30

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

ESCUELA NACIONAL DE ESTUDIOS PROFESIONALES

"A C A T L A N"

CREACION DE UNA LIBRERIA DE LIGADO DINAMICO (DLL) EN DELPHI 3

T E S I S

QUE PARA OBTENER EL TITULO DE:

LICENCIADO EN MATEMATICAS APLICADAS Y COMPUTACION

P R E S E N T A :

RAFAEL MUÑOZ GOMEZ

ASESORA ACT. BEATRIZ ELENA ESCOBEDO DE LA PEÑA.

ACATLAN, EDO. DE MEXICO. JULIO DE 2001.

Índice

Introducción

El presente proyecto nació como respuesta a lo que, desde mi punto de vista, es un problema cuando los egresados de la Licenciatura en Matemáticas Aplicadas y Computación salen a la vida productiva y pretenden desarrollarse en el área de sistemas en alguna compañía de vanguardia, cuyas herramientas de desarrollo son muy modernas pues son las que mejor resuelven los problemas de clientes de software. Hoy día las compañías que producen herramientas de desarrollo o lenguajes de programación modernos y poderosos, también llamados RAD's (Rapid Aplication Development), están abarcando el mercado de los programadores haciendo paquetes que con un mínimo de esfuerzo sean capaces de desarrollar sistemas para el cliente, muy poderosos. Este es el caso de Borland, ahora Inprise, y uno de sus productos, en especial en el que está desarrollado este proyecto: Delphi.

La problemática a la que muchos egresados se enfrentan, no es que sean malos programadores ni mucho menos profesionales, sin embargo, cuando se está en competencia directa contra gente que conoce herramientas modernas, se tiene una fuerte desventaja. En el caso de Delphi, cuando se intenta hacer el cambio a esta herramienta, la migración del programador puede ser muy fácil si este conoce el lenguaje de programación Pascal, de hecho Delphi es la versión visual y orientada a eventos de este lenguaje ya clásico entre los programadores. Por lo tanto, un buen programador no tendrá problemas al cambiar de plataforma, pues sólo se debe acostumbrar a algunas diferencias en cuanto al flujo lógico del programa y a manejar los componentes de Delphi, si se consigue esto, el potencial de programación y de competencia comercial, puede ser mucho mas grande de lo que se puede imaginar.

El objetivo buscado ahora es compartir esta experiencia de manera que los siguientes egresados interesados en desarrollarse profesionalmente en el área de sistemas, tengan algún tipo de apoyo en el tema referido, es decir, la intención del trabajo es mostrar cómo se puede generar una aplicación común en Delphi, como se puede construir una librería de ligado dinámico (Dll) y cómo se puede utilizar este *dll* en algún otro programa. No se pretende enseñar a programar, por el contrario, mostrarle a los programadores que utilizan Pascal como se pueden cambiar de herramienta de desarrollo de una manera sencilla y sin tener que cambiar todo lo que saben de programación. Este proyecto presenta algunos tips para iniciar la exploración de un lenguaje nuevo, que es altamente cotizado en el mercado actual y que puede resultar muy interesante para aquellos con expectativas de crecimiento en base al autoaprendizaje. También muestra como se puede generar un *dll* que puede ser una herramienta muy útil pues genera opciones de crecimiento de una manera óptima y que no todo el mundo sabe como hacer.

El proceso para ser un experto en el uso de Delphi, no es mas que la práctica, cuando ya se es buen programador, sólo resta familiarizarse al máximo con alguna herramienta

poderosa de desarrollo, Delphi puede ser el lenguaje indicado pues tiene muchas opciones y variantes para desarrollar aplicaciones, además, existen muchos lugares en Internet donde se pueden encontrar nuevos componentes y nuevas opciones de desarrollo, así que las opciones se limitan a la imaginación de cada programador, pues la estructura del lenguaje permite generar componentes personalizados, nuevas clases a partir de las que ya existen, hay compañías que se dedican a hacer componentes mejorados y todo esto Delphi lo permite sin problemas. Además existen ya 4 versiones del lenguaje y se debe estar trabajando ya en la quinta[1], por lo que no debe haber preocupación en cuanto a la obsolescencia.

