Guardar todo y Ejecutar -> Ejecutar.
- 4.13 - Prueba la funcionalidad de la aplicación. ¿Hay algún problema con lo que se espera de su funcionamiento?
- 4.14 - Modifique el código del CBPermiteCambiarChange para lo indicado en la **figura 34**.

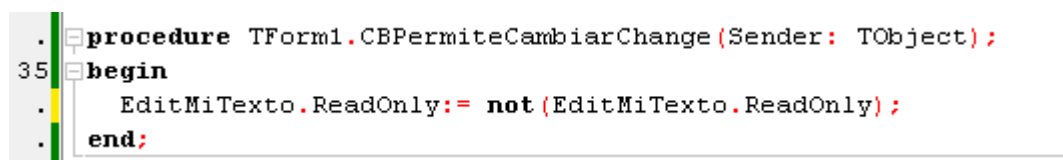


Figura 34: Nuevo código para el evento Change del CheckBox.

Fíjese que CBPermiteCambiar es el nombre del CheckBox y CBPermiteCambiarChange es el nombre del procedimiento que es llamado en el evento OnChange de este objeto.

- 4.15 - Graba y ejecuta la aplicación para observar que ahora sí la funcionalidad es la esperada. O sea, cuando la casilla de verificación está marcada, nos permite editar el texto y cuando no está marcada, el texto no puede ser alterado.
- 4.16 - Agrega un botón con el nombre de BActualizar e con Caption "&Actualizar".

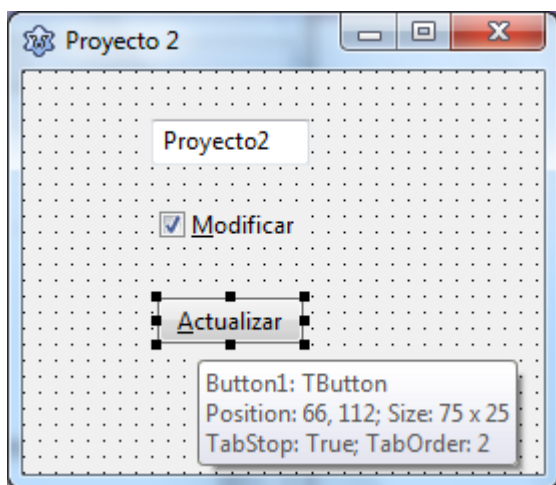


Figura 35: Botón Actualizar agregado al formulario.



- 4.17 – Haga un doble clic en el botón y escriba el código indicado en la Figura 36.

```

1 procedure TForm1.BActualizarClick(Sender: TObject);
42 begin
    Form1.Caption:=EditMiTexto.Text;
    end;

```

Figura 36: Código para el evento OnClick del botón BActualizar.

- 4.18 – Guarda todos los cambios oprimiendo el botón  y ejecuta la aplicación con el .
- 4.19 – Prueba acceder al los controles visuales oprimiendo las combinaciones de teclas Alt+M y Alt+A.

4.20 – Ideas a retener:

- Fue utilizado un EditBox llamado EditMiTexto. Es una buena práctica que todos los EditBoxes (cuadros de texto) tengan el prefijo "Edit".
- Fue utilizado un CheckBox llamado CBPermiteCambiar. Es una buena práctica que todos los CheckBoxes tengan el prefijo "CB".
- Cuando el programa arranca, es posible modificar el texto del componente EditMiTexto pues su propiedad "ReadOnly" fue definida en tiempo de diseño como False e por este motivo, al arrancar el programa dicha propiedad sigue como False.
- El componente CBPermiteCambiar fue definido en tiempo de diseño como estando seleccionado y por esto, en el arranque, esta casilla de verificación se presenta seleccionada: Checked como True.
- El texto asociado a la casilla de verificación indica "Modificar", por lo que será intuitivo que cuando esta casilla estuviera seleccionada, será posible modificar el contenido del cuadro de edición de texto. En términos de programación, solo tenemos acceso a las propiedades "**EditMiTexto.ReadOnly**" e "**CBPermiteCambiar.Checked**", lo que obliga a la utilización de la negativa booleana: `EditMiTexto.ReadOnly:= not(EditMiTexto.ReadOnly);`

6. Utilización de procedimientos y escrita rápida (Code Templates, Completion y Parameter CheckOut)

Aquí aprenderemos a crear procedimientos definidos por el usuario utilizando para ello las funcionalidades de escrita rápida disponibles en Lazarus.

- 6.1 – Busca en el código de la unidad *myunit.pas* el procedimiento *CBPermiteCambiarChange*. Más arriba de ese código y fuera de cualquier procedimiento (que empiezan por *begin* y terminan con *end;*), escriba "proc".

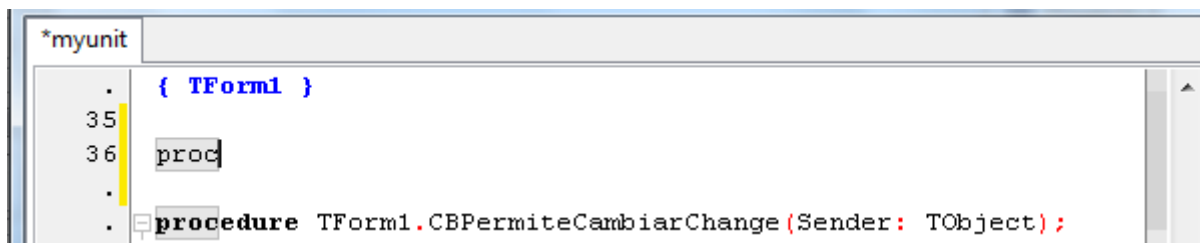


Figura 38: Creando un nuevo procedimiento.

Oprima **CTRL+J** para activar la funcionalidad "Code Templates" y obtendrá la estructura mostrada en la figura 39.

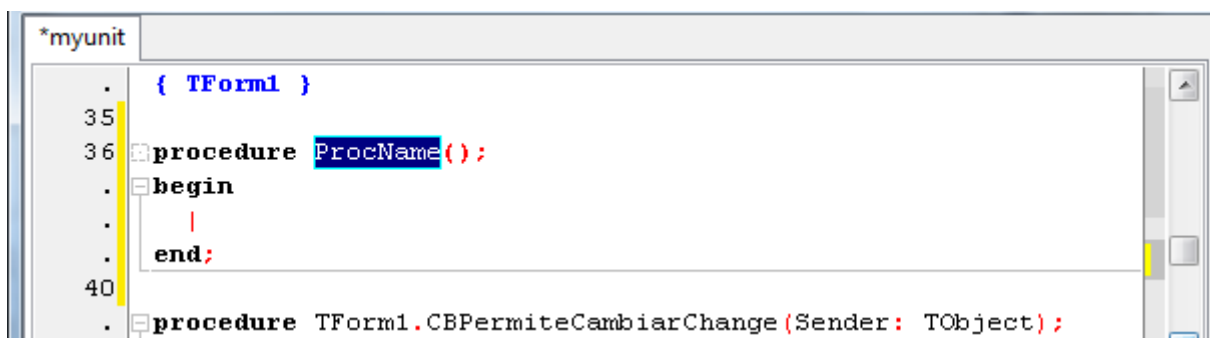


Figura 39: Estructura creada por la plantilla de código "procedure".

Nota: si en lugar de la estructura mostrada en la figura 39, aparece algo como: `procedure $Param(ProcName) | ($Param())`. Oprima **Ctrl+Z** para deshacer la última acción, vaya a **Herramientas -> Plantillas de código...**, seleccione la plantilla **Procedure** en el cuadro de lista correspondiente y luego marque la casilla ☒ **Activar Macros**. Oprima **Aceptar** y vuelva a intentar la combinación de teclas **Ctrl+J**. Ahora ya debería funcionar como esperado.

Escriba "Actualiza" y oprima **TAB** para que el cursor se posicione dentro del paréntesis. Complete el resto del código del procedimiento como lo mostrado en la figura abajo.

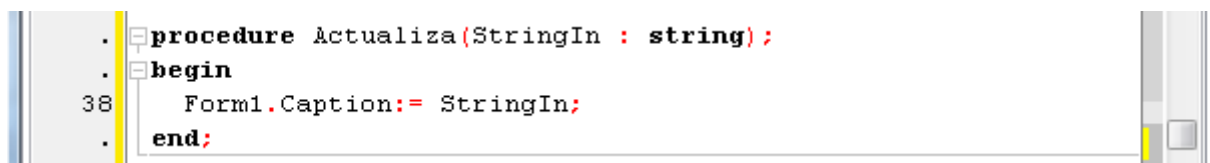
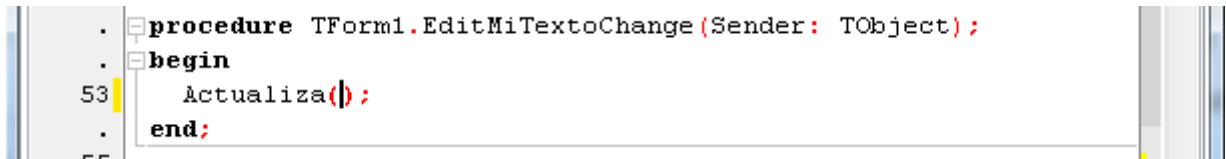


Figura 40: Código para el procedimiento *Actualiza*.

Este código define un **procedimiento** (una función que no devuelve nada) el cual recibe como parámetro de entrada una variable con el nombre "StringIn" del tipo **string** (cadena de caracteres).

- 6.2 – Haga un doble clic sobre el Edit y será llevado al código correspondiente al evento EditMiTextoChange. Elimina el código que habíamos puesto allí y luego escriba "actu" y oprima CTRL+SPACE(barra espaciadora).



```
procedure TForm1.EditMiTextoChange(Sender: TObject);
begin
53   Actualiza();
end;
```

Figura 41: El código autocompletado con el nuevo procedimiento creado.

Luego que aparezca la referencia al nuevo procedimiento creado, con el cursor dentro del paréntesis para que se ingrese el parámetro respectivo, oprima las teclas Ctrl+Shift+Space para ver los argumentos que la función o procedimiento esperan (vea el texto flotante que aparece luego arriba del cursor).

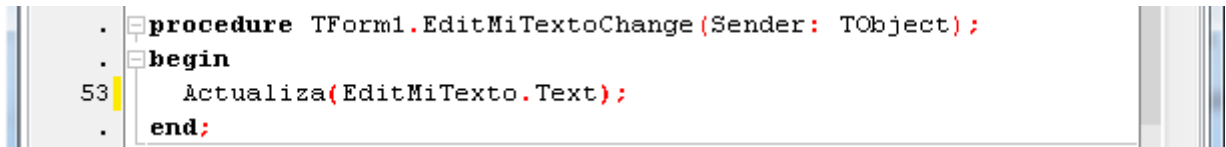


```
procedure TForm1.EditMiTextoChange(Sender: TObject);
begin
53   Actualiza();
end;
```

Figura 42: Texto flotante mostrando el formato definido para el procedimiento Actualiza.

En este caso, el procedimiento espera un parámetro con el nombre "StringIn" del tipo string (cadena de caracteres).

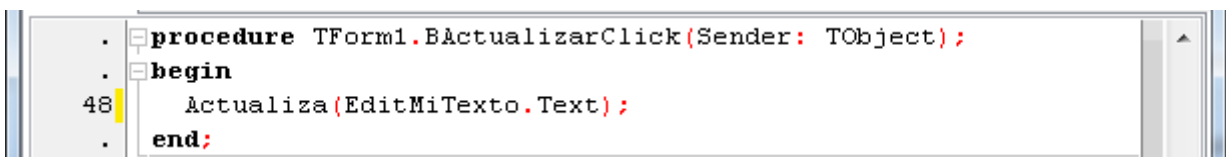
- 6.3 – Complete el código tal como lo mostrado en la figura abajo. Para ello debería seguir utilizando la utilidad de complementación automática de código (CTRL+SPACE).



```
procedure TForm1.EditMiTextoChange(Sender: TObject);
begin
53   Actualiza(EditMiTexto.Text);
end;
```

Figura 43: El nuevo código para el procedimiento EditMiTextoChange.

- 6.4 – Navega por el código hasta encontrar el procedimiento BActualizarClick y modifícalo de acuerdo con lo mostrado en la figura abajo.



```
procedure TForm1.BActualizarClick(Sender: TObject);
begin
48   Actualiza(EditMiTexto.Text);
end;
```

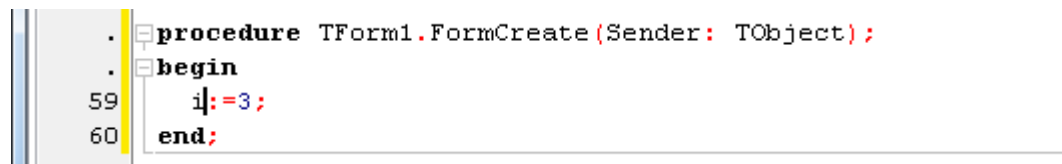
Figura 44: El nuevo código para el procedimiento BActualizarClick.

- 6.5 – En este momento los procedimientos EditMiTextoChange y BActualizarClick llaman a otro procedimiento llamado "Actualiza" que centraliza el proceso de actualización.

- 6.6 – Graba y ejecuta el proyecto para verificar que el conjunto funciona.

- 6.7 – Como sabemos, en FPC siempre debemos declarar las variables antes de utilizarlas. Así que otra forma muy útil de complementación de código es el Complementar Declaración de Variable (Variable Declaration Completion). Esta herramienta agrega la definición de una variable local para una instrucción tipo: Identificador:=Valor. Es invocada cuando el cursor está en el identificador de una asignación o parámetro. Veamos un ejemplo de cómo usarla:

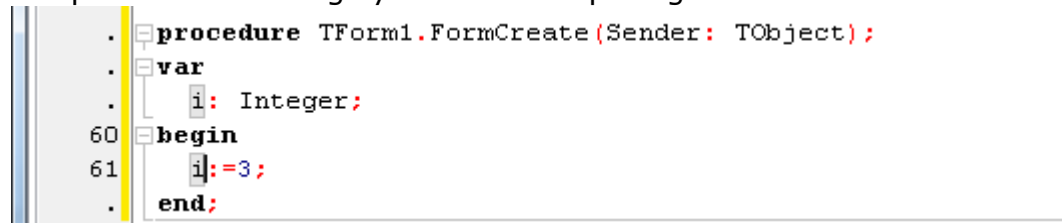
6.7.1 – Vaya al formulario y haga un doble clic en cualquier parte. Ingresa el código mostrado en la figura abajo.



```
procedure TForm1.FormCreate(Sender: TObject);
begin
59   i:=3;
60 end;
```

Figura 45: Código de una asignación simple a una variable local.

6.7.2 – Posiciona el cursor en el identificador "i" o justo después del mismo (como en la figura). Oprima la combinación de teclas CTRL+SHIFT+C para complementar el código y obtendrá lo que sigue:



```
procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
60   i:=3;
61 end;
```

Figura 46: Acción del complemento de código para definir una variable local.

La herramienta primero verifica si el identificador "i" ya fue definido y, si no, agregará la declaración "var i : integer;". El tipo del identificador es presumido a partir del valor a la derecha del operador de asignación ":= ". Números como el 3 predeterminan Integer (entero).

6.8 – **Ejercicio libre:** Agregue otra CheckBox al formulario con el nombre y Caption adecuados. Modifique la aplicación para implementar la funcionalidad de que la actualización solo debe ser automática cuando la casilla de verificación esté marcada. Cuando esté desmarcada, será necesario oprimir el botón para actualizar el formulario.

6.9 – **Ejercicio libre:** Modifique la aplicación de tal forma a que el Caption del formulario sea el texto del Edit repetido con un espacio intermedio. Para tener mínimo esfuerzo, ¿dónde deberá insertar las modificaciones????

6.10 – Ideas a retener:

- El evento *OnChange* del componente *TEdit* llama al procedimiento llamado **Actualiza** que cambia el **Form.Caption**
- Fue creado un procedimiento para evitar la repetición de código y el ejercicio 6.8 se beneficia de este hecho.
- La escritura de código FPC/Lazarus se ve facilitada por medio de diversas técnicas, entre ellas:
 - *Code Templates (Plantillas de Código)* – atajo CTRL+J
 - *Code Completion (Autocompletado de Código)* – atajo CTRL+SPACE
 - *Parameter CheckOut (Parámetros pedidos)* – atajo CTRL+SHIFT+SPACE
 - *Variable Declaration Completion (Auto creación de variables)* – atajo CTRL+SHIFT+C

7. Navegación por el código y formulario/código

En este tópico vamos a ver más algunas de las combinaciones de teclas que nos pueden ayudar a navegar más fácilmente por el código.

- 7.1 – Oprima la combinación de teclas Ctrl+Inicio para ir al inicio del código de la unidad actual. Por ejemplo, para recordar que el nombre de la unidad es **myunit**.



Figura 47: Cursor posicionado en el punto inicial del código de la unidad.

- 7.2 – Oprima CTRL+H para volver al punto de edición anterior.

- 7.3 – Mantenga oprimida la tecla CTRL y haga clic con el ratón sobre el texto "TForm1", que aparece subrayado.

```
procedure TForm1.EditMiTextoChange(Sender: TObject);
```

- 7.4 – Con este clic, el cursor será llevado a la parte superior de la unidad myunit, donde se define el tipo del Form1 llamado TForm1. Es aquí donde se lista los componentes que están dentro del formulario.

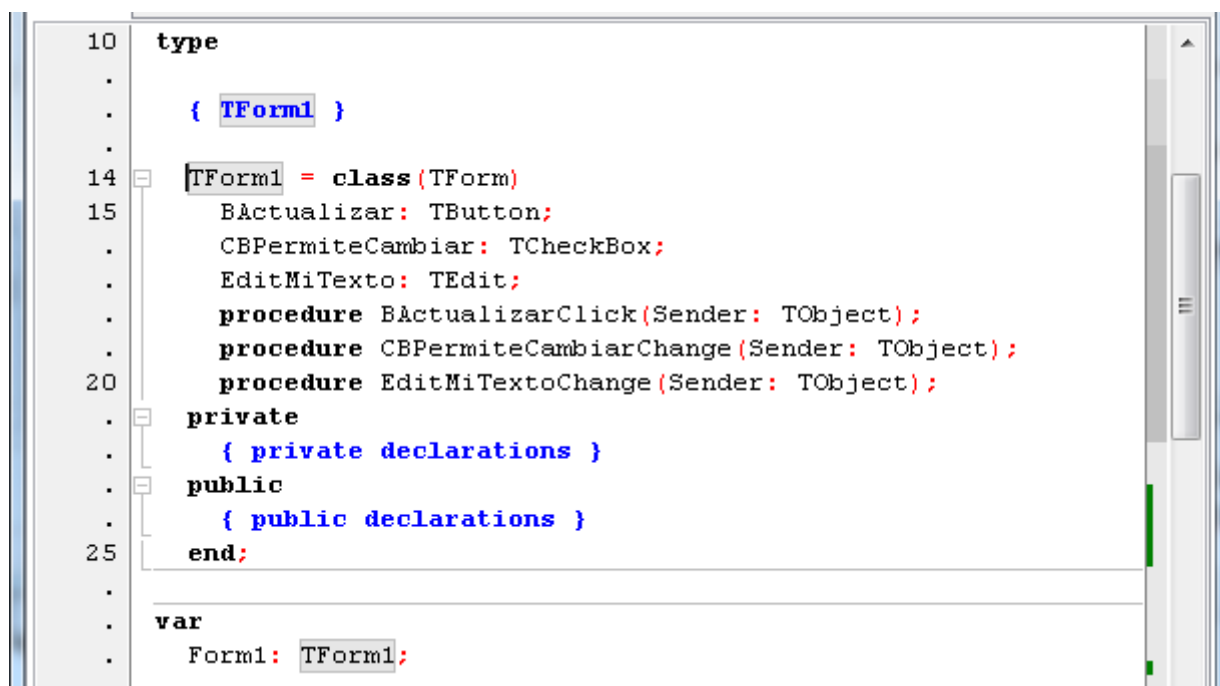
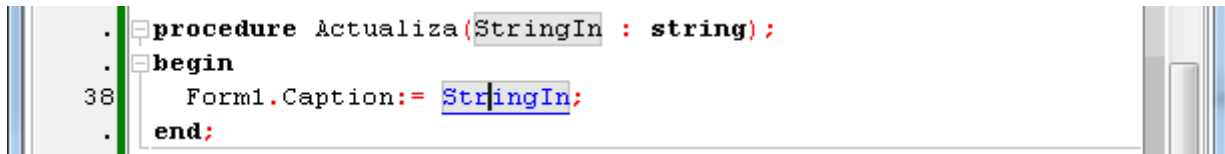


Figura 48: Código de definición de la clase TForm1 e instanciación del Form1.

Este código indica que el Form1 incluye un TButton llamado BActualizar, un TCheckBox llamado CBPermiteCambiar y un TEdit llamado EditMiTexto.

- 7.5 – Después del "end" que indica el final de la declaración del tipo TForm, aparece la instanciación del formulario propiamente dicho, con "**Var** Form1: TForm1;" indicando que el Form1 es de la clase TForm1.
- 7.6 – Navegue con el cursor hasta la línea donde se declara cualquiera de los procedimientos **del formulario** y oprima CTRL+Shift+flecha_abajo para ser llevado hasta la implementación de este procedimiento o función. Oprima CTRL + Shift + flecha_arriba o CTRL+H para volver a la declaración. Ejercite para percibir.