Otra ventaja que se puede considerar en Delphi, es que los ambientes de desarrollo de otras compañías tienden a parecerse mucho cada vez mas, por lo que sería muy sencillo una vez que se conozca Delphi emigrar hacia otro tipo de lenguaje, incluso a otra compañía de software. Así que realmente se aprendería la nueva forma de desarrollar o lo que está de moda en el mercado actual, espero que el proyecto sea de utilidad para las personas que tengan este tipo de inquietudes y pueda ser un punto a favor para alguien cuando busque un lugar en este competido pero basto mercado.

Al final se pretende poner a disposición de todos los programadores unas rutinas básicas de manipulación estadística y de probabilidad que podrán ser utilizadas desde cualquier lenguaje de programación orientado a Windows que permita hacer llamadas a librerías dinámicas. Las rutinas podrán ser adecuadas, mejoradas o incluso agregar mas rutinas al *dll* siempre que así se necesite, este trabajo deberá ser muy sencillo pues la librería está lista para correr y no deberá presentar ningún problema modificarla.

Quiero dejar bien claro que este trabajo no pretende servir como referencia teórica de ninguno de los temas expuestos, es simplemente el resumen de mi experiencia para aprender y desarrollar en Delphi; a partir de teoría matemática y del propio lenguaje, se muestra como unirlas para obtener un programa que pueda resolver algún problema real con la ayuda de la computadora.

Rafael Muñoz Gómez

[1] La versión de Delphi en que están desarrollados todas las aplicaciones del proyecto es la 3

Capítulo I – Conceptos de Delphi

Durante el desarrollo del presente capítulo, se observará la parte del proyecto que muestra el conocimiento acerca del archivo *dll* que se está desarrollando y del ambiente en el que tomará forma. Es decir, se verán todos los conceptos que tienen que ver con la programación de librerías dinámicas en Delphi 3, que es la aportación mas importante y mas costosa del desarrollo que se presenta La investigación realizada para la elaboración del presente capítulo va mas allá de lo que se puede ver en un curso de programación estándar, pues no siempre se parte de la base que el lector está familiarizado con el lenguaje de programación Pascal, lo que evita llegar a temas tan específicos como la creación de este tipo de librerías.

1.1 ¿Qué es Delphi?

El lenguaje de programación Delphi, nació como una versión beta del lenguaje Pascal para Windows, fue tal la aceptación que tuvo este RAD (Rapip Aplication Development) con el nombre de la versión beta, que sus productores (Borland International o últimamente Inprise) decidieron dejarle el mismo nombre y darle una nueva vida e imagen a un "heroico" y ancestral lenguaje de programación con ya varias versiones en su haber.

La filosofía del ambiente de desarrollo está completamente adaptada al estilo Windows, como resultan las interfaces de algunos otros RAD's como: Visual Basic de Microsoft, Power Builder de PowerSoft, etc. sin embargo la interfaz, cada vez mas depurada, es mucho mas amigable y resulta muy fácil, para los usuarios Pascal, adaptarse a su forma de codificar, pues las definiciones de funciones, variables, procedimientos, objetos, clases, etc. son idénticas a lo que se hacía antes, sólo que ahora hay *componentes, eventos, formas* y *propiedades* que se deben conocer para poder realizar cualquier tarea en esta nueva opción para los "programadores" del ambiente Windows. Para hacer dichas tareas, afortunadamente, se tienen muchas facilidades, pues la comunicación con el ambiente de producción es sumamente intuitivo y la planeación que hicieron para llegar al producto que hoy día conocemos, hacen que el desarrollar grandes y sofisticadas pantallas o sistemas sea algo realmente sencillo. Como se podrá ver adelante.

De manera general, Delphi 3 es un lenguaje visual, un compilador a 32 bits por lo que se utiliza en máquinas con sistema operativos Windows 95 o posterior, orientado a eventos y que soporta objetos. es decir, en él se desarrollan aplicaciones orientadas al ambiente Windows, sus procedimientos se ejecutan por respuestas a eventos tipo Windows (el click de botones, el Exit de componentes, el activate de formas, el close de aplicaciones, alguna combinación de teclas o HotKeys, etc) que se irán conociendo, algunos, poco a poco. Además soporta la ahora tradicional programación orientada a objetos que puede ser estudiada en libros de Pascal versión 5 o posterior.

1.1.1 ¿Qué es un componente Delphi?