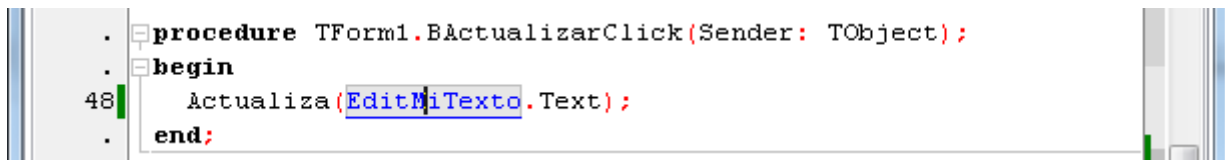
- 7.7 – Encuentre el código abajo, mantenga presionada la tecla CTRL y haga clic en StringIn para ser conducido a la declaración respectiva.



```
. procedure Actualiza(StringIn : string);  
. begin  
38   Form1.Caption:= StringIn;  
. end;
```

Figura 49: Utilizando la tecla CTRL para acceder a la declaración de StringIn.

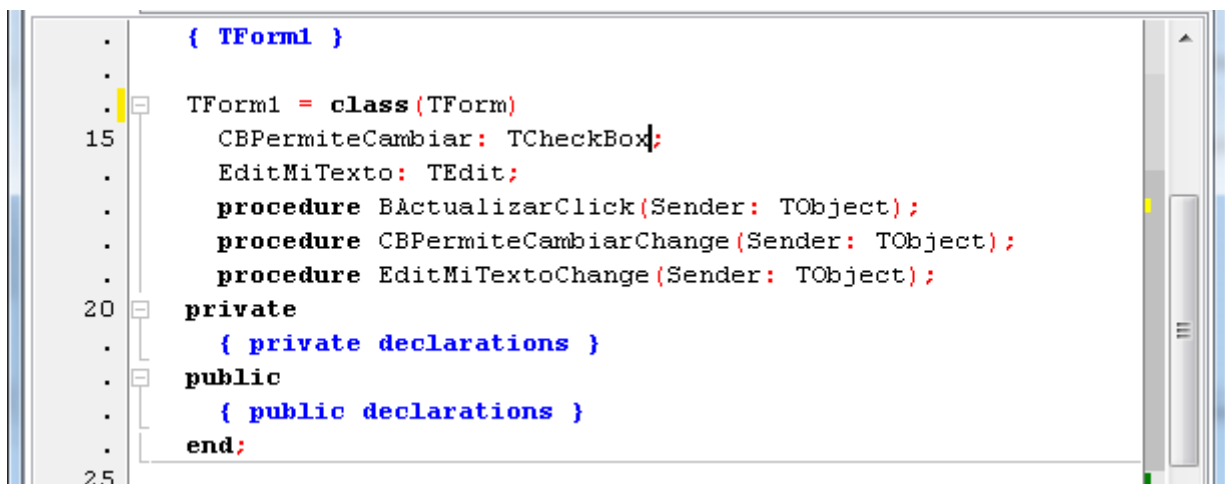
- 7.8 – Pruebe lo mismo con el EditMiTexto que está en el código para el evento Click del botón, para ser llevado hasta la definición del componente.



```
. procedure TForm1.BActualizarClick(Sender: TObject);  
. begin  
48   Actualiza(EditMiTexto.Text);  
. end;
```

Figura 50: Utilizando la tecla CTRL para acceder a la declaración del EditMiTexto.

- 7.9 – Oprima F12 para ser llevado hasta el formulario.
- 7.10 – Borre el botón Actualizar.
- 7.11 – Oprima de vuelta el F12 para ver que en la declaración del formulario dejó de constar el botón eliminado.



```
. { TForm1 }  
.   
. TForm1 = class(TForm)  
15   CBPermiteCambiar: TCheckBox;  
.   EditMiTexto: TEdit;  
.   procedure BActualizarClick(Sender: TObject);  
.   procedure CBPermiteCambiarChange(Sender: TObject);  
.   procedure EditMiTextoChange(Sender: TObject);  
20 private  
.   { private declarations }  
. public  
.   { public declarations }  
. end;  
25
```

Figura 51: Declaración de TForm1 luego de eliminar el botón.

El código del botón BActualizarClick se mantiene pero nunca será llamado.

7.12 – Ideas a retener:

- Las modificaciones hechas en el Form cambian el código de la respectiva Unit.
- Existe la posibilidad de que se tenga código que nunca será llamado.
- Quien decide qué código es ejecutado y por orden de quién es el S.O. que llama a la rutina dentro de la aplicación según los eventos generados, por medio de la inclusión de las órdenes del usuario.
- A un Form corresponde un tipo de formulario (TForm).
- Un Form está en una unidad de código ("unit"). La unidad y el Form pueden tener cualquier nombre, iidesde que sean diferentes!!!

- Normalmente, una Unit solo contiene un Form.
- Una Unit puede contener un formulario o no (la Unit puede ser visual o no).
- Obs.: No todas las informaciones del Form son mantenidas dentro de la unidad – hay otras informaciones, guardadas en otros archivos.
- Por seguridad y facilidad de utilización, se debe crear una nueva carpeta para cada proyecto Lazarus y guardar todos los archivos de dicho proyecto en la misma carpeta.
- ¡Nunca alterar el código que no fue hecho por nosotros!
- Navegación:
 - F12 alterna entre el formulario y el Editor de Código fuente.
 - F11 activa el Inspector de Objetos.
 - CTRL + Clic lleva a la declaración de algo.
 - CTRL + SHIFT + Arriba y CTRL + SHIFT + Abajo alternan entre la declaración y la implementación de procedimientos y funciones.

8. Aplicación con RadioGroup y Memo

- 8.1 – Crea un nuevo proyecto de aplicación: Proyecto -> Nuevo Proyecto... -> Aplicación -> Aceptar
- 8.2 – Diseñe el siguiente formulario que contiene un TRadioGroup y un TMemor, ambos situados en la pestaña Standard de la Barra de Componentes.

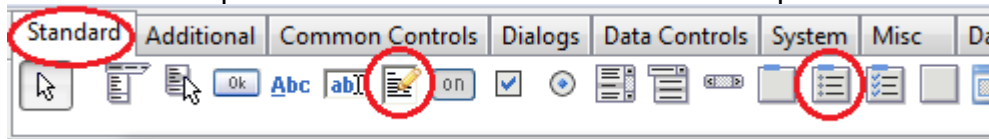


Figura 52: Posición de los botones para instanciar TRadioGroup y TMemor

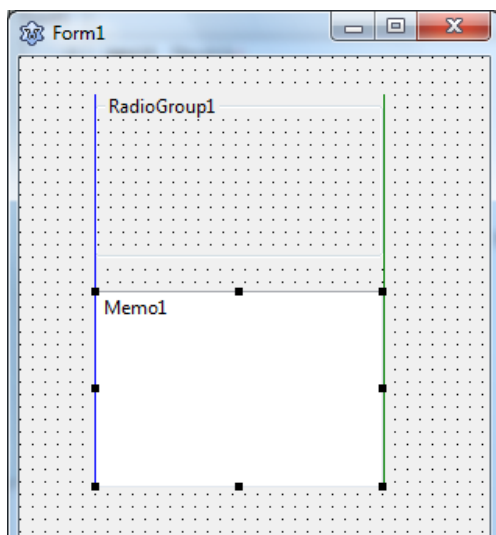


Figura 53: TRadioGroup y TMemor instanciados en el formulario Form1.

- 8.3 – Encuentra la propiedad Lines del componente Memo1, haga clic en la propiedad y aparecerá un botón de acceso [...]. Oprima dicho botón en el *Inspector de Objetos* y, en el *Diálogo Editor de Strings*, ingrese tres líneas para el Memo: Línea uno, Línea dos y Línea tres.

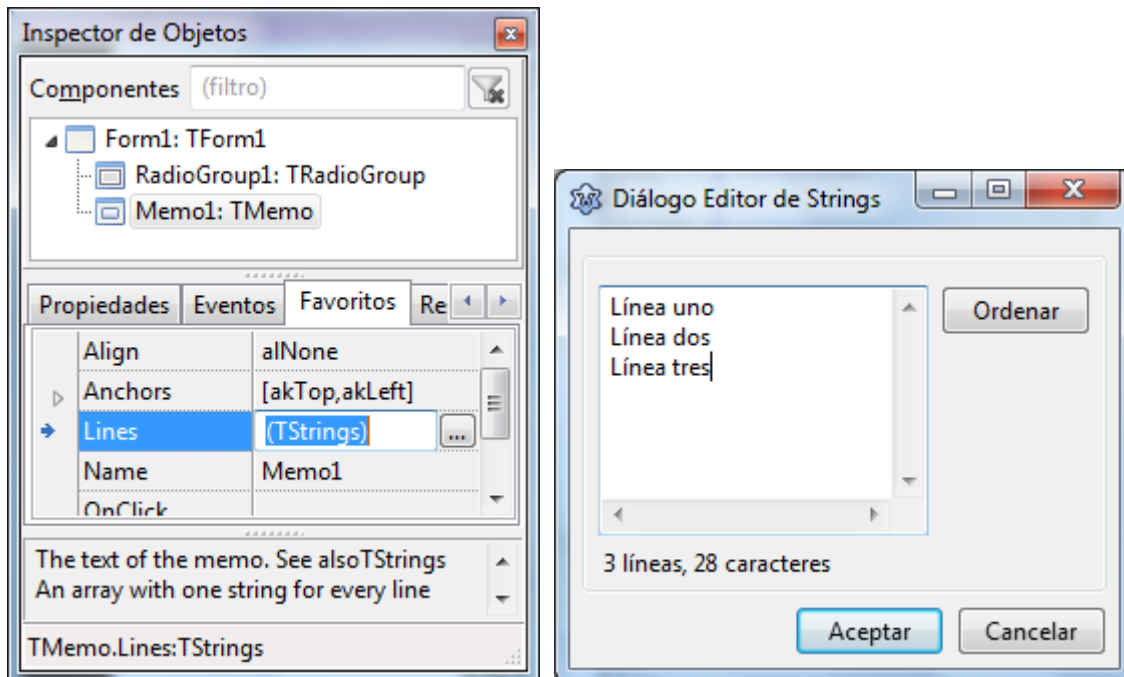


Figura 54: Agregando líneas al componente Memo1.

Así como puede ver, un Memo Box es un cuadro de texto que permite múltiples líneas. La propiedad "Lines" es del tipo TStrings el cual permite contener un texto con varias líneas.

- 8.4 – Encuentra la propiedad Items del RadioGroup1, oprima [...] e ingrese "Primero" y "Segundo". Después de eso modifica Item Index para 0 – note que el ítem "Primero" pasó a estar seleccionado. Debe haber notado que este es más un componente estándar del S.O..

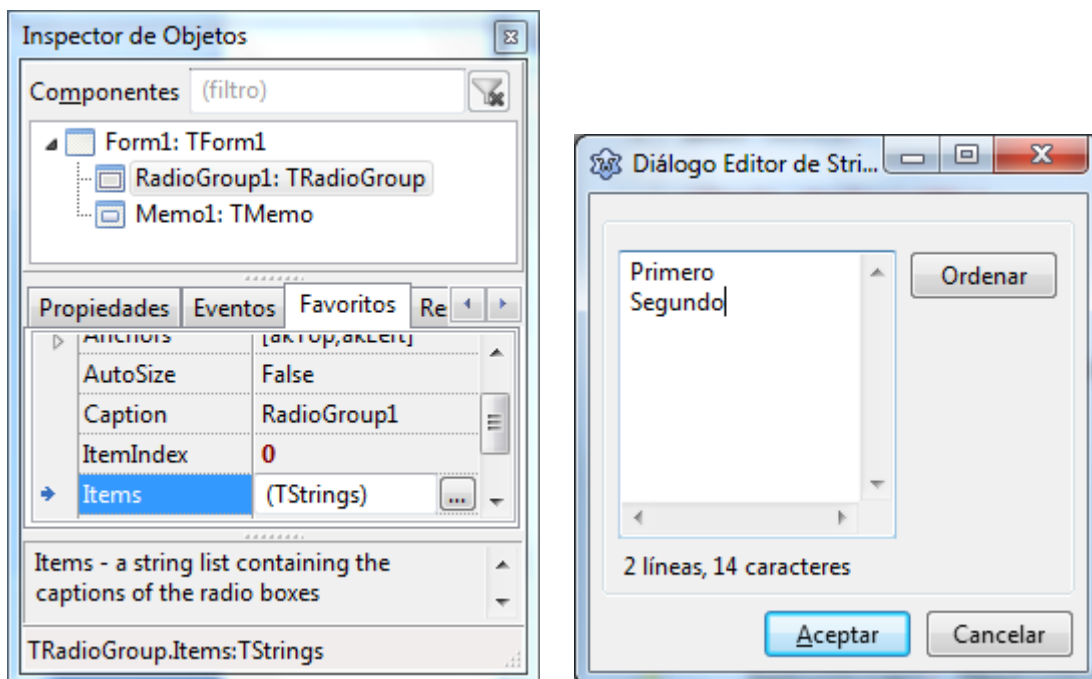


Figura 55: Agregando opciones al Radio Group.

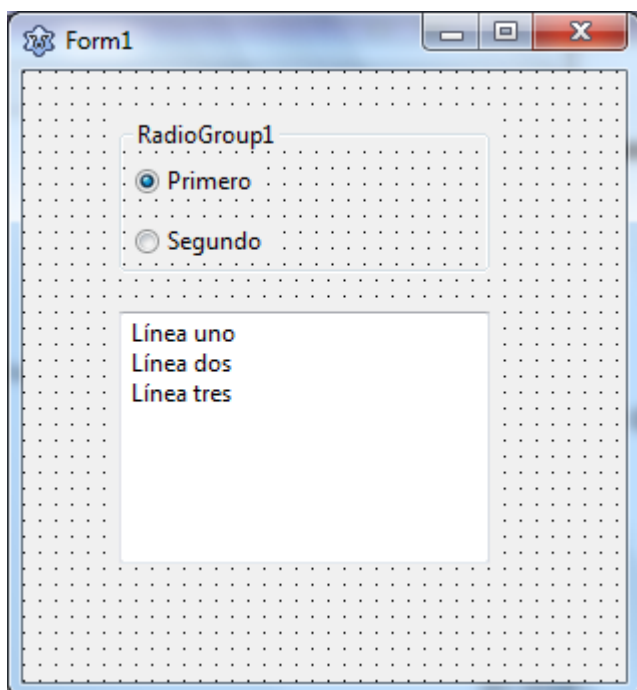


Figura 56: El Memo1 y el RadioGroup1 debidamente configurados.

Nota: el espaciado entre las opciones de un RadioGroup es automático, equidistante y depende del tamaño con que se defina el componente como un todo.

8.5 – Haga un doble clic sobre el *Radio Group*, agrega el código tal como se muestra en la siguiente figura y luego oprima CTRL+SHIFT+Space para ver las listas de parámetros que esa función acepta.

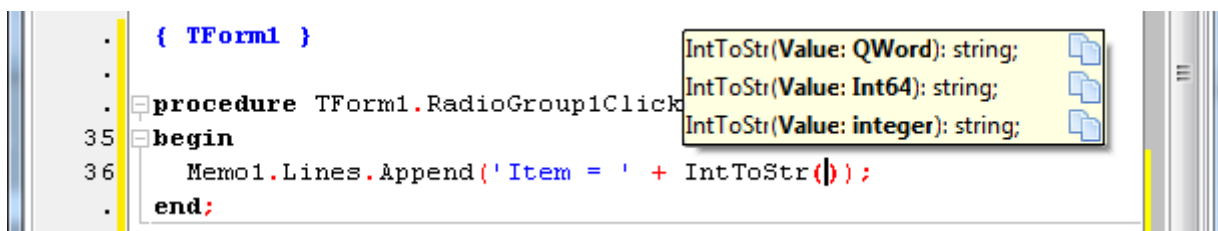


Figura 57: Argumentos permitidos para la función IntToStr.

Para la función IntToStr(), los argumentos pueden ser de varios tipos (pero siempre enteros).

8.6 – Complete el código tal como se muestra en la figura abajo.

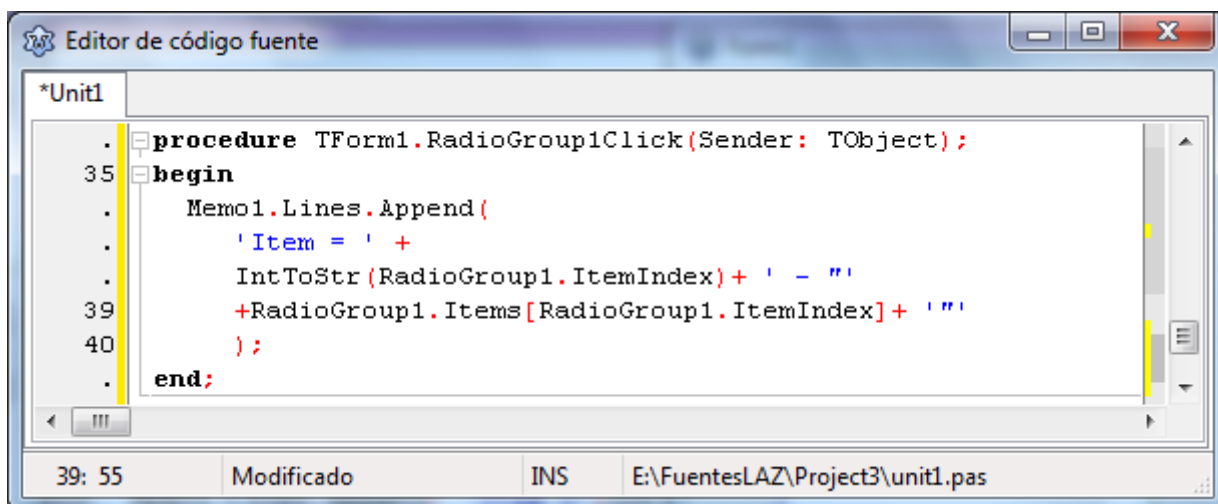


Figura 58: Código completo para el evento Click del Radio Group.

8.7 – Graba y ejecuta el proyecto para probar las funcionalidades: cuando se cambia la selección en el radio group, es agregada una línea de texto que describe lo que se está seleccionando.

8.8 – Ahora agrega un botón con el Caption “Describe” cuyo nombre sea “BDescribe”.

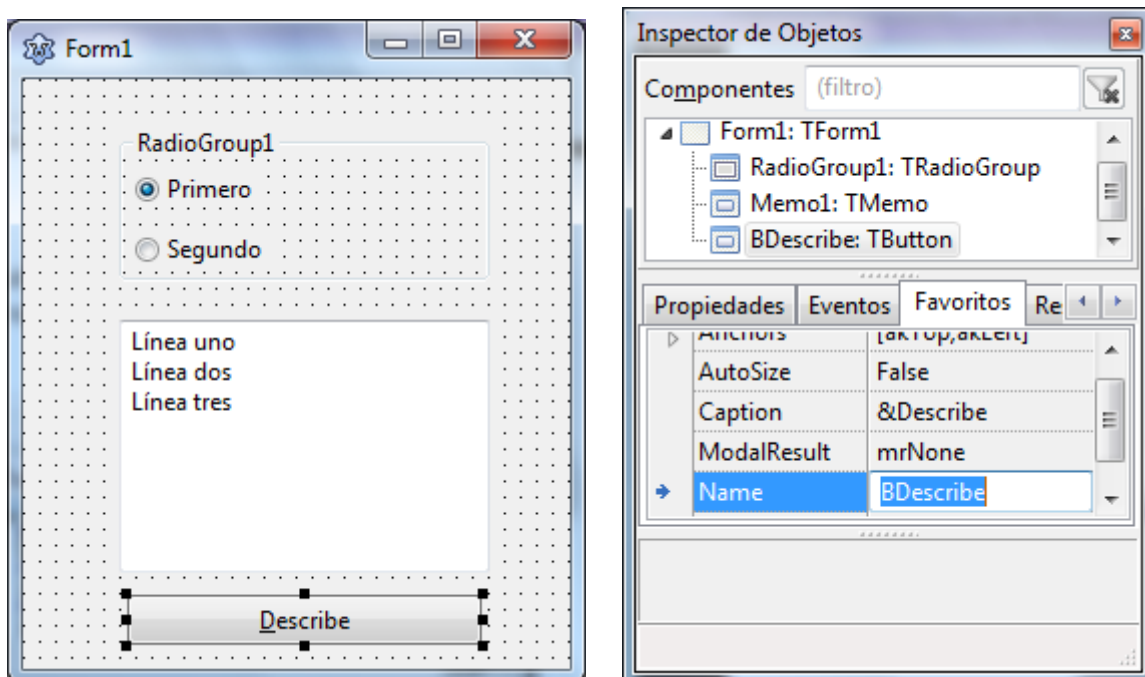


Figura 59: Agregando el botón BDescribe al formulario.