Es una parte fundamental del desarrollo de aplicaciones en Delphi, pues son como los ladrillos para construir en la interfaz que el programador tiene para hacer con Delphi lo que un RAD debe poder hacer, desarrollar intuitivos y complejos programas de forma rápida y con un mínimo de esfuerzo. Estos componentes, permiten "pegar" en una *forma*[2] en que se desarrolla una aplicación "cosas" tales como botones, grids, etiquetas, diálogos, menús, campos de tipo memo, grupos, paneles, listas, combos, querys, tablas, herramientas para hacer data warehousing y un muy largo etcétera pues aunque Delphi tiene, prácticamente, un componente para cada necesidad, además existe la forma de crear componentes a la medida, lo que ha generado un amplio mercado, pues en Internet existen páginas de compañías de software que sólo se dedican a la creación de componentes o de productos auxiliares que para poder ser compatibles con Delphi, ya tienen interfaz vía componente

¿Cómo se agrupan?

En la pantalla de interfaz del usuario con Delphi se tienen diferentes "pestañas" dónde se ubican todas las herramientas visuales y de interfaz con que cuenta este lenguaje, estas pestañas presentan los distintos grupos donde se localizan todos los componentes de acuerdo a su utilidad; algunos de ellos son :

En el grupo "Standard" se ubican los componentes mas requeridos y generalmente utilizados en un mayor número de ocasiones, dado que sirven para: poner letreros en las pantallas, permitir captura de datos, generar botones para ejecutar procedimientos, listas para poder seleccionar objetos, checks para prender alguna "bandera[3]", etc. Este grupo de componentes mas que permitir hacer sistemas fáciles y amigables, son la parte de Delphi con la que se podrían hacer programas parecidos a los que se pueden hacer con Pascal antiguo, pues las herramientas mas poderosas del lenguaje son las rutinas que se encuentran en las pestañas "Decision Cube", "Data Access", "Data Controls", "Internet" y algunas otras que hacen rutinas muy complicadas sin tener que codificar gran cosa, mas bien se tiene que configurar algunas propiedades para que los componentes hagan lo que se desea.

[2] Una forma en Delphi, es una pantalla en la que se pueden pegar distintos componentes Delphi para darle funcionalidad. Es decir, una forma en ambiente de desarrollo será una pantalla en ambiente de ejecución.
[3] Una bandera es una variable, generalmente booleana (falso / verdadero), que se activa para determinar si se ejecuta algún procedimiento determinado, o se le da un formato a un texto, etc. nos da información acerca de algo muy específico, pues sólo permite un valor muy específico

En el grupo "Additional" se tienen componentes mejorados de la versión "Standard" o de alguna manera menos utilizados, en este se ubican mas bien, los componentes que son requeridos menos veces pero que pueden resultar herramientas mas completas para la creación de pantallas mas complicadas y con mucha mas presencia que sus antecesores, por poner un ejemplo, en el grupo "Standard" existe un componente Button y en "Additional" otro BitBtn, aunque funcionan igual en cuanto a los eventos que pueden "disparar[4]", la diferencia está en las propiedades, pues el segundo permite incluir una imagen en la carátula del botón, lo que hace botones mas alegres y coloridos. Este es un caso de "vanidad" únicamente, sin embargo hay otros componentes que se basan en el "Standard" pero tienen atributos mejorados, por ejemplo el CheckListBox de la pantalla "Additional" es muy parecido al CheckList de la pantalla "Standard", la diferencia radica en que el primero contiene como parte del elemento de la lista un check o marca que permite prender mas de una "bandera" durante la ejecución del programa, es decir, es una combinación inteligente de lista y check para facilitar el manejo por parte del usuario.

En el grupo "Dialogs" se tienen pantallas predefinidas y estándares en Windows, los componentes que aquí se encuentran no son sólo componentes que se ven durante la ejecución de la aplicación, si no que además tienen sus propias funciones, letreros y botones listos para ser usadas sin mayor complejidad, por ejemplo, siempre que se quiere abrir un archivo, se presenta una pantalla muy familiar para alguien que ya antes ha querido abrir uno, pues casi todos los programas que corren sobre Windows utilizan este tipo de "diálogos" predefinidos, para facilitar la operación de un sistema. Delphi tiene acceso a estos diálogos de Windows y los pone al alcance del programador para que sus aplicaciones también puedan tener lo mas elemental en cuanto a la programación visual se refiere y sin tener que hacer un desarrollo mayor del necesario.

Hay muchos componentes, todos muy útiles dependiendo de la aplicación que se desea realizar, en el presente proyecto, no se describirán todos, sin embargo se puede acudir a la ayuda de Delphi para conocer el funcionamiento de cada componente, además existen

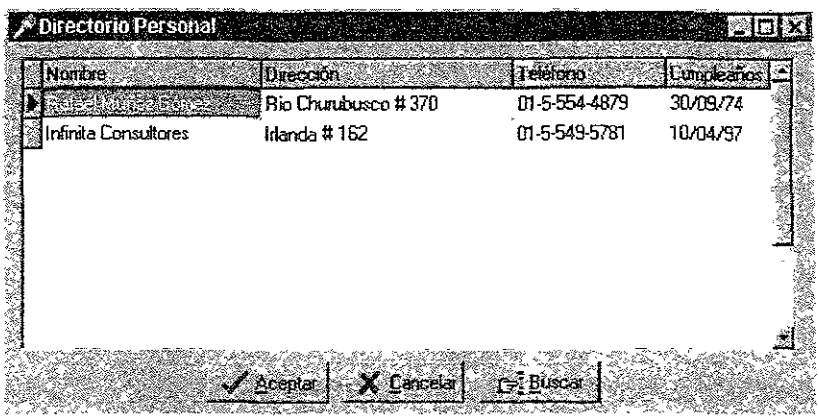
[4] Un evento se dispara desde cualquier control de Delphi, cuando el control reconoce alguna acción tomada por el usuario y reconocida por el objeto en que se llevó a cabo la acción, por ejemplo: Un botón en la pantalla puede reconocer el evento "Click" que se acciona cuando el usuario presiona el botón izquierdo del mouse sobre el botón.

muchos libros en los que se puede empezar a familiarizarse con estos componentes, su manejo, sus capacidades y sus limitaciones[5].

Tipos de Componentes y ¿cómo se utilizan?

Los componentes pueden ser, por su intervención en la interfaz con el usuario final, visibles, que son todos los componentes que el usuario que utiliza la aplicación puede ver, algunos de estos son: botones, grids, checks, combos, listas, campos memo, etc., mientras que los no visibles, pueden ser de dos tipos, los que se ven cuando son ejecutados, por ejemplo: todos los dialogos, y los que nunca se pueden ver en tiempo de ejecución, pero que realizan alguna función específica, por ejemplo: los querys, las tablas, el timer, etc.

Estos componentes pueden ser "arrastrados" desde la paleta de componentes hasta la forma en la que se está diseñando una aplicación. Dependiendo de lo que se necesite, se puede tener una combinación de componentes visibles y no visibles para llegar al objetivo planeado para la pantalla final. Por ejemplo: Supóngase que se necesita una forma con un grid para poner datos de un directorio telefónico personal, tres botones: para aceptar, para cancelar y para buscar, respectivamente, y los datos del directorio se alimentarán desde una base de datos.