8.9 – Haga con que el evento OnClick del Radio Group y el OnClick del botón llamen a la función DescribeRadioGroup que se muestra en la figura abajo. Note que como la función es externa al Form1, es necesario resaltar la noción de que todos los controles visuales que hemos estado manipulando están dentro del Form1 y por esto es necesario indicar Form1.Memo1.Lines.Append, etc. A esto se le dice indicar la **ruta jerárquica** del objeto, propiedad o evento al que nos referimos.

```
35 procedure DescribeRadioGroup();  
begin  
    Form1.Memo1.Lines.Append( 'Item = ' +  
        IntToStr (Form1.RadioGroup1.ItemIndex) + ' - "'  
        +Form1.RadioGroup1.Items[Form1.RadioGroup1.ItemIndex] + '"'  
40 );  
end;
```

Figura 60: El código para la función DescribeRadioGroup.

8.10 – Graba y ejecuta para probar todas las funcionalidades.

8.11 – **Ejercicio libre:** Modifica el formulario para que pase a tener la configuración mostrada en la figura abajo. Utiliza el nuevo botón para agregar un nuevo ítem, en tiempo de ejecución, al *Radio Group* (utiliza la función RadioGroup.Items.Append para agregar ítems en tiempo de ejecución).

Nota: para agregar la barra de desplazamiento vertical que aparece en el componente Memo, hay que cambiar la propiedad ScrollBars de dicho componente a "ssVertica"l (aparece siempre) o a "ssAutoVertical" (solo aparece cuando es necesario).

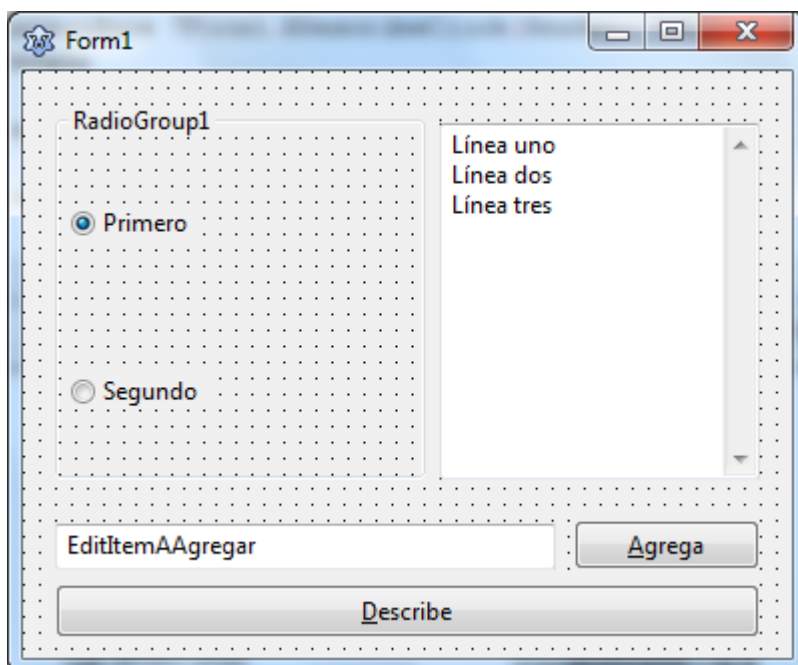


Figura 61: Formulario de ejemplo para el ejercicio libre.

8.12 – Ideas a retener:

- En el control visual *RadioGroup* se elige al máximo una de varias opciones.
- Cada opción del *RadioGroup* pertenece a un *TStrings* llamado *Items*.
- El ítem seleccionado actualmente es indicado por la propiedad *ItemIndex* (-1 significa que nada está seleccionado, 0 es el primer ítem, 1 el segundo y así sucesivamente).
- Es posible tener ítems iguales, ya que dichos ítems son numerados.
- Los ítems pueden ser indicados en tiempo de diseño, en el Inspector de Objetos, pero también pueden ser agregados en tiempo de ejecución.
- Para agregar ítems al RG, se utiliza `RadioGroupName.Items.Append()`
- Los Cuadros de Texto (*TMemo*) permiten tener varias líneas de texto. El texto dentro del cuadro de texto, es decir, sus líneas, también son *TStrings* con el nombre "Lines".
- Para agregar líneas al Memo, se utiliza `NombreDelMemo.Lines.Append()`
- La función **`Append()`** llamada en ambos casos pertenece a la clase *TStrings* que está tanto en el *RadioGroup.Items* como en el *TMemo.Lines*.
- Cuando se utiliza una función que no pertenece al formulario, para acceder a los controles visuales del formulario, es necesario escribir explícitamente la ruta jerárquica respectiva. Algo como: `"Form1.ControlVisual.PropiedadOEvento"`.

9. Aplicación con ComboBox

Un *ComboBox* es una combinación de un cuadro de edición y una Lista (desplegable) permitiendo elegir una de varias opciones. Por esto también se le puede llamar de *Cuadro Combinado*. Hay muchas formas de utilizar este tipo de componente.

9.1 – Crea un nuevo proyecto aplicación.

9.2 – Agrega al formulario Form1 un cuadro combinado TComboBox desde la pestaña Standard de la paleta de Componentes.

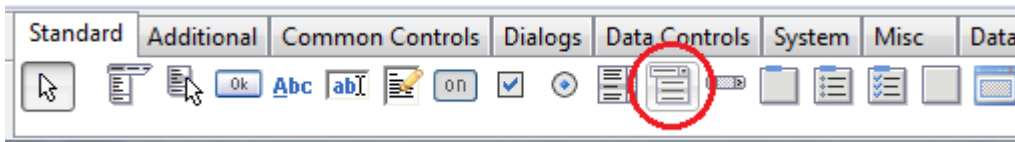


Figura 62: Seleccionando un TComboBox.

9.3 - En el cuadro combinado, las cadenas almacenadas se guardan en la propiedad Items, que es del tipo TStringList. De este modo se puede asignar o quitar cadenas al ComboBox, como en una TStringList o su madre TStringList.

9.4 – Hay dos formas básicas de cargarle opciones a un ComboBox:

9.4.1 – Usando el Inspector de objetos:

- Selecciona el combobox haciendo un clic sobre él.
- Vaya al *Inspector de objetos*, selecciona la pestaña Propiedades y la propiedad Items.
- Haga un clic en el botón de los tres puntos para abrir el Editor de Strings.
- Ingrese el texto para las opciones deseadas y confirma con [Aceptar].

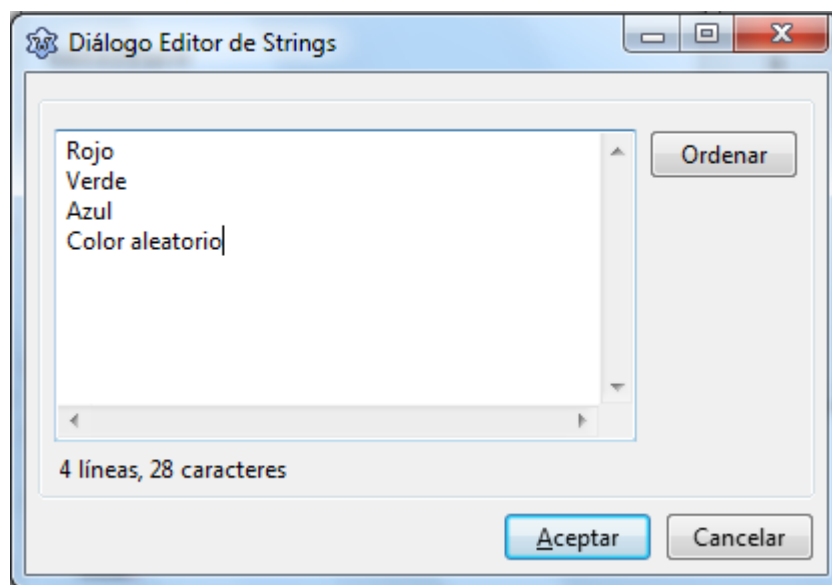


Figura 63: Opciones del combobox definidas en la propiedad Items.

- Ejecuta la aplicación [F9] para comprobar el funcionamiento del combobox.

9.4.2 – Por código, cuando se crea el formulario:

- Si agrega otro combobox al formulario, normalmente se llamará Combobox2.

- Haga un clic en el formulario y vaya al *Inspector de objetos*, selecciona la pestaña *Eventos* y luego el evento *OnCreate*.
- Haga un clic en el botón de los tres puntos para crear el código inicial del procedimiento *FormCreate*.
- En el editor de código ingresa el texto para las opciones deseadas, para nuestro ejemplo, ingresa el código abajo.

```
procedure TForm1.FormCreate(Sender: TObject);
begin
    ComboBox1.Items.Clear;           //Borra las opciones existentes
    ComboBox1.Items.Add('Rojo');     //Agrega una opción...
    ComboBox1.Items.Add('Verde');
    ComboBox1.Items.Add('Azul');
    ComboBox1.Items.Add('Color aleatorio');
end;
```

- Vuelva a probar la aplicación (F9), para ver que el cuadro combinado sigue funcionando igual.
- Si lo desea, puede eliminar las opciones de la propiedad *Items* y volver a ejecutar la aplicación para asegurar que ahora es este código el responsable de poblar el *Combobox*.

9.5 – Hagamos que algo suceda con la opción elegida.

Como todos los componentes, incluso el *TComboBox* ofrece varios eventos, que son llamados cuando el usuario utiliza el desplegable. Para responder a un cambio de selección en el cuadro combinado, puede utilizar el evento *OnChange*:

9.5.1 – Haga un doble clic en el *Combobox* en el formulario o elija su evento *OnChange* en el inspector de objetos y haga clic en el botón [...].

9.5.2 – El procedimiento correspondiente al evento *OnChange* será creado. Ingresa el código abajo que permitirá cambiar el color de fondo del formulario según la elección del usuario.

```
procedure TForm1.ComboBox1Change(Sender: TObject);
begin
    //Determina cual ítem fue elegido actualmente
    case ComboBox1.ItemIndex of
        0: Color:= clRed;
        1: Color:= clGreen;
        2: Color:= clBlue;
        3: Color:= Random($1000000);
    end;
end;
```

Nota: las constantes *clRed*, *clGreen* y *clBlue* son constantes de Lazarus para definir los colores rojo, verde y azul. Otros colores son definidos en formas semejantes.

9.5.3 – Ejecuta la aplicación y observa que el color de fondo del formulario cambiará de acuerdo con la opción seleccionada.

9.6 – Así como está, el *ComboBox* trae inicialmente el nombre del objeto y permite cambiar el texto de las opciones existentes. Para cambiar esto:

- Selecciona el *ComboBox*.

- Vaya al Inspector de objetos y elimina "ComboBox1" de la propiedad Text.
- Cambia la propiedad Style a "csDropDownList".
- Si desea que se muestre inicialmente el primer ítem de la lista, cambia la propiedad ItemIndex a 0(cero). Verás que la opción "Rojo" aparecerá en el combo, aún en tiempo de diseño.

Nota: *si has eliminado la lista de opciones de la propiedad Items, puede que no le deje cambiar ItemIndex, así que esta configuración también deberás hacerla en el evento OnCreate del Form (como en 9.4.2), agregando allí la siguiente instrucción:*

```
ComboBox1.ItemIndex:=0;
```

En este caso, es lógico asumir que no se podrá ver el primer ítem en tiempo de diseño. Solo en tiempo de ejecución.

- Ejecuta la aplicación con [F9] y observa el cambio de comportamiento.

9.7 – Así como está todo, nuestro *ComboBox* trae inicialmente el nombre de la primera opción ("Rojo") pero el fondo del formulario no refleja dicha acción. Para cambiar esto:

- Volvemos al código del evento OnCreate del form, como lo hicimos en 9.4.2.
- Agregamos la siguiente línea de código al final de la definición del Combobox (justo antes del **end;**).

```
ComboBox1Change(self);
```

- Este comando ejecutará código del evento *OnChange* que hemos creado para nuestro ComboBox (en 9.5.2), luego de crear las opciones del mismo y asignarle la opción predeterminada (ItemIndex=0). El efecto será tal y como sucedería si hubiéramos elegido dicha opción interactivamente.

Note en la definición del evento *ComboBox1Change*, que este recibe como parámetro un "Sender" del tipo TObject. Por lo tanto, al llamar dicho evento, debemos indicar el propio Combobox con la palabra reservada **Self**. Veremos este tema con más detalles en el ítem [12 – Eventos con parámetros](#) de este mismo tópico.

- Ejecuta nuevamente la aplicación y verá que ahora el fondo ya aparecerá en rojo, que es color que aparecerá en el ComboBox.

9.8 – Ideas a retener:

- *Operativamente, existen muchas semejanzas entre el RadioGroup y el ComboBox, como sean:*
- *En el control visual ComboBox se elige una de varias opciones de una lista.*
- *Las opciones del ComboBox también pertenecen a un TStringList llamado Items.*
- *El ítem seleccionado actualmente es indicado por la propiedad ItemIndex (-1 significa que nada está seleccionado, 0 es el primer ítem, 1 el segundo y así sucesivamente).*

- Es posible tener ítems iguales, ya que dichos ítems son numerados.
- Los ítems pueden ser indicados en tiempo de diseño, en el Inspector de Objetos, pero también pueden ser agregados en tiempo de ejecución.
- Para agregar ítems a la lista desplegable del ComboBox, se utilizó la función `NombreDelCombo.Items.Add()`, pero también se podría utilizar la función `Append()` del `TStrings`.
- Es posible ejecutar la acción esperada del objeto ComboBox, vía programación, llamando a su evento `OnChange` y enviando como parámetro una referencia al propio objeto (`self`).

10. Temporizador

- 10.1 – Crea una nueva aplicación.
- 10.2 – Agrega al formulario Form1: un cuadro de edición de texto TEdit y un botón TButton.
- 10.3 – Haga con que el texto inicial en el arranque de la aplicación sea "0" (cero).
- 10.4 – Haga con que cada vez que se presione el botón sea ejecutado el siguiente código:

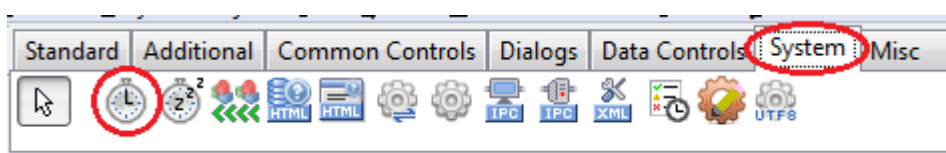
```
Edit1.Text:= Edit1.Text + '!' ;
```
- 10.5 – Graba el proyecto en una carpeta adecuada y ejecútalo.
- 10.6 – Confirme que cada vez que se presiona el botón, un punto de exclamación es agregado al texto.
- 10.7 – Modifica el evento OnClick del botón para:

```
Edit1.Text:= IntToStr(StrToIntDef(Edit1.Text,0) + 1);
```

Este código convierte el texto del Edit1 en un número entero, suma 1 y luego pone el resultado de vuelta en el mismo control, bajo el formato de cadena.

Nota: la función `StrToIntDef()` permite transformar una cadena a entero, indicando un valor predeterminado como segundo parámetro, el cual será devuelto en caso de que el valor de la cadena contenga caracteres inválidos o un formato inválido.

- 10.8 – Graba y ejecuta para confirmar que cada vez que el botón es presionado, el número en el cuadro de edición es incrementado.
- 10.9 – A seguir, agrega un temporizador TTimer (de la pestaña System de la Paleta de Componentes) al formulario.



- 10.10 – Haga un doble clic sobre el Timer que acaba de poner en el formulario para ser llevado hasta el evento “OnTimer” y completa dentro del procedimiento Timer1Timer el siguiente código con la llamada:

```
Button1Click(Sender);
```

Este código será llamado periódicamente por el timer y entonces llamará a la rutina que se ejecuta cuando el botón es presionado.

- 10.11 – Graba y ejecuta – observe que la numeración es incrementada a cada segundo automáticamente y también siempre que se oprima el botón. Fíjese también que el timer desaparece en tiempo de ejecución (¡pero sus funcionalidades están allí!).
- 10.12 – Ahora, modifica la propiedad Timer1.interval para 2000.
- 10.13 – Ejecuta para ver que el número pasa a ser incrementado cada 2 segundos. Esto nos indica que el valor de dicha propiedad es expresado en milisegundos. Es decir, para cada segundo que queramos indicar como intervalo entre las ejecuciones del código OnTimer debemos ingresar 1000 unidades en esa propiedad.
- 10.14 – **Ejercicio libre:** Agrega una CheckBox “CBHabilita” que cuando seleccionada permita el funcionamiento del incremento automático. La checkbox debe aparecer inicialmente seleccionada.
Pista: Utiliza Timer1.Enabled:=...

10.15 – Ideas a retener:

- *Cuando no hay eventos, no hay código de aplicación a ser ejecutado.*
- *Los temporizadores normales del Lazarus / Windows son poco precisos y se aconseja tiempos siempre superiores a 0.1 segundos.*
- *Existen varios controles que solo son visibles en tiempo de diseño, sin embargo cumplen con sus funciones en tiempo de ejecución.*
- *Podemos utilizar la propiedad Enabled de determinados controles, tanto en tiempo de diseño como en tiempo de ejecución, para activar o desactivar las funciones relativas a los mismos.*

11. Menú de la aplicación

- 11.1 – Agrega a la aplicación anterior un TMainMenu desde la pestaña Standard.
- 11.2 – Haga un doble clic sobre el componente agregado para abrir el *Editor de Menús*:



Figura 65: El Editor de Menús en su estado inicial.

- 11.3 - En el Inspector de Objetos, haga clic en el Caption “New Item1” y cámbialo a “&Archivo” (sin las comillas).

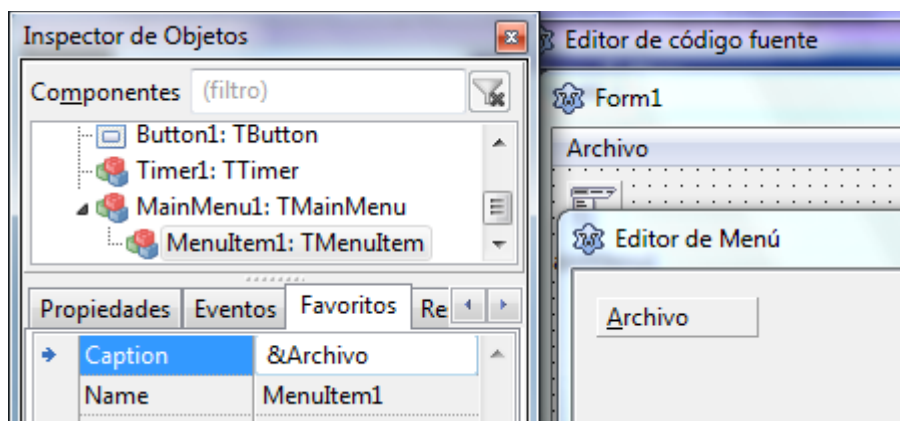


Figura 66: Primera opción del menú principal definida como Archivo.

- 11.4 – En el Editor de Menú, haga clic con el botón derecho del ratón sobre el botón [Archivo] y elija la opción Crear Submenú.

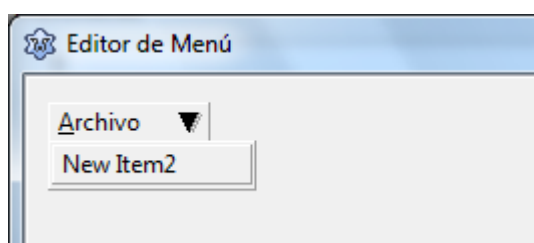


Figura 67: Creando un submenú para el menú Archivo.

- 11.5 – De vuelta al Inspector de Objetos, haga clic sobre el Caption New Item2 y cámbialo a "i&Salir!" y su propiedad Name para "MenuArchivoSalir".
- 11.6 – Haga un clic derecho en el botón [Archivo] y elija "Insertar Nuevo Elemento (después)".

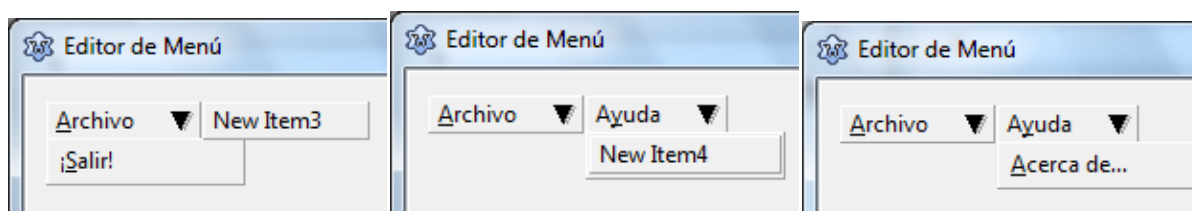


Figura 68: El Editor de Menú, definiendo el menú Ayuda ...

- 11.7 – Haga un clic en el último elemento de menú y modifica su Caption para "A&yuda", como vemos en la figura arriba.
- 11.8 – Agrega un submenú al Ayuda y al nuevo ítem cambia su Caption para "&Acerca de..." y su nombre para "MenuAyudaAcercaDe".
 Observa en la **Figura 69** que, para mayor facilidad de edición de estas propiedades hemos utilizado la pestaña Favoritos del *Inspector de Objetos*.