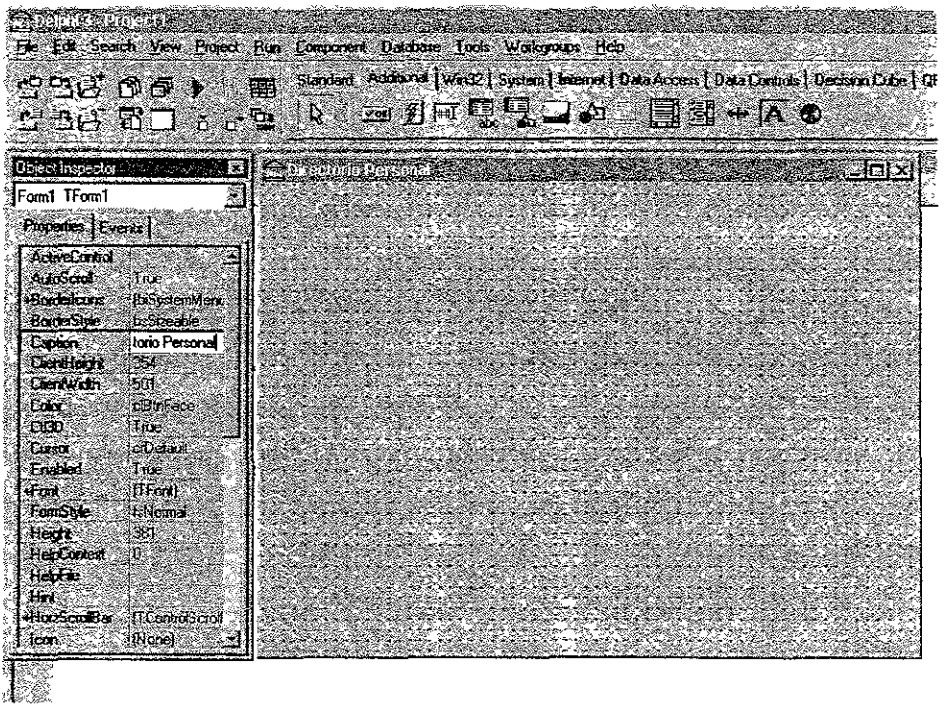


El caso anterior puede tener una presentación preliminar parecida a lo siguiente:

Para conseguir una pantalla como esta, se deben manejar algunos componentes básicos y otros que no lo son tanto, pero lo que es mas interesante de aprender en este caso, es el uso de dichos componentes, así que mientras se describe cómo se puede desarrollar una pantalla como la anterior, también se definirá que son las propiedades, eventos y métodos, tan importantes en el manejo de componentes en Delphi.

[5] En la ayuda de Delphi se puede ubicar el tema "Component Palette" donde se describe cada componente y su agrupación en las paletas de componentes. También puede consultarse en la bibliografía sugerida para conocer mas acerca de un componente en especial

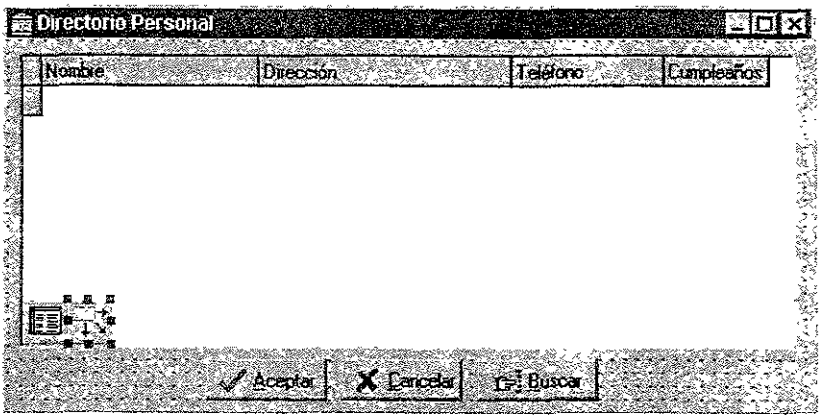
Las propiedades de los componentes, son características de cada objeto en Delphi, estas propiedades determinarán letreros, comportamientos y la forma en cómo se verá un elemento de la aplicación. Las propiedades, entonces, pueden tener algún valor o no tenerlo, afectando esto la forma en como se pueda desplegar algún componente *visible;* por ejemplo: La propiedad de "Caption" de la forma del ejemplo (directorio personal) tiene un valor = *"Directorio Personal"*, lo que hace que la esquina superior izquierda de la pantalla muestre este valor La pregunta que puede surgir en este punto, es ¿cómo cambio el valor de una propiedad en Delphi?, la respuesta es bastante simple cuando el programador se familiariza al "Object Inspector" de Delphi, pues desde éste se pueden cambiar, en tiempo de desarrollo, las propiedades de los componentes que así lo permitan. La modificación se presenta a continuación:



En este, que es el ambiente de desarrollo de Delphi, se puede observar en la parte superior, el menú para archivos, opciones, edición, etc. que Delphi usa para administrar el desarrollo de sus programas, también se puede ver la paleta de componentes, desde la cual, se pueden "arrastrar" los componentes que formarán parte de la forma en blanco que se observa en la parte inferior derecha. Y también se puede tener acceso al "Objetc Inspector" que se encuentra a la izquierda de la forma en blanco, desde aquí el programador se ubica en la propiedad *Caption* y puede modificarla a su gusto. En este caso además de la propiedad "caption" de la forma, también se modificó la propiedad de "Position" por el valor "poScreenCenter", lo que hará que la pantalla al momento de que se ejecute el programa, se presente justo al centro del monitor. La propiedad "Name", se puede