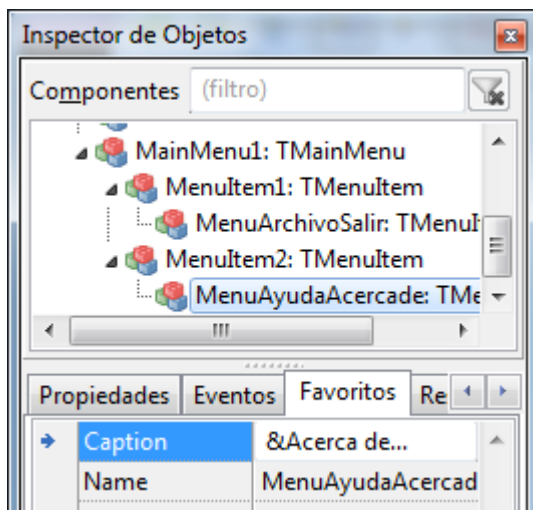


Figura 69: Propiedades de la opción de menú Acerca de...

- 11.9 – Cierra el *Editor de Menú* y aun en tiempo de diseño, verifica la funcionalidad de las opciones recién creadas. Debemos tener un menú Archivo con la opción ¡Salir! y otro menú Ayuda con la opción Acerca de...

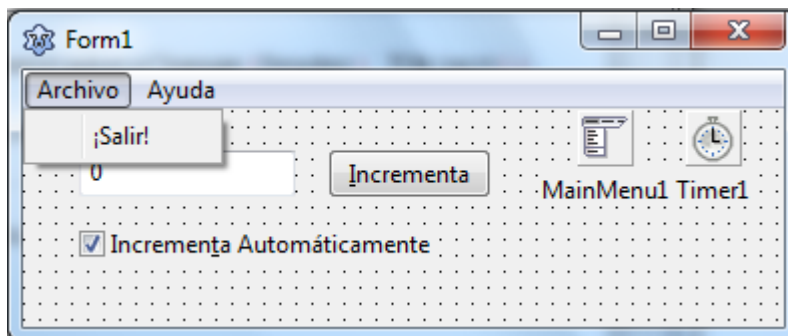


Figura 70: La opción ¡Salir!, del menú archivo, en tiempo de diseño.

- 11.10 – Como en la figura arriba, selecciona Archivo -> ¡Salir! y, en el *Editor de Código*, completa el código indicado abajo.

```

55 procedure TForm1.MenuArchivoSalirClick(Sender: TObject);
56 begin
57     Form1.Close;
58 end;

```

Figura 71: Código para el evento OnClick de la opción de menú Archivo->Salir.

Fíjese que el procedimiento generado tiene como nombre el contenido de la propiedad Name del ítem, seguido de la palabra "Click". De ahí la importancia de nombrar descriptivamente al menos aquellos ítems que vayan a ejecutar acciones. Este detalle nos facilitará en mucho la lectura del código en mantenimientos futuros.

- 11.11 – Aun en tiempo de diseño, selecciona Ayuda -> Acerca de y completa con el siguiente código:

```

63 procedure TForm1.MenuAyudaAcercadeClick(Sender: TObject);
64 begin
65     ShowMessage('¡Mi proyecto Rocks!!');
66 end;

```

Figura 72: Código para el evento OnClick de la opción de menú Ayuda -> Acerca de.

- 11.12 – Graba y ejecuta para probar las funcionalidades agregadas.
- 11.13 – Abra el editor de menús y agrega un ítem al menú Archivo con el caption “-” y otro con el caption “Salir...” y el nombre “MenuSalirConsulta”. Para esto, primero haga un clic derecho en el botón de la opción ¡Salir! y elija “Insertar Nuevo Elemento (después)”. Repita la acción para la siguiente opción.

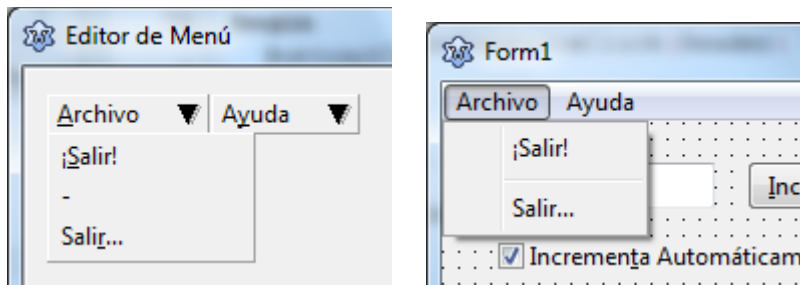


Figura 73 – La nueva opción Salir... con un separador de menús.

- 11.14 – Cierra el editor de menús, selecciona Archivo -> Salir... y completa el código como se muestra en la figura abajo.

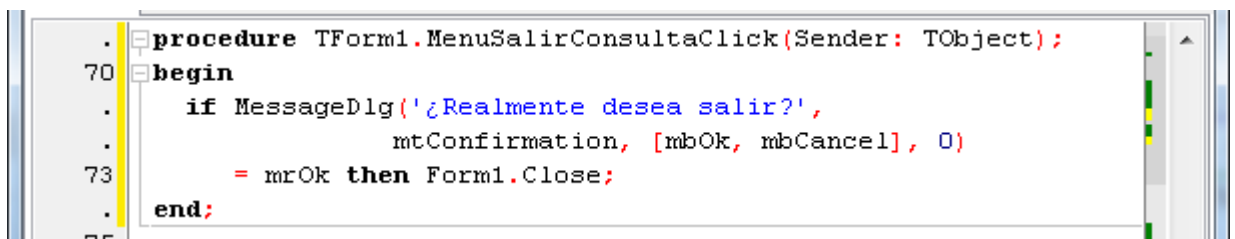


Figura 74: Código para el evento OnClick de la opción Salir...

Nota: es una buena práctica que el Caption de las opciones que abren ventanas de confirmación o diálogos tengan reticencias (...) al final.

- 11.15 – Ejecuta y prueba las funcionalidades de la aplicación.
- 11.16 – Habrá observado que los botones mostrados en la ventana de confirmación, que se ejecuta por medio de la función MessageDlg, tanto los captions de los botones como el de la propia ventana están en idioma inglés. Esto es debido a que la Librería de Componentes de Lazarus (LCL) está escrita en ese idioma y por lo tanto, el valor predeterminado de las constantes mtConfirmation, mbOk y mbCancel que definen dicho diálogo también lo están.

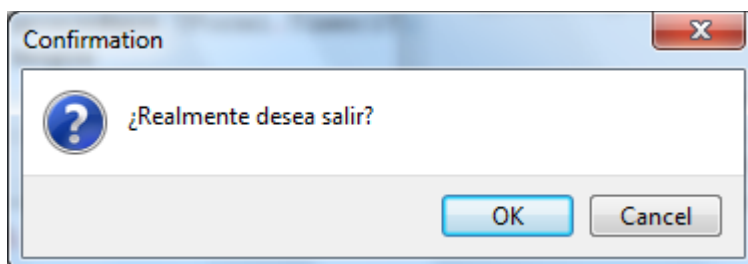


Figura 75: Ventana de confirmación ejecutada por la opción Salir...

Por suerte, la LCL ya viene con los archivos necesarios para ser traducida a otros idiomas, así que lo único que nos hace falta es cargar el que necesitamos.

Para hacer con que estos componentes aparezcan en el idioma de su preferencia, abajo le indicaremos la forma correcta de **cómo cambiar el idioma de las constantes de mensajes de un proyecto Lazarus:**

- 11.16.1 – En la misma carpeta donde se encuentra el archivo del proyecto, crea una subcarpeta llamada "languages".
- 11.16.2 – Suponiendo que tienes a Lazarus instalado en la unidad C:, adentro de la carpeta recién creada copia el archivo correspondiente al idioma español que es el: "C:\lazarus\lcl\languages\lclstrconsts.es.po"
- 11.16.3 – Por medio del menú Proyecto -> Ver Fuente Proyecto, abra el código fuente de su proyecto (archivo .lpr) y añada a la sección **uses** la unidad translations. Por ejemplo:

```

. { you can add units after this }
. ,translations;
13
. {$R *.res}

```

Figura 76: Agregando la unidad de traducción a la cláusula Uses.

- 11.16.4 – **Antes de inicializar** la aplicación, efectuamos la traducción del recurso externo mediante algo como esto:

```

begin
...
  TranslateUnitResourceStrings('LCLStrConsts',
    'languages\lclstrconsts.%s.po', 'es', '');
  Application.Initialize;
...
end.

```

Fíjese que la función TranslateUnitResourceStrings() es la que se encarga de la traducción. De esta misma manera, es posible cargar traducciones de otros elementos externos que dispongan de sus correspondientes archivos ".po".

- 11.16.5 – Guarda todo y ejecuta para confirmar que nuestro diálogo ya aparece en el idioma esperado.

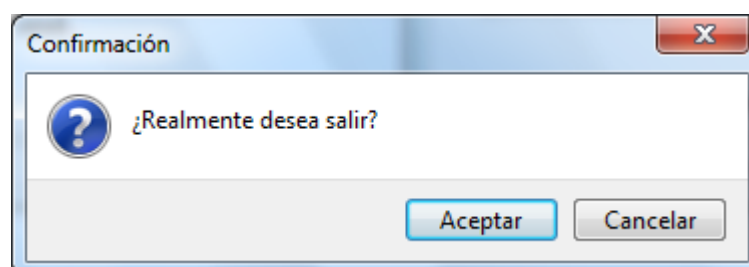


Figura 77: Diálogo de confirmación, ahora en el idioma español.

- 11.16.6 – **¡MUY IMPORTANTE!**

Como *Lazarus* es un IDE en constante evolución, a cada versión de Lazarus que es emitida también se emiten nuevas versiones de los archivos de traducción de constantes. Por este motivo, si vamos a compilar un proyecto creado con una versión anterior a la que tenemos instalada, es importante actualizar el archivo de la carpeta [languages] de dicho proyecto con el correspondiente a la versión actual, de la misma forma como hemos visto en el tópico 11.16.2.

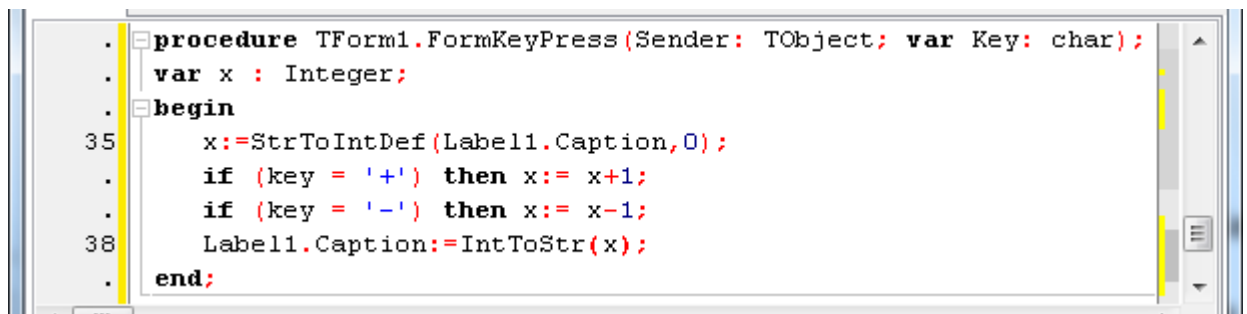
11.17 – Ideas a retener:

- *Los menús son una funcionalidad básica del Sistema Operativo.*
- *Los menús deben tener nombres cortos, descriptivos y que no se presten a confusiones – recuérdese que el usuario no es programador.*
- *Es interesante agrupar las acciones a ejecutar por temas y así tener menús (y submenús...) acordes a una determinada lógica.*
- *Buenas prácticas:*
 - *Definir teclas de atajo para los menús utilizados más frecuentemente.*
 - *Una "Opción..." da la idea de que aparecerá un diálogo adicional.*
- *Aunque el idioma predeterminado de los diálogos creados por Lazarus es el inglés, es posible cambiar dicho idioma en nuestro proyecto.*

12. Eventos con parámetros

12.1 – Crea una nueva aplicación e instancia una etiqueta TLabel en el Form1.

12.2 – En la lista de eventos del formulario, defina el evento OnKeyPress (oprimiendo el botón [...] correspondiente) y completa el respectivo procedimiento tal y como se muestra en la figura abajo.



```

. procedure TForm1.FormKeyPress(Sender: TObject; var Key: char);
.   var x : Integer;
.   begin
35     x:=StrToIntDef(Label1.Caption,0);
.     if (key = '+') then x:= x+1;
.     if (key = '-') then x:= x-1;
38     Label1.Caption:=IntToStr(x);
.   end;

```

Figura 78: Código para el evento OnKeyPress del formulario.

Los eventos reciben siempre la identificación del control que lo generó por medio del parámetro de entrada "Sender" en el procedimiento que atiende a dicho evento. El procedimiento *FormKeyPress* atiende a los eventos OnKeyPress del Form1 y recibe no solamente el Sender, sino que también recibe la tecla presionada por medio del parámetro de entrada "key". La variable *x* es definida en el código como siendo del tipo entero y es local a este procedimiento (su valor no se mantendrá entre las llamadas al procedimiento).

12.3 – Graba el proyecto en una carpeta específica y ejecútalo para comprobar que oprimiendo las teclas [+] y [-] se verá un número en la etiqueta, incrementado o disminuido en una unidad a cada pulsación.

12.4– Ideas a retener:

- *Los procedimientos que atienden a los eventos, reciben parámetros.*
- *Uno de los parámetros es el "Sender", es decir, una referencia al objeto que generó la llamada al procedimiento actual.*

- Otros eventos reciben informaciones específicas del evento llamador. Por ejemplo: *OnKeyPress* recibe el carácter correspondiente a cuál fue la tecla oprimida.

13. Confirmar salida de la aplicación – Close Query – parámetros de salida en el evento

13.1 – Crea una nueva aplicación.

13.2 – En la lista de eventos del formulario, defina el evento *OnCloseQuery* y complétalo con el código mostrado en la figura abajo.

```
. procedure TForm1.FormCloseQuery(Sender: TObject; var CanClose: boolean);
. begin
.     if MessageDlg('¿Desea realmente salir?', mtConfirmation,
4     [mbOk, mbCancel], 0) = mrCancel then CanClose:=False;
5 end;
```

Figura 79: Código para el evento *OnCloseQuery*.

Como ya hemos visto en el tópico II-11, la función *MessageDlg* muestra una ventana de diálogo estándar del sistema operativo y pide confirmación de salida con los botones estándares [Aceptar] y [Cancelar].

En el procedimiento *FormCloseQuery* entran los parámetros *Sender* y “*CanClose*” que es un parámetro **var** del tipo “**boolean**”. El valor de esta variable a la salida del procedimiento será utilizado por el sistema operativo para validar realmente el pedido de salida del programa (en forma predeterminada, *CanClose:=True* lo que dirá al S.O. para cerrar de hecho la aplicación).

- 13.3 – Por medio de este código en este evento, sea cual sea la forma en que se pida la finalización de la aplicación, siempre será lanzada la ventana de diálogo. Muchos programas solo lanzan este tipo de ventana de diálogo si existen modificaciones que grabar.
- 13.4 – Guarda el proyecto y ejecútalo para observar que, al cerrar la ventana seremos consultados sobre la certeza o no de la acción.

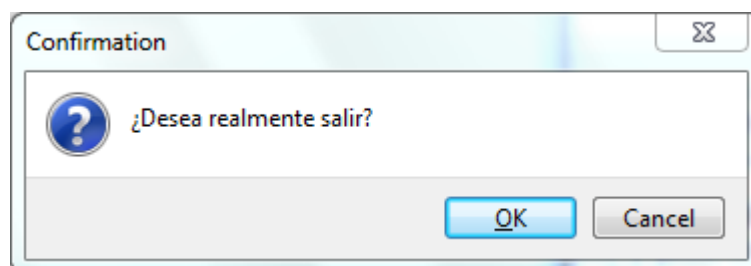


Figura 80: Ventana de confirmación lanzada por el evento *OnCloseQuery*.

Nota: Fíjese que, una vez más aparecen los Captions del diálogo en el idioma inglés. Esto es porque las acciones indicadas en el tópico II-11.16 deben ser ejecutadas para cada proyecto que vayamos a desarrollar, y en este último no lo hicimos.

14. El Depurador de Código – Debugger

Los programadores expertos saben que la depuración es fundamental para probar código complejo a fin de asegurarse de que funciona correctamente. Y a veces la depuración se convierte en absolutamente necesaria cuando un error aparece en el programa. La depuración ayuda a analizar mejor lo que está pasando en el fondo, cuando el programa se está ejecutando.

- A continuación se utilizará el Depurador de código (Debugger) del IDE de Lazarus. Ese depurador bajo Windows puede funcionar mal y puede, algunas veces, obligar a reiniciar el Lazarus (aunque esta situación actualmente es rara, es mucho menos frecuente en Linux) o puede ser necesario usar la opción de menú Ejecutar -> Reiniciar depurador.
- Lazarus viene con el GDB (GNU Debugger) como su depurador predeterminado.
- GDB no es parte de Lazarus, a menos que usted esté utilizando Windows o ciertas versiones de Mac OS X con XCode, habrá que instalarlo a parte de la herramienta Lazarus antes de usarlo. Vea la página web [Debugger Setup](#) en la wiki de Lazarus para mayores informaciones de cómo instalarlo y configurarlo correctamente.
- Para terminar una sesión de depuración, usar Ejecutar -> Detener (Ctrl+F2).

14.1 – Crea una nueva aplicación con un botón y completa el código de su evento OnClick según se muestra en la figura abajo.

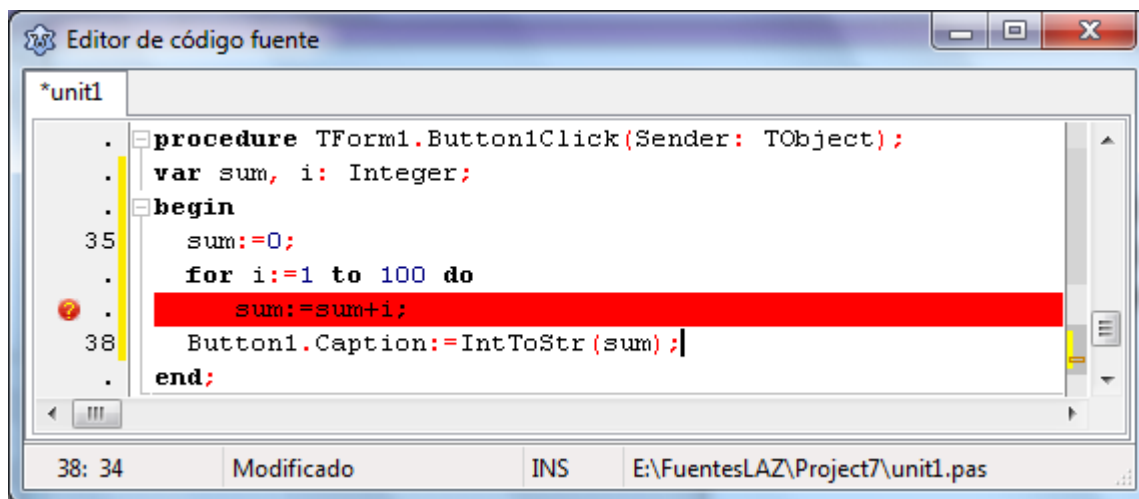



Figura 81: Código con un punto de interrupción del depurador establecido.

14.2 – Haga un clic con el ratón en la posición dónde está el símbolo  de interrogación en rojo para definir un *punto de interrupción* (break point) del depurador.

14.3 – Graba la aplicación en una carpeta apropiada y ejecuta con el [F9]. Verifica que el punto de interrupción aparece con un símbolo de confirmación.

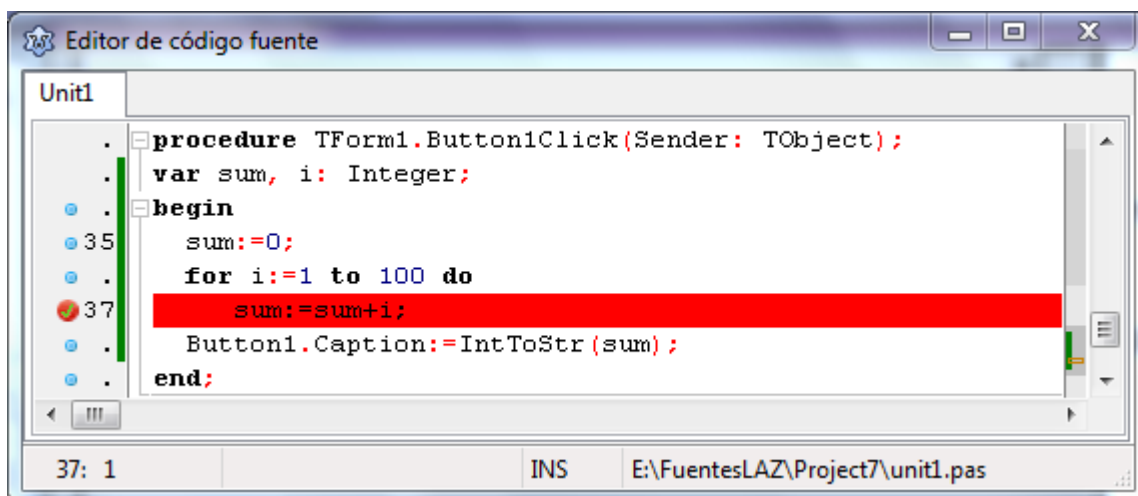


Figura 82: Punto de interrupción confirmado luego de compilar el proyecto.

- 14.4 – Oprima el botón de la aplicación – el evento será atendido, el procedimiento que responde al evento será llamado y el procedimiento `Button1Click` será ejecutado desde el inicio y seguirá. La ejecución será interrumpida en la posición del punto de interrupción.

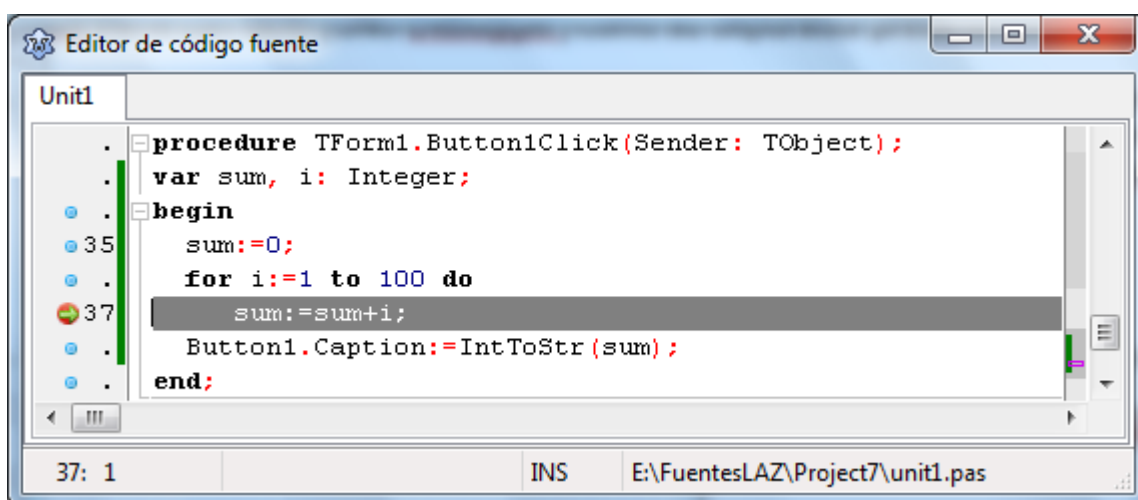


Figura 83: Ejecución del código detenida en el punto de interrupción.

- 14.5 – Durante este tiempo de depuración, es posible ver el valor de las variables pasando el puntero del ratón por encima de los respectivos nombres de dichas variables.
- 14.6 – Oprima la tecla [F7] repetidamente para ver que el programa sea ejecutado paso a paso y mientras tanto puede inspeccionar las variables en cuestión.
- 14.7 – Retire el punto de interrupción haciendo un clic sobre el mismo y luego oprima la tecla [F9] para ejecutar el resto del programa normalmente.
- 14.8 – Durante la depuración de código, no es posible acceder al formulario de la aplicación.
- 14.9 – La tecla [F7] hace "step into" y salta hacia adentro de los procedimientos que vayan siendo llamados (Paso a paso por instrucciones).
- 14.10 – La tecla [F8] hace "step over" y entra y sale de un procedimiento sin mostrarlo, pasando a la línea siguiente del procedimiento llamador.
- 14.11 – La tecla [F9] ejecuta el resto del programa normalmente.

14.12 – Consulte el menú Ver -> Ventanas de depuración ->

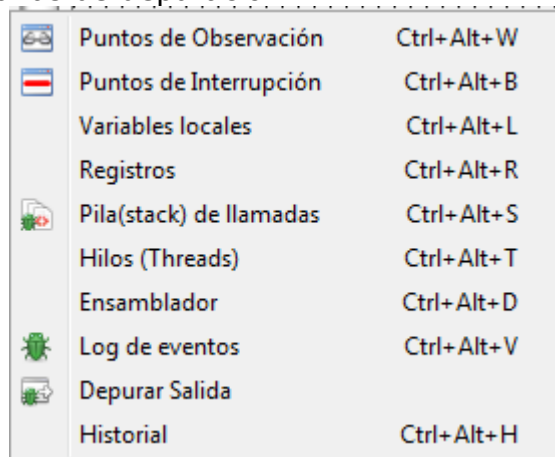


Figura 84: Menú Ventanas de depuración

- La ventana Puntos de Observación (Watches), permite definir las variables que queremos acompañar.
- Variables locales permite ver **todas** las variables locales de la función en ejecución.
- Pila(stack) de llamadas permite ver la pila de llamadas a funciones y procedimientos. La línea superior es la función actual, la línea más abajo es el programa principal.

15. Comentarios finales del tutorial básico

Este conjunto tutorial sirve para dar bases a que los alumnos puedan explorar por sí mismos las funcionalidades de Lazarus.

El Lazarus es una herramienta genérica de inmenso tamaño...

Queda el reto de probar esta herramienta en Linux y otros sistemas operativos. Como ya hemos comentado, es útil recordar que es posible portar proyectos entre Windows y los otros sistemas operativos.

III – EJERCICIOS DE LAZARUS

1. “Hola mundo” en un botón que se mueve.

- Crea una aplicación con un botón en un formulario.
- Crea un procedimiento que cuando se oprima el botón modifique el Caption del botón para “Hola mundo!!”
- Modifica el procedimiento anterior de tal manera que el botón se mueva por el formulario: cada vez que se lo oprima, el botón debe moverse 20 píxeles a la derecha y abajo. Antes que cualquier parte del botón salga del formulario este debe ser recolocado al centro del mismo. Asegura que el programa funcione cuando el formulario tenga cualquier tamaño.
- **Consejos prácticos:**
 - Los controles de Lazarus en general tienen las propiedades Top, Left, Width y Height.
 - Utiliza el operador **div** para divisiones enteras.
 - Las posiciones y dimensiones son siempre en píxeles.
 - Los ejes de Lazarus tienen origen en el rincón superior izquierdo.

2. Edit, Botón y Memo

- Crea una aplicación que cada vez que se oprima un botón, agregue en una nueva línea de un Memo box el texto escrito en un Edit Box.
- Selecciona la opción correspondiente al apareamiento automático de barras de desplazamiento en el Memo.
- **Consejo práctico:**
 - Utiliza la función Memo.Lines.Append.

3. Cálculo de la operación elegida por Radio Group (resultado en Edit Box)

- Crea una aplicación con tres Edit Boxes, un Radio group y un botón.
- Dos cuadros de texto serán términos para una operación matemática a ser elegida por medio del Radio Group.
- El Radio Group puede elegir entre las siguientes operaciones: suma, resta y multiplicación.
- El cálculo será efectuado cuando se oprima el botón.
- **Consejos prácticos:**
 - Utiliza las funciones StrToInt e IntToStr
 - Utiliza el RadioGroup.ItemIndex

4. Cálculo de la operación elegida por Radio Group (descriptivo en Memo Box)

- Crea una aplicación con dos Edit Boxes, un Radio Group, un botón y un Memo.
- Completa el código de tal forma a obtener un programa que, al oprimir el botón, sea calculada la operación seleccionada en el Radio Group y su descripción (Término 1) (operación) (Término2) = (Resultado) aparezca en el Memo.
- **Consejos prácticos:**
 - Utiliza StrToInt, StrToIntDef e IntToStr
 - Utiliza RadioGroup.ItemIndex
 - Utiliza MemoResultados.Lines.Append
 - Utiliza la suma de cadenas (strings) para la concatenación.

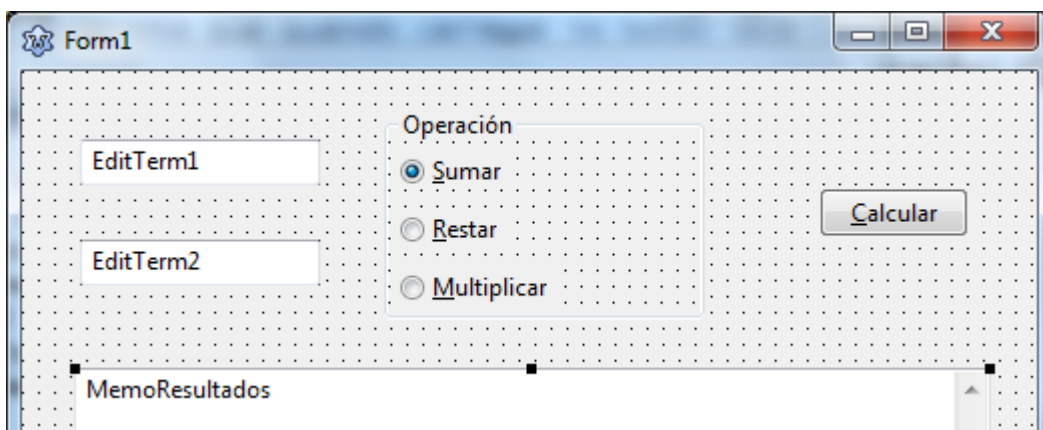


Figura 85: Aspecto del formulario creado para el ejercicio 4.

- Modifica el programa para que el resultado sea calculado siempre que haya modificaciones en cualquiera de los cuadros de texto (Edits).
- Consejos prácticos:
 - Utiliza el evento Edit.OnChange
 - Verifica que el programa puede generar errores y aun así sigue funcionando después que la situación de error haya sido corregida.
- Modifica el programa para que la operación sea calculada de 5 en 5 segundos
- **Consejo práctico:** Utiliza un control TTimer (pestaña System).

5. Diseño básico (Canvas)

- Crea una aplicación y ponga un Timer que llame al siguiente código a cada segundo.

```
with Form1 do begin
    Canvas.Ellipse( -2+Width div 2, -2+Height div 2,
                   2+Width div 2, 2+Height div 2);
    Canvas.Line ( Width div 2, Height div 2,
                  9+Width div 2, Height div 2);
end;
```


Estudia este código para entenderlo.

Ejecuta y redimensiona el formulario para ver el efecto.

- Modifica el programa para que haya una variable global llamada "rotacion" que defina el ángulo de rotación del marcador. Agrega un botón que incremente $\pi/10$ radianos.

Consejos prácticos:

- Las variables globales son definidas fuera de cualquier función (en el área de interface o de implementación) y son conocidas desde allí hacia adelante en el código.
- Ponga atención en el sentido de rotación que se intenta obtener y en los ejes del Lazarus.
- Utiliza las funciones Sin y Cos que reciben argumentos en radianos.
- El Lazarus conoce la constante π .
- Tendrá que utilizar las funciones Round o Trunc para convertir de real a entero.
- Comande la rotación con el teclado: por ejemplo "+" y "-" hacen girar el marcador en una y otra dirección.






Consejos prácticos:

- Ponga como verdadera (True) la propiedad Form.KeyPreview.
- Utiliza el evento Form.OnKeyPress.

6. Aplicación multi-ventana.

- Crea una aplicación con tres botones en un formulario. Al oprimirse el primero se abre un segundo formulario, oprimiéndose el segundo botón se cierra el segundo formulario y al oprimirse el tercer botón abre el formulario en modo "modal".
- No olvide que, al crear un nuevo formulario, también se creará una nueva unidad. Esta unidad deberá ser llamada por el **Uses** del primero para que dicho Form pueda ser reconocido por el mismo.
- **Consejos prácticos.**
 - Utilizar Archivo -> Nuevo formulario o el botón correspondiente en la barra de accesos directos.
 - Ver Proyecto -> Opciones del proyecto -> Formulario
 - Utilizar los métodos Form.Show, Hide, ShowModal
 - La tecla [F12] alterna entre el formulario y la Unidad activa en el Editor de Código.
 - Para mayores referencias, visite a: http://wiki.freepascal.org/Form_Tutorial (sitio en inglés)

IV – CONEXIÓN DE LAZARUS A UNA BASE DE DATOS PGSQL

- Es muy fácil acceder a Bases de Datos desde *Lazarus*. Para acceder a bases de datos creadas con PostgreSQL, podemos utilizar el componente TPQConnection  que está en la pestaña SQLdb de la *Paleta de Componentes*. Este componente registra todos los datos relativos al servidor de bases de datos, al usuario y respectivas contraseñas.
- Para acceder a los datos de la BD, es necesario que haya una protección contra órdenes contradictorias simultáneas de diferentes usuarios y para tal efecto se utiliza el objeto llamado TSQLTransaction , disponible en la misma pestaña de la paleta.
- Para realizar consultas (queries) hacia la base de datos es necesario el componente TSQLQuery .
- Un componente TDBGrid  (el cual se encuentra en la pestaña Data Controls), es una grilla automática con el formato de una tabla, dicha tabla puede ser una tabla temporal resultante de una consulta SQL. Para utilizar este componente aun necesitamos una forma de redireccionar los datos, lo que se consigue por medio del componente DataSource , el cual encontraremos en la pestaña Data Access de la misma paleta.

1. Diferentes formas de uso.

- Hay tres formas básicas de acceder a bases de datos bajo Lazarus.
 - Utilizando un componente **DBGrid** – visualizaciones simples.
 - Utilizando PQConnection.**ExecuteDirect** – modificar datos.
 - Utilizando SQLQuery.Fields – obtención de datos elegidos (respuesta a una consulta SQL).

2. Preparación

- Utiliza una herramienta de acceso a la base de datos PostgreSQL (por ejemplo *pgAdmin* en Windows o el *Gnome* en Linux), elija una base de datos de prueba y cambia su contraseña por alguna poco importante.
 - Ubica una tabla de prueba que contenga datos.
 - Prueba hacer una consulta de prueba para asegurar la disponibilidad de datos y el acceso a la tabla. Ejemplo: `select * from cualquiertabla`

3. Acceso utilizando DBGrid

- Esta es la prueba más visual, que presenta datos de la conexión a la base de datos tanto en tiempo de diseño como durante la ejecución de la aplicación. Su utilización está limitada a mostrar los datos en forma sencilla y automática, sin flexibilidad.

- **ATENCIÓN:** diversas configuraciones pueden evitar el modo de funcionamiento aquí descrito, pero los procedimientos que aquí presentamos fueron probados en una máquina con Windows 7 y una instalación local básica de PostgreSQL versión 9.2.9, dónde fue creada una base de datos de ejemplo llamada "pruebapg". Para las pruebas a seguir, se utilizó una tabla llamada "producto" con la siguiente estructura:

- iid – integer – No nulos - Identificador único del registro (PK).
- descprod – character varying (40) – No nulos – Descripción del producto.
- precunit – numeric (10,0) – No nulos – Precio del producto
- irubro – integer – No nulos – Código de rubro (default=0)

A esta tabla se agregaron ocho filas de datos de ejemplo para productos correspondientes a tres rubros diferentes.

- Construya su aplicación con aspecto similar al siguiente. La secuencia de operaciones y configuración es indicada luego después.

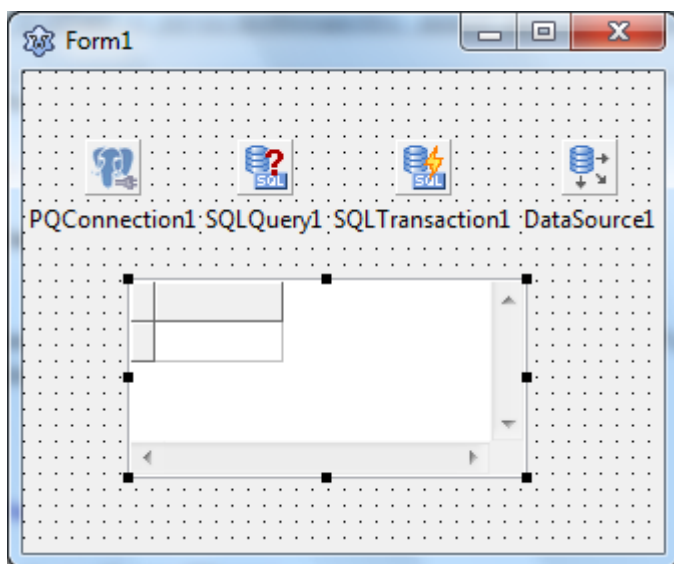


Figura 86: Componentes necesarios para la conexión a la BD.






Componente	Pestaña	Ícono
PQConnection	SQLdb	
SQLQuery	SQLdb	
SQLTransaction	SQLdb	
DataSource	DataAccess	
DBGrid	DataControls	

Tabla 1: Componentes necesarios y su ubicación en la paleta.

- En el **PQConnection1** completar los siguientes valores:

- DatabaseName: nombre de la BD a utilizar, por ejemplo: pruebapg
 - HostName: nombre del servidor de BD a utilizar, por ejemplo: localhost
 - Password: contraseña del usuario en el servidor de BD.
 - UserName: nombre del usuario en la BD, por ejemplo: postgres
 - Transaction: SQLTransaction1
- Confirmar que el **SQLTransaction** apunta a PQConnection1 (propiedad Database).
 - En el **SQLQuery** completar o confirmar:
 - Database: PQConnection1
 - SQL: completar con código SQL. Por ejemplo: select * from producto;

Para esto, deberá posicionarse en la propiedad y usar el [...] **Editor de código SQL** para ingresar la consulta.

 - Transaction: SQLTransaction1
- En el **DataSource** completar DataSet como SQLQuery1.
- En el **DBGrid** completar el DataSource como DataSource1.
- Cambia la propiedad PQConnection1.Connected a True y también la SQLQuery1.Active. Verifica que SQLTransaction1.Active también esté True. Si todo sale bien, aun en tiempo de diseño será posible ver el resultado de la consulta SQL hecha aparecer en la DBGrid.

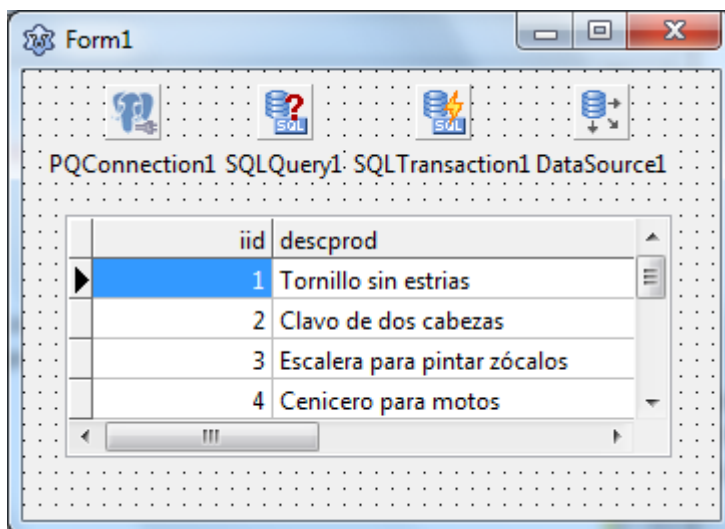


Figura 87: datos de la BD en PostgreSQL mostrados ien tiempo de diseño!

- **Atención:** en Windows, si estás usando por primera vez este recurso, en esta máquina, puede que al intentar conectar, salte el mensaje de error [Can not load PostgreSQL client library "libpq.dll". Check your instalation.] Para solucionar este problema, vea "[4. Lazarus, PostgreSQL y errores relacionados a libpq](#)" en el tópico VI de este documento, correspondiente a una pequeña FAQ Lazarus.

La solución más fácil a este problema es copiar los archivos **libpq.dll**, **libeay32.dll**, **libintl-8.dll**, **ssleay32.dll** desde la carpeta [Bin] de la instalación del PostgreSQL, hacia la carpeta system32 del SO. Por ejemplo: [c:\windows\system32].

- Guarda y ejecuta la aplicación para probar las posibilidades de navegar por la grilla, redimensionar las columnas, etc.
- Vamos a dedicar algunos momentos para explorar más algunas propiedades del DBGrid. Por ejemplo: habrá notado que, así como lo planteamos, se muestran todas las columnas de la tabla con sus anchos predeterminados por la longitud de cada una. Esto hace un poco difícil de visualizar sus datos en el form que acabamos de montar. Hay al menos dos formas de mejorar dicha visualización:
 1. Usa la propiedad de anclaje (Anchors) del DBgrid y demás objetos, a fin de que, al redimensionar el formulario, también se redimensione la grilla permitiéndonos mejor visualización de su contenido. Para esto:
 - En el Inspector de Objetos, elija la propiedad Anchors y haga clic en el botón [...].
 - En el *Editor de Anclaje*, activa las casillas referentes al Anclaje derecho y Anclaje inferior. Normalmente, el superior y el izquierdo ya vienen marcados, déjalos así.
 - Haga un clic derecho sobre la grilla y elija la opción Editor de Anclaje. Esta es otra forma de acceder a dicho editor.
 - Cierra el Editor de Anclaje.
 - Guarda y ejecuta para ver la nueva funcionalidad. Verifica que, al redimensionar el formulario, en tiempo de ejecución, la grilla acompañará las nuevas dimensiones a lo largo y a lo ancho del mismo.

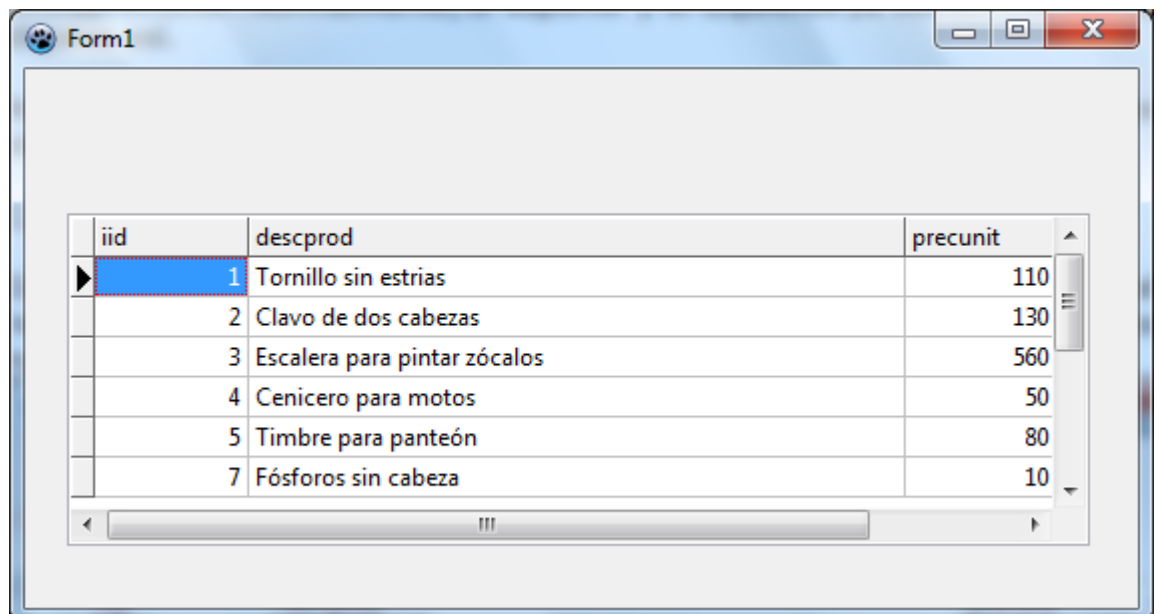



Figura 88: La grilla se adapta al formulario redimensionado.

- El efecto observado, también se reflejará en tiempo de diseño. Es decir, si redimensionamos el formulario, en estas condiciones, también se redimensionará la DBGrid. Si este no es el efecto deseado deberemos reacomodarla manualmente o desactivar los anclajes hasta terminar nuestro diseño.

2. Otra posibilidad es la de determinar manualmente las columnas que queremos ver en la grilla y, de esa forma configurar también dichas columnas. Para tanto debes actuar como sigue:

- Antes que nada, selecciona el SQLQuery1 y ponga su propiedad Active como False.
- Selecciona el DBGrid y en el Inspector de Objetos, elija la propiedad Columns y haga clic en el botón [...].
- En el Editor de Columnas del DBGrid, haga un clic en  para crear la columna 0.

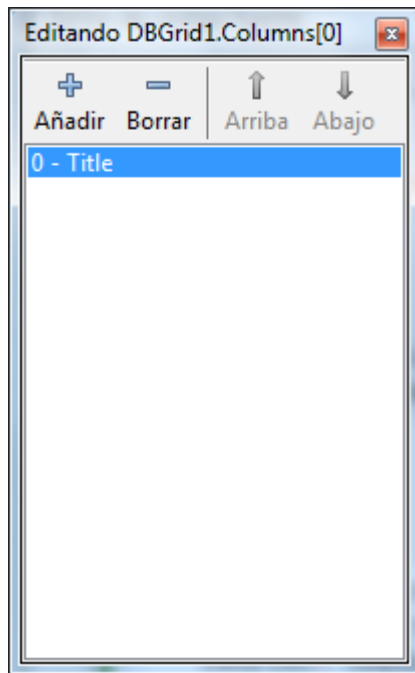


Figura 89: el Editor de columnas del DBGrid editando la columna 0.

- De vuelta al Inspector de objetos, elija la propiedad FieldName y, en el combobox asociado, indica el campo "iid".
- Haga un doble clic sobre la propiedad Title, si esta no está expandida, y selecciona Caption. Cambia el caption de la columna para "# ID".
- Finalmente, en la propiedad Width de la columna ponga el valor 40.
- Vuelva al panel de Edición de columnas y añada una columna más.
- Siga los mismos procedimientos anteriores para definir las columnas:
 - FieldName := irubro; Title.Caption := ID Rubro; width := 55
 - FieldName := descprod; Title.Caption := Nombre producto; width := 180
 - FieldName := precunit; Title.Caption := Precio; width := 70
- Vuelva a "Activar" al SQLQuery1 y se dará cuenta de que los datos se cargan en las columnas, pero estas mantienen las características con que fueron configuradas.

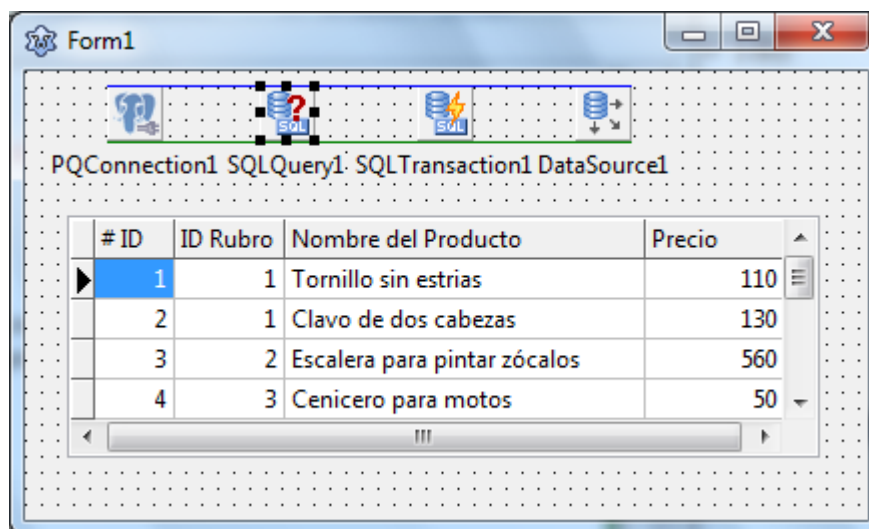


Figura 90: DBGrid con las columnas configuradas manualmente.

- Observa que, de esta manera, es posible determinar qué columnas queremos mostrar, en qué orden, anchos, títulos, etc.
 - Vuelva a ejecutar la aplicación y fíjese que, en estas condiciones, anclar a la derecha ya no es tan necesario.
3. Otra configuración que casi siempre deseamos, cuando mostramos columnas con valores, es la de visualizar los datos con separadores de miles y decimales (por ejemplo: 1.234.567,50). Para obtener este efecto, sólo debemos utilizar la propiedad **DisplayFormat** de la columna y agregar una cadena de formateo. Algo así como: `'###,##0.00'`. En la figura abajo vemos el resultado de implementar la configuración mencionada a la columna de Precio (precunit).

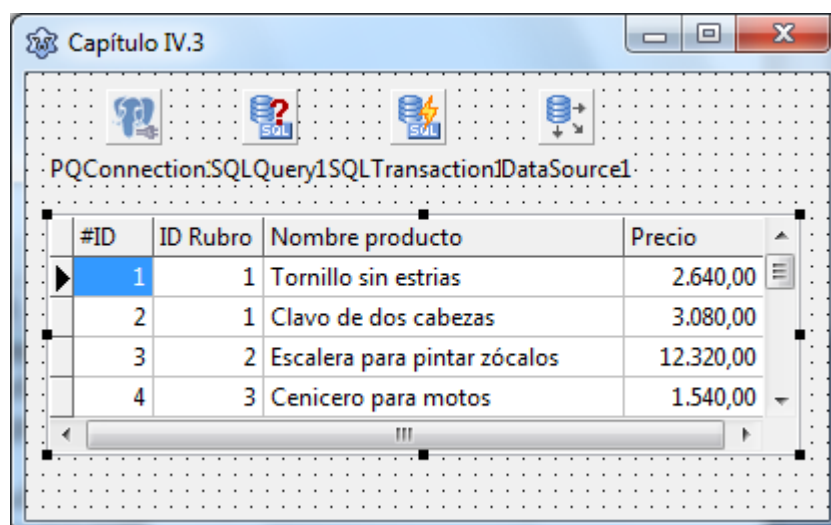


Figura 91: DBGrid con la columna Precio formateada con separadores.

Fíjese que, aunque la cadena de configuración utiliza la coma como separador de miles y el punto como separador de decimales, los valores mostrados tienen como separador de miles el punto y la coma decimal. Esto se debe a que Lazarus compara dicha configuración con la *Configuración Regional* del Sistema Operativo iiautomáticamente!!

Nota: para obtener los valores del ejemplo hemos modificado temporalmente los valores del campo precunit.

4. Ejecución directa de código usando la TConnection.

Es posible ejecutar comandos SQL directamente sobre la base de datos, usando la función ExecuteDirect de la conexión.

- Antes que nada, es importante aclarar al alumno que la aplicabilidad práctica o no de nuestros ejemplos no viene al caso. Aquí, lo importante es entender la forma como logramos dichos resultados.
- Para este ejemplo, en el form de la aplicación anterior, debajo de la grilla, agrega un objeto TLabel (pestaña Standard) con el caption "ID Rubro:", un EditBox con el nombre "EditIdRubro", al que eliminaremos el contenido de la propiedad Text, y un botón llamado CBElevaPrecio con el Caption "Precio+10".
- Agrega también un MemoBox llamado "MemoLog" para captar los posibles errores de conexión.
- Para modificar la Base de Datos, podemos hacer uso de un código similar al siguiente. Haga de este el código para el evento OnClick del botón CBElevaPrecio.

```
var
    enter, s : string;
    liRubro : integer;
begin
    liRubro:= StrToIntDef(EditIdRubro.Text,0);
    enter:= chr(13)+chr(10);
    s:= 'update producto set precunit = precunit + 10 where '+'
        'iRubro=' + IntToStr(liRubro); // cadena de código SQL
    try
        If NOT PQConnection1.Connected Then
            PQConnection1.Connected:= True;

        PQConnection1.ExecuteDirect('Begin Work;');
        PQConnection1.ExecuteDirect(s);
        PQConnection1.ExecuteDirect('Commit Work;');
        {Esto es para refrescar la grilla}
        SQLQuery1.Active:= False;
        SQLQuery1.Active:= True;
        {PQConnection.Connected:=False;}
    except
        on E : EDatabaseError do
            MemoLog.Append('ERRORBD:' + enter +
                E.ClassName + enter + E.Message);
        on E : Exception do
            MemoLog.Append('ERROR:' + enter +
                E.ClassName + enter + E.Message);
    end;
end;
```

- Guarda y prueba la aplicación. Si en el EditBox ponemos un ID de rubro existente y luego oprimimos el botón, el valor de la columna "Precio" de todos los productos correspondientes a dicho rubro deberá incrementarse diez puntos.

# ID	ID Rubro	Nombre del Producto	Precio
4	3	Cenicero para motos	70
5	3	Timbre para panteón	100
7	3	Fósforos sin cabeza	30
8	3	Borrador de carbón	35

ID Rubro: 3 Precio + 10

MemoLog

Figura 92: datos de ejemplo luego de pulsar el botón Precio+10.

- Observa que, con esta configuración de componentes, las filas modificadas van a parar al final de la grilla.
- De la misma forma, podríamos crear **nuevos registros** o **eliminar filas** existentes. Para tanto solo habría que modificar la cadena de código SQL (S) por el código correspondiente.
- **Ejercicios libres:** agrega un EditBox para aceptar el monto de alteración del precio y cambia el código del botón para usar dicho monto en lugar del 10 fijado actualmente. Agrega otro botón para reducir el precio. Considera crear un procedimiento con parámetros que ejecute la acción necesaria y que sea llamado por ambos botones (vea el ítem II.6).

5. Utilizar las consultas SQL (Fields)

- En la misma aplicación anterior, agrega un botón con el Caption "Consultar".
- Para obtener los campos y las filas resultantes de una consulta SQL, utiliza para dicho botón un código similar al que sigue:

```
...
var
  ln, col : integer;
  s : string;
begin
  SQLQuery1.SQL.Text:= 'select * from producto';
  PQConnection1.Open; // por si...
  SQLQuery1.Open;
  for ln:= 1 to SQLQuery1.RecordCount do begin
    SQLQuery1.RecNo:= ln;
    s:='';
    for col:=0 to SQLQuery1.FieldCount-1 do begin
      s:= s + SQLQuery1.Fields[col].AsString + '; ';
    end;
  end;
```

```

MemoLog.Append(s);
end;
{SQLQuery1.Close;}
{PQConnection1.Close;}
end;

```

- Certifíquese de que la propiedad ScrollBar del Memo está asignada como ssAutoBoth.
- Guarda y ejecuta la aplicación para probar la nueva funcionalidad.

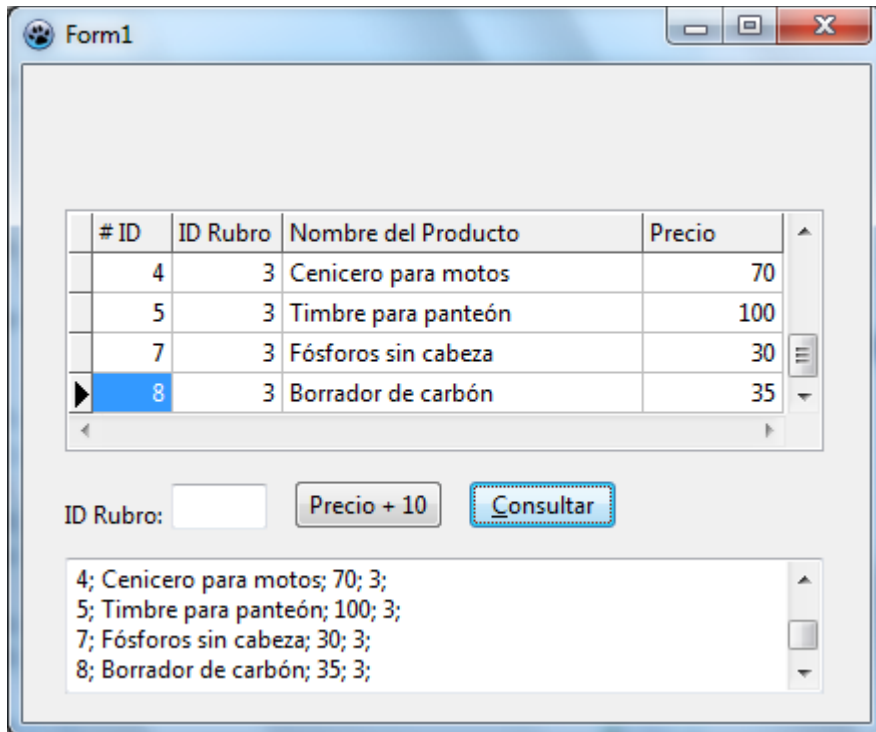


Figura 93: efecto del botón Consultar desplegado en el MemoLog.

- Fíjese que cuando se ejecuta el botón de Consulta, todos los datos de la tabla producto son agregados al MemoLog, dónde el valor de cada columna es separado del siguiente por un punto y coma (;).
- Además, el contenido de la grilla desaparecería si activamos la ejecución de los eventos Close en SQLQuery y PQConnection, al final del código.
- La respuesta a la consulta SQL tiene filas desde 1 hasta el valor de `SQLQuery.RecordCount` y las columnas van de 0 hasta `SQLQuery.FieldCount - 1`.
- Utilizando las propiedades mencionadas como límites, el código del botón recorre todas las filas obtenidas y, por cada fila, recorre todos los campos recibidos para montar la cadena "s" que es agregada al MemoBox.
- La fila actual puede ser obtenida e impuesta por medio de `SQLQuery.RecNo`. Este valor es utilizado para realizar el recorrido, posicionando el puntero adecuadamente.
- Cada columna puede ser de un tipo diferente de dato, por ejemplo la primera columna puede ser un entero, la segunda una cadena, por lo que – en otra situación – podría ser necesario indicar el tipo de datos en cuestión:
`SQLQueryFields[0].AsInteger` ó `SQLQueryFields[2].AsString`

6. Dejando al TSQLQuery la tarea de manejar los datos

Como vimos en el t3pico VI.4, si queremos, podemos usar el componente TConnection para realizar accesos directos a las bases de datos. Sin embargo, el componente TSQLQuery es el que tiene como principal finalidad la de recuperar y manejar datos por medio de un par conexi3n/transacci3n. Vamos a montar un ejemplo para ver c3mo esto funciona:

- o Crea una nueva aplicaci3n semejante a la que creamos en el 3tem 3 de este t3pico. Es decir, un formulario con los cinco componentes para acceso por medio de un DBGrid. Con esto tendr3s algo semejante al mostrado en la **Figura 87**.
- o Verifica que la configuraci3n de los componentes est3 correcta y que los datos se cargan en la grilla.
- o Por debajo del DBGrid, agrega cuatro etiquetas (TLabel) y cuatro cuadros de texto (TEdit). Organiza cada etiqueta con su cuadro de texto y cambia las propiedades Caption de cada TLabel por "#ID:", "ID Rubro:", "Descripci3n:" y "Precio". Algo semejante a lo mostrado en la **Figura 94**.

Nota: *f3jese que estamos cambiando el orden de los campos, respecto a c3mo est3n definidos en la tabla. Esto afectar3 nuestras instrucciones de inserci3n.*

- o A los TEdit, borra el contenido predeterminado de la propiedad Text. No olvide hacer el TEdit correspondiente a la descripci3n un poco m3s ancho que los dem3s.
- o Para completar, por debajo de todos, agrega un bot3n con el Caption "&Insertar".
- o Haga doble clic sobre el bot3n y agrega el siguiente c3digo a su evento OnClick:

```
procedure TForm1.Button1Click(Sender: TObject);
var
    iCodigo : integer;
    s: string;
begin
    iCodigo:= StrToIntDef(Edit1.Text,0);
    if iCodigo > 0 then
        begin
            { Note que respetamos el orden de los campos en la tabla
              para simplificar el comando insert }
            s:= 'insert into producto values (' + Edit1.Text+ ', ''
                +Edit3.Text+ ''', ' +Edit4.Text+ ', ' +Edit2.Text+ ')';
            SQLQuery1.Active:= False;
            SQLQuery1.SQL.Clear;
            SQLQuery1.SQL.Add(s);
            SQLQuery1.ExecSQL;
            SQLTransaction1.Commit;
            SQLQuery1.SQL.Clear;
            SQLQuery1.SQL.Text:= 'select * from producto;';
            SQLQuery1.Active:= True;
            Edit1.Text:='';
            Edit2.Text:='';
            Edit3.Text:='';
            Edit4.Text:='';
```

```

    ShowMessage('Registro almacenado con suceso.');
```

end

```

    else
        ShowMessage('El ID debe ser mayor a cero.');
```

end;

- Guarda y ejecuta la aplicación. Ingresa datos a los cuadros de texto y oprima el botón [Insertar]. **Atención:** como sólo estamos validando que algún valor entero sea ingresado para el ID, por ahora es importante ingresar uno que aun no exista en la tabla, a fin de evitar errores. Ya veremos cómo mejorar esto.
- Si se hizo todo bien, un nuevo registro será insertado en la tabla y podremos verlo en la grilla de datos, como muestra la figura abajo.

iid	descprod	prec
4	Cenicero para motos	
5	Timbre para panteón	
7	Fósforos sin cabeza	
8	Borrador de carbón	
9	Bocina para aviones	

#ID: ID Rubro:

Descripción:

Precio:

Figura 94: nueva fila insertada a la tabla de productos

- Como habrá notado, nuestro código hasta aquí solo valida que ingresemos un número entero para el ID. Pero siendo el ID una PK, el valor de este campo no puede estar repetido tampoco. Así que, para consultar su existencia en la base de datos, necesitaremos de **otro** componente TSQLQuery.
- Agrega otro componente TSQLQuery a tu formulario, desde la pestaña SQLdb y configura las siguientes propiedades:

```

Database:      PQConnection1
Transaction:   SQLTransaction1
```

- Ahora modifica el código del evento OnClick del botón para que quede tal y cual mostramos a seguir. Fíjese en las referencias al SQLQuery**2**.

```

procedure TForm1.Button1Click(Sender: TObject);
var
    iCodigo : integer;
    s: string;
begin
```

```

iCodigo:= StrToIntDef(Edit1.Text,0);
if iCodigo > 0 then
begin
    SQLQuery2.SQL.Text:= 'select iid from producto where iid='
                        +Edit1.Text+ ' ';
    SQLQuery2.Open;
    if SQLQuery2.RecordCount > 0 then
        ShowMessage('Identificador ya existe.')
    else
        begin
            s:= 'insert into producto values ('+Edit1.Text+ ', '' '
                +Edit3.Text+''', ' +Edit4.Text+', '+Edit2.Text+ ' );' ;
            SQLQuery1.Active:= False;
            SQLQuery1.SQL.Clear;
            SQLQuery1.SQL.Add(s);
            SQLQuery1.ExecSQL;
            SQLTransaction1.Commit;
            SQLQuery1.SQL.Clear;
            SQLQuery1.SQL.Text:= 'select * from producto;';
            SQLQuery1.Active:= True;
            Edit1.Text:='';
            Edit2.Text:='';
            Edit3.Text:='';
            Edit4.Text:='';
            ShowMessage('Registro almacenado con suceso. ');
        end;
        SQLQuery2.Close;
    end
else
        ShowMessage('El ID debe ser mayor a cero. ');
end;

```

- Guarda, ejecuta e intenta ingresar un producto cuyo ID ya exista. El resultado debería ser semejante al mostrado en la **Figura 94**.
- Como vemos aquí, por medio del SQLQuery también podemos ejecutar código directamente contra la base de datos. **Esta es la forma más utilizada para este tipo de acciones.**

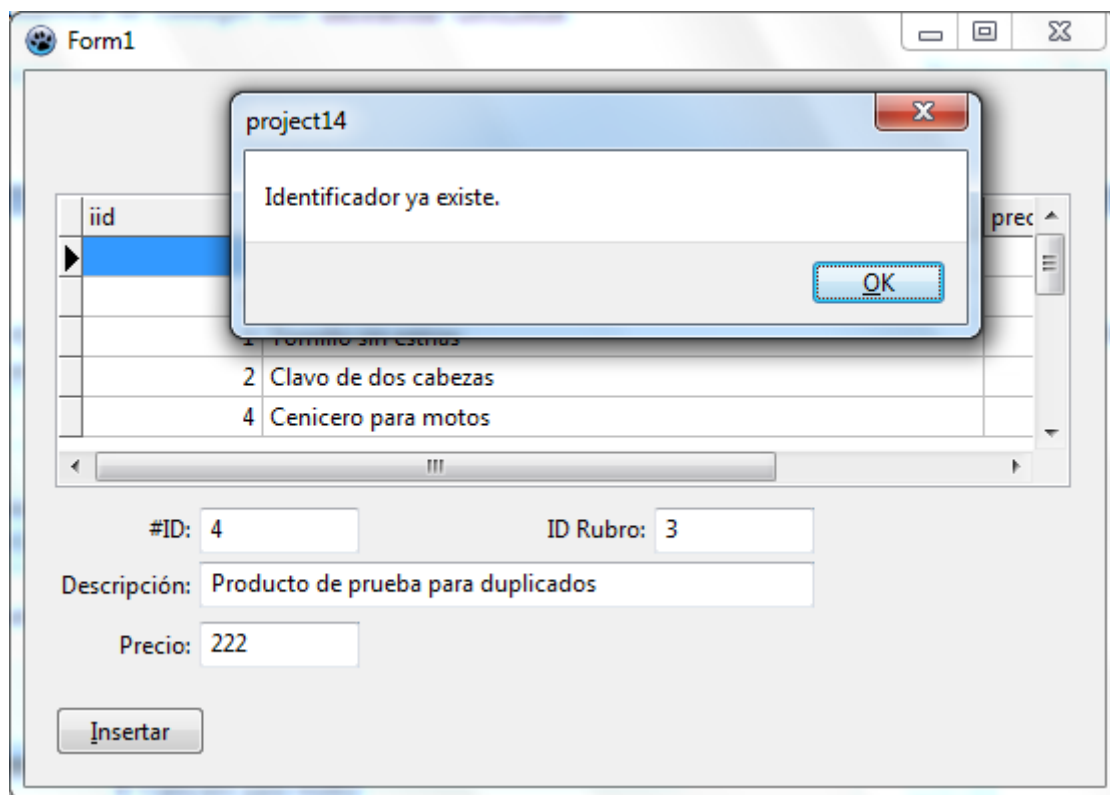


Figura 95: evitando el ingreso de un ID existente.

7. Ejercicio libre

- Este ejercicio libre compara una aplicación basada en Base de Datos con otra que utiliza un archivo que es leído a la memoria.
- Considera la empresa QuiereLista S.A., que desea ver implementada una lista telefónica.
- Una empresa llamada SoloLazarus desarrollará una aplicación que hace uso de una list-box de Lazarus para almacenar los datos que serán grabados en disco por medio del componente TXMLPropStorage. Agregar los Listbox.Items en las Session Properties del Form. Si lo consideras útil, usa el carácter especial # para separar el nombre del número de teléfono guardando la información en la misma línea de una única ListBox.
- Otra empresa llamada LazGres va a desarrollar una aplicación rival haciendo uso de PostgreSQL y Lazarus.
- Ambas empresas van a utilizar un listado con solo dos columnas: nombre y número de teléfono: lista_telef(nombre, número_telefono)
- Para cada una de las aplicaciones rivales, crea una aplicación gráfica que permita.
 - Ingresar una nueva entrada
 - Borrar una determinada entrada
 - Listar toda la lista telefónica
 - Buscar por medio de nombre y número de teléfono.

- Debe ser posible hacer consultas de fragmentos de datos, por ejemplo: listar todos los nombres "Aquino" registrados en la base de datos.
- Empieza por proyectar el interfaz gráfico que debe, en la medida de lo posible, ser lo más común posible a ambas aplicaciones.
- Compara ambas aplicaciones y caracteriza la posibilidad de crecimiento de cada una de las bases de datos, incluyendo para el caso de múltiples accesos en simultáneo desde varias computadoras distintas.

8. Mantenimiento práctico enlazado por controles visuales

Ahora que ya hemos visto las principales formas de acceder a los datos de una tabla en una base de datos, veamos un ejemplo más práctico, usando para ello solamente controles visuales.

- Crea una nueva aplicación semejante a la que creamos en el [ítem 3](#) de este tópico. Es decir, un formulario con los cinco componentes para acceso por medio de un DBGrid. Solo que esta vez configura las columnas del la grilla en forma adecuada. Con esto tendrás algo semejante a lo mostrado en la **Figura 91**.
- Verifica que la configuración de los componentes esté correcta y que los datos se cargan en la grilla.
- Por debajo del DBGrid, agrega las cuatro etiquetas (TLabel), pero esta vez agregaremos cuatro *cuadros de edición de datos vinculados (TDBEdit)*. Estos últimos se encuentran en la pestaña Data Controls de la paleta de componentes.

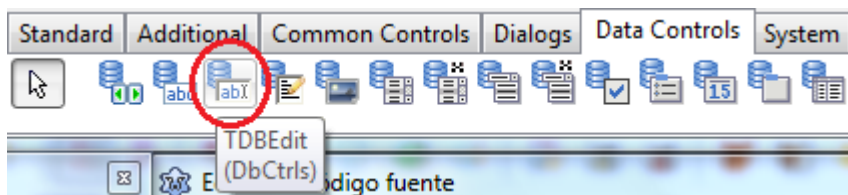


Figura 96: el componente TDBEdit ubicado en la pestaña Data Controls.

- Organiza cada etiqueta con su cuadro de texto y cambia las propiedades Caption de cada TLabel por "#ID:", "ID Rubro:", "Descripción:" y "Precio".
- En cada uno de los TDBEdit, cambia la propiedad **DataSource** por DataSource1 y la propiedad **Fields** por el campo correspondiente a la etiqueta que lo acompaña. Si todo está bien, esto nos permitirá ver el contenido de los campos, aún en tiempo de diseño!

Otra posibilidad es utilizar el evento *OnCreate* del formulario para establecer estos valores, descartando así la visibilidad de los campos en tiempo de diseño. Podríamos utilizar un código como el que sigue:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
    DBEdit1.DataSource:=DataSource1;
    DBEdit2.DataSource:=DataSource1;
    DBEdit3.DataSource:=DataSource1;
    DBEdit4.DataSource:=DataSource1;

    DBEdit1.DataField:='IID';
    DBEdit2.DataField:='IRubro';
```

```

DBEdit3.DataField:='DescProd';
DBEdit4.DataField:='PrecUnit';
end;

```

- Guarda, ejecuta y navega por la grilla para comprobar que, al cambiar de fila, los controles de edición se van actualizando automáticamente. El resultado debería ser semejante al mostrado en la **Figura 97**.

# ID	ID Rubro	Nombre del producto	Precio
1	1	Tornillo sin estrias	2.640
2	1	Clavo de dos cabezas	3.080
4	3	Cenicero para motos	1.540
5	3	Timbre para panteón	2.200
7	3	Fósforos sin cabeza	660
8	3	Borrador de carbón	770

#ID: 4 ID Rubro: 3

Descripción: Cenicero para motos

Precio: 1.540

Figura 97: controles individuales de edición, relacionados a los datos.

- Bien, ahora que ya tenemos los controles de edición individuales conectados a los campos de la tabla, podemos desactivar la posibilidad de edición en la grilla, para esto utilizamos la propiedad **ReadOnly** del DBGrid y la ponemos como *True*.
- Sin embargo, aun nos falta un poco más de movilidad y prestaciones de mantenimiento de datos. Para esto vamos a agregar algunos controles más a nuestro formulario.
- Para **implementar una barra de navegación**, tome un componente **TPanel** en la pestaña Standard y póngalo en el form. Defina la propiedad **Align** como *alBottom*. Elimina el contenido de la propiedad Caption. Las propiedades BevelInner y BevelOuter son usadas para definir los bordes para el panel. Establezca, por ejemplo, como *bvLowered* y *bvRaised* respectivamente.
- Ahora selecciona un **TDBNavigator** en la pestaña Data Controls y póngalo en el form, sobre el panel. En la propiedad **DataSource** del navegador selecciona el DataSource1. Cambia **Height** para 35 y **Width** para 320.
- Para terminar el diseño, ponga un **TSpeedButton** de la pestaña Additional sobre el panel. Defina **Name** como btnSalir, **Height** y **Width** como 35, alinea con la DBNavigator y, para completar, elija un ícono apropiado para el botón usando la propiedad **Glyph**. En el evento OnClick de este botón agrega el código:

```
Close;
```


- Guarda, ejecuta y navega utilizando la barra de navegación implementada para comprobar que los botones Primero, Anterior, Siguiente y Ultimo funcionan

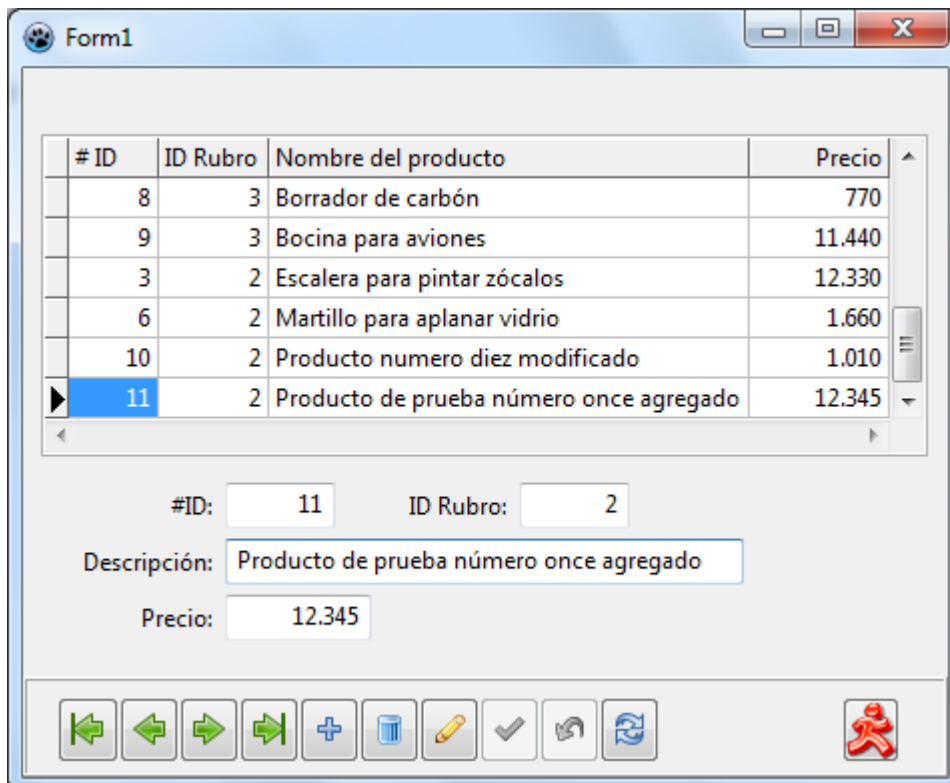
perfectamente reposicionando el puntero de registros en la grilla y los controles de edición siguen actualizándose automáticamente. El resultado ahora debería ser algo semejante al mostrado en la **Figura 98**.

Sin embargo, podrá darse cuenta de que no le permitirá ingresar ni editar datos permanentemente, ya que estos se quedarán en el cache. Es necesario enviar los datos a la BD y finalizar la transacción.

Esto será hecho en el evento **AfterPost** de la SQLQuery. Este evento ocurre luego que un evento sea grabado en la base de datos. Selecciona la SQLQuery1 y en la pestaña **Eventos** del *Inspector de Objetos* edite el evento AfterPost respectivo y agrega el siguiente código:

```
procedure TForm1.SQLQuery1AfterPost(DataSet: TDataSet);
var
    posicion : TBookMark;
begin
    try
        posicion := SQLQuery1.GetBookmark;
        SQLQuery1.ApplyUpdates;
        if SQLTransaction1.Active then
            begin
                SQLTransaction1.CommitRetaining;
                SQLQuery1.GotoBookmark(posicion);
            end;
        except
            SQLTransaction1.Rollback;
        end;
    end;
```

- Guarda, ejecuta y comprueba que ahora, los cambios efectuados y los registros ingresados quedan grabados luego que se oprima el botón  Guardar (post) correspondiente.



# ID	ID Rubro	Nombre del producto	Precio
8	3	Borrador de carbón	770
9	3	Bocina para aviones	11.440
3	2	Escalera para pintar zócalos	12.330
6	2	Martillo para aplanar vidrio	1.660
10	2	Producto numero diez modificado	1.010
11	2	Producto de prueba número once agregado	12.345

#ID: 11 ID Rubro: 2

Descripción: Producto de prueba número once agregado

Precio: 12.345

Figura 98: barra de navegación funcional implementada.

- **Solucionando el problema de los Hints de TDBNavigator.** Los **Hints**, también conocidos como “tooltips”, son pequeñas etiquetas aclarativas que aparecen cerca de los controles, cuando el usuario pasa el puntero del mouse sobre los mismos en tiempo de ejecución. Los Hints de los botones del componente TDBNavigator están originalmente en Inglés, pero a veces es mejor modificarlos para utilizarlos en otro idioma. La posibilidad de cambiar estos Hints está, en principio, dada por la propiedad **Hints** del control que es una TStrings y puede ser editada con la finalidad de ingresar una línea descriptiva para cada uno de los botones disponibles. Para ello podemos usar el Inspector de Objetos, clicar en la propiedad y usar el botón de edición para abrir el *Editor de Strings* asociado.

Dependiendo de la versión de Lazarus que estés utilizando, la propiedad **Hints** del TDBNavigator ya vendrá completada con las opciones correspondientes al idioma en uso. Por ejemplo, para la versión en español que hemos utilizado vienen cargadas las opciones mostradas en la figura abajo.

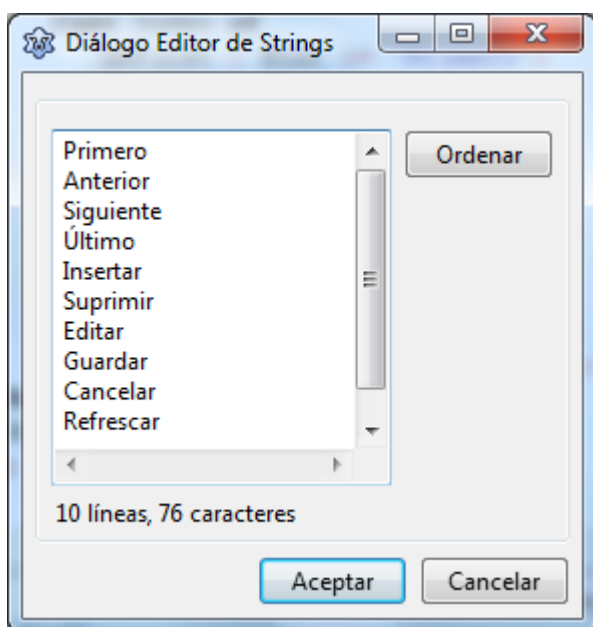


Figura 99: Hints previamente configurados para el TDBNavigator en español.

Aun así, las etiquetas de los botones luego de compilar pueden seguir saliendo ien inglés! Aunque no puedo explicarles el motivo, según mi experiencia, cualquier cambio que ingresemos a este conjunto normalmente soluciona el problema. Por ejemplo, cambiar *Primero* por *Primer* o *Último* por *Ultimo*, y luego compilar, debería funcionar en tiempo de ejecución.

Si por algún motivo lo anterior no funciona o usted **realmente** quiere que los títulos predeterminados sean los que se utilicen como Hints de sus botones, hay otra buena alternativa que es asignarlos vía código. Para esto deberemos cumplir con lo que sigue:

- En el editor de código, antes de declarar el formulario, declaramos una clase sinónima del tipo TDBNavigator, ingresando el siguiente código:

```
type
  TDBNewNavigator = class (TDBNavigator);
```

- Nos aseguramos que la declaración **uses** contiene la librería **Buttons**:

```
uses
  ..... , Buttons;
```

- Finalmente, a la hora de crear el Form, asignamos a cada uno de los botones del DBNavigator el *Hint* correspondiente. Para esto, en el evento **OnCreate** del formulario, ingresamos el siguiente código:

```

procedure TForm1.FormCreate(Sender: TObject);
var
    B: TNavigateBtn;
begin
    for B := Low(TNavigateBtn) to High(TNavigateBtn) do
        with TDBNewNavigator(DBNavigator1).Buttons[B] do
            begin
                Case Index of
                    nbFirst : Hint := 'Primero';
                    nbPrior : Hint := 'Anterior';
                    nbNext  : Hint := 'Siguiente';
                    nbLast  : Hint := 'Último';
                    nbInsert : Hint := 'Insertar';
                    nbDelete : Hint := 'Suprimir';
                    nbEdit   : Hint := 'Editar';
                    nbPost   : Hint := 'Guardar';
                    nbCancel : Hint := 'Cancelar';
                    nbRefresh: Hint := 'Refrescar';

                End;
                ShowHint := True;
            end;
            ... //cualquier otro código necesario en OnCreate
        end;

```

- Observe que cada uno de los botones es un elemento del array Buttons del DBNavigator.
- Guarda, compila y ejecuta con [F9].

Esto es todo, con cualquiera de las técnicas presentadas, los Hints de sus botones de navegación deberían lucir más o menos como en la figura abajo.

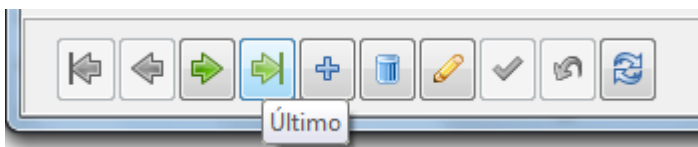


Figura 100: TDBNavigator con los *Hints* traducidos al idioma español.

- Por consistencia de diseño, si aun no lo hizo, no olvide agregar un valor a la propiedad **Hint** del btnSalir. "Salir", por ejemplo.
- **Usando el TDBLookupCombobox para ver y editar claves foráneas.**
Por ahora, en nuestro formulario de mantenimiento, hemos tenido que ver e ingresar la clave foránea (FK) de rubros (producto.irubro) sin estar seguros realmente de lo que representa dicha clave y ni siquiera si ella existe. A parte de todas las validaciones, restricciones y reglas que se pueden hacer al nivel de la base de datos, es costumbre presentar al usuario la definición de FKs por medio de un control del tipo ComboBox dónde se muestran todas las descripciones de la entidad hija y se permite elegir una, cuyo código será grabado en el campo FK correspondiente de la tabla madre.

En el t3pico [II.9](#) vimos como usar un componente **TComboBox** para definir opciones manualmente, obtener un valor y realizar acciones de acuerdo al valor elegido. Sin embargo, dicho componente no est3a preparado para interactuar directamente con las tablas de una base de datos. A partir de este punto vamos a ver c3mo utilizar un control del tipo **TDBLookupCombobox** que es el componente recomendado para efectuar las acciones mencionadas.

- o Antes que nada, debemos tener en mente que queremos mostrar como opciones del combobox los datos de otra tabla que no es aquella a la que accedemos actualmente por medio de nuestro par `SQLQuery1` y `DataSource1`, as3 que debemos obtener otro par de estos que nos apunte a la tabla de rubros.

- Previamente se ha creado en la base de datos la tabla de **rubros** con la siguiente instrucci3n.

```
CREATE TABLE rubro
(
    iid integer NOT NULL,
    describro character(25),
    CONSTRAINT id_rubro_pk PRIMARY KEY (iid)
)
WITH (
    OIDS=FALSE
);
```

- Fueron ingresados 3 registros con los siguientes valores:

<u>IID</u>	<u>DESCRUBRO</u>
1	FERRETER3A
2	HERRAMIENTAS
3	ACCESORIOS

- o Desde la pesta3a SQL Db de la paleta de componentes, inserte un **TSQLQuery** al formulario y desde la pesta3a Data Access obtenga otro **TDataSource**. A seguir configura ambos componentes como indicado abajo.

- En el `SQLQuery2` reci3n creado cambiamos o confirmamos las siguientes propiedades:

Database: PQConnection1
DataSource: (dejar en blanco)
Name: sqRubro
SQL: select * from rubro;
Transaction: SQLTransaction1
Active: True

- En el `DataSource2` reci3n creado cambiamos las siguientes propiedades:

DataSet: sqRubro
Name: dsRubro

- o Eliminamos el `DBEdit2` que estaba enlazado al campo `iRubro` y, en su lugar, agregamos y posicionamos un **TDBLookupCombobox** desde la pesta3a Data Controls. A continuaci3n configuramos las propiedades de dicho componente seg3n lo indicado abajo.

- Para listar los datos en el ComboBox tenemos que completar las siguientes propiedades:

ListSource: dsRubro (DataSource de origen del combo)
ListField: describro (Campo que se va a mostrar al usuario)
KeyField: iid (Campo del que se toma el valor – tabla hija)
Style: csDropDownList (no permite modificar el texto mostrado)

- Para almacenar el valor obtenido, en el campo FK de la tabla madre, hay que llenar las siguientes propiedades:

DataSource: DataSource1 (DataSource de destino, tabla madre)
DataField: irubro (Campo FK que va a almacenar el valor)

- Guarda, ejecuta y comprueba que ahora, el combobox nos mostrará la descripción del código FK de cada registro de la tabla de Productos, como podemos ver en la Figura 101.

#ID	ID Rubro	Nombre del producto	Precio
8	3	Borrador de carbón	770,00
9	3	Bocina para aviones	11.440,00
3	2	Escalera para pintar zócalos	12.330,00
6	2	Martillo para aplanar vidrio	1.660,00
10	2	Producto número diez modificado	1.010,00
11	2	Producto de prueba número once agregado	12.345,00

#ID: 10 Rubro: HERRAMIENTAS
 Descripción: Producto número diez modificado
 Precio: 1.010,00

Figura 101: TDBLookupCombobox representando la clave foránea de rubro.

- Algunas veces puede suceder que al abrir el formulario, el combobox no traiga la descripción correspondiente al primer registro mostrado. Sin embargo, al mover el puntero por cualquier medio ya todo pasa a funcionar normalmente. Podemos solucionar esto fácilmente, forzando el puntero de la tabla madre a reposicionarse en el primer registro. Para esto pondríamos en el método **OnCreate** del Form algo parecido al siguiente código:

```

SQLQuery1.Open;
SQLQuery1.First;

```

Al guardar, compilar y probar ya nos debería mostrar luego de entrada la descripción del código de rubro del primer registro de la tabla. Si deseamos iniciar el formulario en el último registro podemos cambiar el `SQLQuery1.First` por `SQLQuery1.Last` y funcionará igual.

- En la **Figura 102** vemos como el cuadro de lista de nuestro combobox muestra todas las descripciones correspondientes a los rubros existentes, resaltando la opción que define la FK actual (fíjese en el grid). Si cambiamos la selección actual por otra, podemos ver en la grilla que el valor de la FK en la columna ID Rubro también cambia (**Figura 103**), indicándonos que el **TDBLookupCombobox** realizó el trabajo que le hemos asignado.

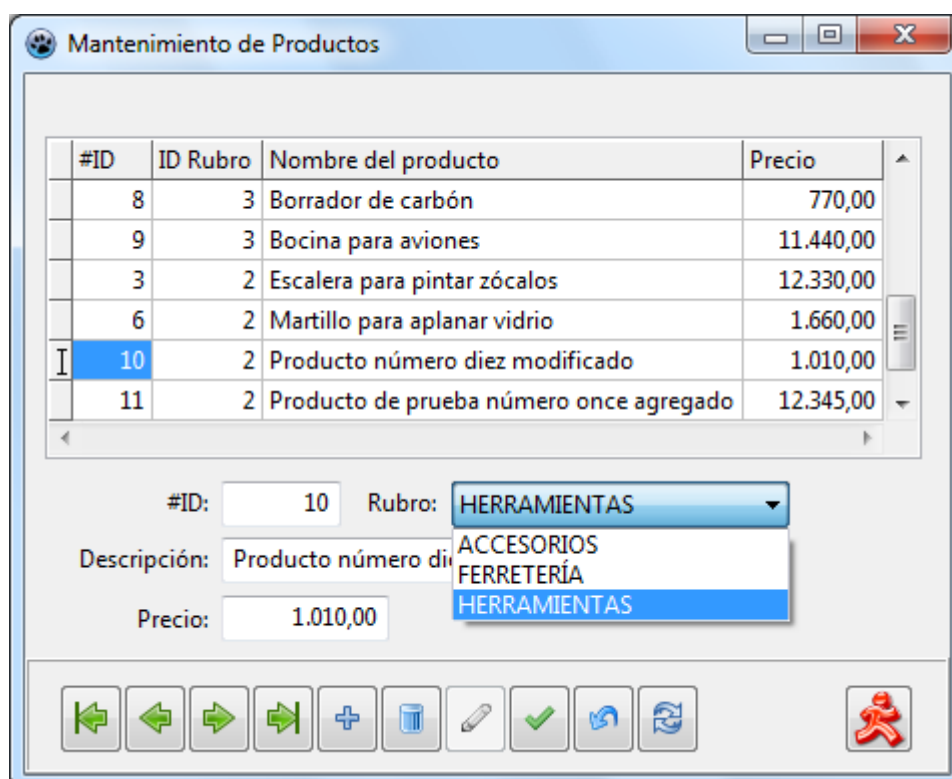


Figura 102: La lista de opciones del combobox de rubros desplegada.

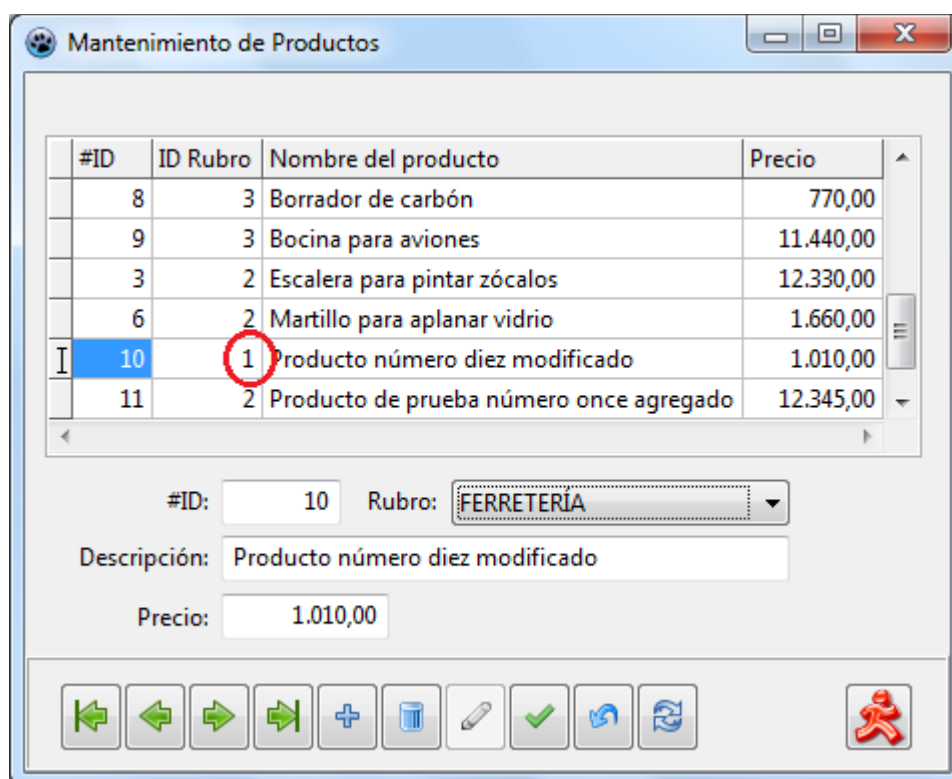


Figura 103: La ID de Rubros modificada al cambiar la opción en el combobox.

V – ALGUNOS COMANDOS SQL (Micro resumen)

1. Manejo de tablas

- CREATE TABLE "nombre_tabla "
("columna_1" "tipo_de_dato_para_columna_1",
"columna_2" "tipo_de_dato_para_columna_2",
...)
 - *tipos → varchar e int*
 - *obs: utilizar nombres de columnas en minúsculas*
- DROP TABLE nombre_tabla

2. Manejo de datos

- INSERT INTO Personas
VALUES('Hussein', 'Saddam', 'White House')
- INSERT INTO nombre_tabla
VALUES (valor_1, valor_2,...)
- INSERT INTO nombre_tabla (columna1, columna2,...)
VALUES (valor_1, valor_2,...)
- UPDATE nombre_tabla
SET nombre_columna_1 = nuevo_valor_1, nombre_columna_2 = nuevo_valor_2
WHERE nombre_columna = algún_valor
- DELETE FROM nombre_tabla
WHERE nombre_columna = algún_valor

3. Consultas de datos (queries)

- SELECT nombre_columna(s)
FROM nombre_tabla
WHERE columna *operador* valor
AND columna *operador* valor
OR columna *operador* valor
AND (... OR ...)
...
 - *Operadores =, <>, >, <, >=, <=, between, like, like → %wildcard%wildcard%*
- SELECT nombre_columna(s)
FROM nombre_tabla
WHERE nombre_columna IN (valor1, valor2, ...)
- SELECT nombre_columna(s)
FROM nombre_tabla
ORDER BY fila_1, fila_2 DESC, fila_3 ASC, ...

- SELECT columna_1, ..., SUM(nombre_columna_grupo)
FROM nombre_tabla
GROUP BY nombre_columna_grupo

■ *Agregados:* avg, count, max, min, sum

- SELECT columna_1, ..., SUM(nombre_columna_grupo)
FROM nombre_tabla
GROUP BY nombre_columna_grupo
HAVING SUM(nombre_columna_grupo) *condición* valor

VI – FAQ BÁSICA DE LAZARUS SOBRE TEMAS AFINES

1. ¿Por qué Lazarus?

Lazarus es un programa en rápida evolución, liviano y que funciona en varios sistemas operativos, como sean Linux, Win32/64, Mac OS x, OS/2, 68K y más.

Como pasa con la mayoría de las aplicaciones multi plataformas, es más rápido bajo Linux que bajo Windows.

Deriva del Pascal que tiene más especificaciones run-time que el propio C y por eso frecuentemente el propio Pascal señala errores que de otra forma habría ocasionado malos funcionamientos difíciles de explicar.

Es un software libre, utiliza licencia LGPL, ver: <http://www.freepascal.org/faq.var>

El proyecto Lazarus se inició en Febrero de 1999, sigue en pleno desarrollo y hoy en día ya es considerado una herramienta bastante estable cuya comunidad mundial es cada vez mayor.

A diferencia de Java, que se esfuerza por ser “una sola escritura y ejecutar en cualquier lugar”, Lazarus y Free Pascal se esfuerza por lograr una sola escritura y compilar en cualquier lugar. Desde que exactamente el mismo compilador está disponible en todas las plataformas de arriba significa que no es necesario hacer ninguna recodificación para producir productos idénticos para las diferentes plataformas.

Además, permite conexión nativa a múltiples bases de datos.

Lazarus es una aplicación normal de FreePascal (FPC) que es una implementación abierta de un lenguaje muy aproximado al Object Pascal propuesto por el entonces Borland Delphi (ahora de Embarcadero).

La compatibilidad de Free Pascal con Delphi incluye no sólo el soporte para el mismo lenguaje de programación Object Pascal que utiliza Delphi, sino que también para muchas de las mismas bibliotecas de poderosas rutinas y clases por las que Delphi es conocido. Los diseñadores de Lazarus indican, que cualquier código de Delphi, con los componentes estándar debería funcionar sin problemas al cargarlos directamente en Lazarus.

Por lo anterior, es válido considerar que prácticamente todo el soporte encontrado para Delphi en la web puede ser aplicado casi en su totalidad a los proyectos hechos en Lazarus.

2. Lazarus y POE

[Lazarus](#) es un [IDE](#) y un [RAD](#) y permite realizar programación de varios tipos, incluyendo Programación Orientada a Eventos ([POE](#)).

3. Lazarus, debugger, inestabilidades y errores raros

Como ya vimos, Lazarus viene con GDB (GNU Debugger) como su depurador predeterminado. Esta herramienta está en constante estado de desarrollo y algunas veces pueden aparecer problemas de estabilidad, principalmente sobre Windows.

Para contornar dichos problemas:

- Usar siempre la última versión de Lazarus.
- Probar el programa fuera del IDE, ejecutándolo por medio del S.O.
- Desconectar el depurador del Lazarus.
- Bajo Linux y pretendiendo obtener información adicional relacionado con algún tipo de error con falta de información de depuración (por ejemplo: errores del FPC)
 - Ejecutar el Lazarus dentro de una shell de texto.
 - Ejecutar la aplicación creada dentro de una shell.

4. Lazarus, PostgreSQL y errores relacionados a libpq.xxx

- **Windows:**

Para que el Lazarus se conecte al servidor de PGSQL es necesario que:

- En la máquina del Lazarus debe estar instalado el PGSQL (cliente o servidor).
- Las DLLs del [PGSQL\bin] deben estar en el sendero predeterminado (path) del Sistema Operativo.
 - Para modificar el sendero predeterminado, ir al Panel de Control, Sistema, Configuración avanzada del Sistema, Opciones Avanzadas, Variables de entorno y en Variables del Sistema seleccionar la variable PATH (que siempre existirá). Clic en [Modificar] y agregar al final del contenido existente un punto y coma (;) seguido por el sendero completo hacia la carpeta donde se encuentran las DLLs mencionadas (por ejemplo: e:\PostgreSQL\9.2\bin\).
 - Una solución alternativa es copiar los archivos **libpq.dll**, **libeay32.dll**, **libintl-8.dll**, **ssleay32.dll** desde la carpeta [Bin] de la instalación del PostgreSQL, hacia la carpeta c:\windows\system32.

- **Linux:**

- Crear un *soft link* del liqpq.so para apuntar al fichero liqpq.so.* con la extensión más avanzada.
- Agregar el *path* del liqpq.so.* en la sección libraries del archivo de configuración del FPC que generalmente es el etc/fpc.cfg.
Por ejemplo: **-FI/usr/local/pgsql/lib**

5. Error "Circular Unit Reference"

Algunas veces, si no se tiene el cuidado correspondiente, según vaya creciendo nuestra aplicación, durante la compilación puede surgir este problema común a cualquier ambiente de programación basado en Pascal.

Significa que una o más unidades se utilizan entre sí en sus partes de la interfaz.

Como el compilador tiene que traducir la parte de interfaz de una unidad antes de que cualquier otra unidad pueda utilizarlo, el compilador debe ser capaz de encontrar un orden de compilación para las partes de interfaz de las unidades.

Compruebe que todas las unidades de las cláusulas "uses" del área "interface" son realmente necesarias, y si algunas se pueden mover a la parte de la implementación de una unidad en su lugar. Es decir, sacar del "uses" del interface las unidades referentes a los formularios del usuario y crear / agregarlas a un "uses" después de la implementación de nuestras unidades.

Para ilustrar mejor, veamos la siguiente situación:

```
unit A;
interface
uses B;           (*A uses B, and B uses A*)
implementation
end.
```

```
unit B;
interface
uses A;
implementation
end.
```

Como dicho anteriormente, el problema es causado porque A y B usan uno al otro en sus secciones "interface". Así que lo correcto sería:

```
unit A;
interface
uses B;           //Orden de compilación: B.interface, A, B.implementation
implementation
end.

unit B;
interface
implementation
uses A;           //Movido a la sección de implementación
end.
```

O sea, la solución más fácil sería dejar la unitB en la cláusula uses de la sección de interfaz de unitA y pasar la unitA a la cláusula uses de la sección de implementación de la unitB.

La mejor solución, sin embargo, sería la de romper la dependencia para ambas (o al menos una) unidades.

Puede romper la dependencia, ya sea:

- Moviendo todas las llamadas de la unitA a unitB en unitB

- Creando una tercera unidad, de código compartido, utilizando ambas unidades A y B.

6. Cadenas Dinámicas (' comillas dentro de "comillas"')

- En Lazarus, las cadenas de caracteres (strings) son delimitadas por apóstrofes (también conocidos como 'comillas simples').
- Es perfectamente posible poner "comillas dobles" dentro de 'apóstrofes'.
- Un par de apóstrofes dentro de una cadena Lazarus equivale a una cadena con **un** apóstrofe en ella.
- Ejemplos:

- Ejecutando el código: `s := ' 1 2 3 ' ' 4 5 6 ' ' 7 8 9 ' ' ' ;`
s quedará con la *string*: `1 2 3 ' 4 5 6 ' 7 8 9 '`

- Considere la siguiente consulta SQL:
SELECT nombre FROM paciente LIKE 'T%'

El código necesario para realizar esta consulta es:

s := 'SELECT nombre FROM paciente LIKE "T%";

Para utilizar la información de una EditBox:

s := 'SELECT nombre FROM paciente LIKE "' + EditBox.text + '"';

Donde la editbox tendría, por ejemplo, el valor T%.

Fíjese también que el espacio luego del LIKE es esencial.

7. Consultas que no funcionan y Esquemas y/o Relaciones (Tablas) a las que no se consigue acceder

- Utilizar tablas con los nombres solo en minúsculas.
- Poner el nombre de la tabla entre comillas. Por ejemplo: "Tabla".
- Utilizar el esquema público o "Esquema"."Tabla"

8. Passwords (contraseñas)

Para el ingreso de contraseñas, se puede utilizar la propiedad Edit.PasswordChar. Si se cambia el valor por defecto (#0) a, por ejemplo, * (un asterisco) en el "TEdit" se visualizarán asteriscos en lugar de los caracteres que se escriban en el mismo.

9. ¿Cómo crear informes con Lazarus?

Aunque es posible utilizar otras herramientas de terceros como el [Fortes Report](#), Lazarus trae la posibilidad de instalar un [paquete](#) llamado [LazReport](#) que es un grupo de componentes que agregan a las aplicaciones la capacidad de generar reportes, usa un diseñador visual para crear reportes basados en bandas e incluye un mecanismo de reportes con previsualizador que además incluye un intérprete para ejecutar guiones

(scripts) de usuario. El diseñador de reportes puede ser invocado en tiempo de ejecución.

10. ¿Qué es un paquete Lazarus?

Es un conjunto de unidades y componentes, que contiene información sobre cómo compilarlos y utilizarlos en proyectos, otros paquetes o en el IDE. Al contrario que en Delphi los paquetes no están limitados a librerías y pueden ser independientes de la plataforma.

Actualmente el compilador Free Pascal únicamente soporta paquetes estáticos. Por tanto hay que compilar y relanzar el IDE cada vez que se instala o elimina un paquete. El IDE automáticamente compila los paquetes si alguno de sus archivos es modificado.

Un paquete Lazarus se identifica y distingue por su nombre y su versión. Los paquetes son ideales para compartir código entre proyectos.

Para más informaciones sobre paquetes en Lazarus, consultar la wiki correspondiente: http://wiki.freepascal.org/Lazarus_Packages/es

11. ¿Cómo reducir el tamaño de los ejecutables?

De forma predeterminada, Lazarus crea un ejecutable (los archivos .EXE) incluyendo información de depuración que lo hace extremadamente grande, alcanzando los 14 MB o más. Se puede quitar esta información de depuración con facilidad y hacer que sus archivos EXE sean bien más pequeños.

Para reducir el tamaño de un ejecutable, se puede utilizar el comando "**strip**", que se encuentra en forma predeterminada en el path del S.O. Linux. En Windows, se puede usar el ejecutable desde una ventana de símbolo del sistema. Típicamente el comando a utilizar sería (adaptar según la carpeta de instalación y las versiones específicas instaladas):

```
c:\lazarus\fpc\2.6.4\bin\i386-win32\strip.exe nombre_del_exe_a_reducir
```

En las últimas versiones de Lazarus hay formas de ejecutar este comando desde el propio IDE, inclusive en forma automática durante la compilación. Debido al alcance de este documento, no entraremos en mayores detalles sobre este tema, pero si desea investigar más puede empezar por las FAQs de la wiki de Lazarus, en el siguiente enlace: [¿Por qué los binarios generados son tan grandes?](#)

12. Archivos que pueden ser borrados

Lazarus utiliza varios archivos temporales, algunos de los cuales pueden ser eliminados sin problemas. Para tal efecto, utilizar la opción del menú Archivo -> Limpiar directorio... Algunos de los archivos que pueden ser eliminados son: *.compiled, *.o, *.ppu.

VII – LISTADO DE TECLAS Y FUNCIONES

Lazarus dispone de un gran número de teclas de atajos para funciones específicas del IDE. Abajo se lista algunas de las más utilizadas en las tareas normales de programación:

- CTRL + Espacio – Completar código
- CTRL + SHIFT + Espacio – Parámetros de la llamada al procedimiento/función
- CTRL + J – Plantillas de código (Code Templates)
- CTRL + SHIFT + ARRIBA / CTRL + SHIFT + ABAJO – declaración / implementación
- CTRL + Clic - Declaración / definición (de cualquier variable, etc.)
- CTRL + H – Lleva al punto de edición anterior
- CTRL + SHIFT + H – Lleva al punto de edición siguiente
- F12 - Cambia entre el formulario y el editor de código
- F11 - Activa el Inspector de Objetos
- CTRL + SHIFT + C – Completa clase / implementación de clase
- CTRL + K + I – Indenta hacia la derecha varias filas de código seleccionadas
- CTRL + K + U – Indenta hacia la izquierda varias filas de código seleccionadas
- CTRL + Y – Borra la línea actual en el Editor de Código
- CTRL + Z – Deshacer la última acción realizada
- F4 – Depurador (Debugger) -> Ejecutar hasta el cursor
- F7 – Depurador (Debugger) -> Paso a paso por Instrucciones (Step into)
- F8 – Depurador (Debugger) -> Paso a paso por Funciones (Step Over)
- F9 – Ejecutar