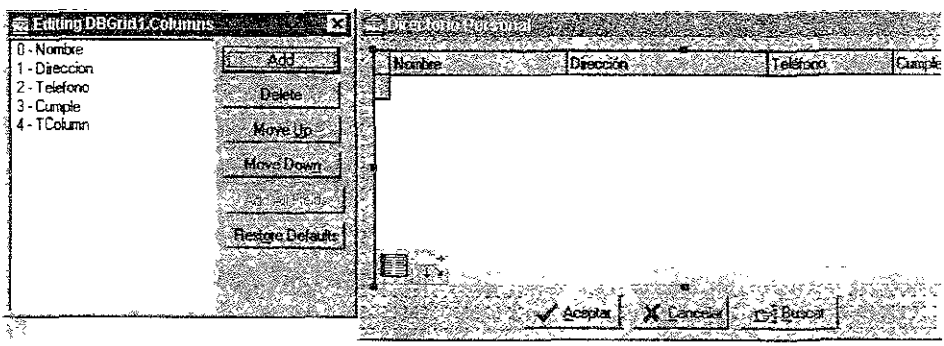
modificar para darle a la forma el nombre que el programador quiera, el valor[6] por default es "Form1".

Una vez generada la forma en blanco, se pueden ir arrastrando cada componente necesario a ésta, para que la pantalla sirva para lo que se necesita. En este caso particular, se agregaron Un *DbGrid* para que despliegue los datos de la base, un *Table* para que controle los datos, un *DataSource* para asociar el *Table* al *DbGrid* y tres *BitBtn's* para poner los botones con imágenes, de forma que la pantalla en desarrollo se ve así:



Las propiedades que se modificaron son las siguientes:

Del **DbGrid**, se le dio un doble click para que apareciera el editor de columnas de Delphi, el cual permite configurar los títulos y asociar campos de la base a cada columna del *DbGrid* para ser desplegados. En este caso sólo se agregaron 4 campos (Nombre, Dirección, Teléfono y Cumpleaños) que son los necesarios para mostrar los datos que tiene la base que se utilizará en el ejemplo



[6] El detalle de propiedades, valores y resultados puede consultarse componente por componente en la ayuda de Delphi, siendo esta bastante clara como para entender que efectos tiene en la pantalla de ejecución cada modificación que se haga

En el cuadro de la izquierda (Editor de Columnas), se puede situar el cursor sobre el renglón *"4 – Tcolumn"* y se presiona F11, se tiene así, acceso al *Object Inspector*, desde el cual se pueden modificar las propiedades que para este caso es necesario cambiar:

FieldName, presionando el combo[7] que para tal efecto existe, se puede elegir un campo para dicha columna y que el *DbGrid* lo presente. (nota si al presionar el combo no aparece ningún valor, existe un problema en la configuración del *"Table"*, el *"DataSource"* o el *"DbGrid"*).

Caption, que se encuentra si se da doble click sobre la propiedad **Title**, en esta propiedad se pone el letrero que se quiere aparezca como encabezado de la columna en la que se está desplegando algún dato determinado

Se debe hacer esto para cada columna, de forma que todas queden al gusto del programador, teniendo el campo que se necesite y el título que mejor corresponda al dato que se muestra, así se pueden poner las columnas necesarias para los campos a desplegar.

Del **DataSource** sólo se modifico la propiedad **DataSet**, que indica de dónde se tomarán los datos (*Table* para el ejemplo) que el componente asociará a algún otro componente (*DbGrid* en este caso) que permita en su configuración un *DataSource*.

Del **Table**, se modificaron las propiedades.

DataBaseName, en esta opción, que también cuenta con un combo, se presentan todos los posibles "alias" predefinidos en el DataBase Engine de Delphi, es necesario definir aquí los drivers de Base de datos o de Windows que se necesitan para tener acceso a una base de datos local o remota[8].

TableName, que permite decir que tabla de la base de datos es la que se quiere modificar o asociar al componente, ya que una base de datos puede, y generalmente, tiene muchas tablas asociadas de alguna manera.

De cada BitBtn se alteró lo siguiente:

Caption, para poner en la cara del botón el letrero más apropiado a la función que realizará el botón. Nótese que cuando en el *caption* de los botones, se pone el signo "&", la siguiente letra aparecerá subrayada y así esta letra + la tecla ALT, hará que se ejecute la función asociada al botón.

Glyph, que tiene a su cargo la imagen que el botón presentará, para asociar el archivo que contenga la imagen, simplemente se da un click sobre el botón con tres puntos (...) que

[7] Combo es el botón situado a la derecha del campo de la propiedad que tiene un triángulo dibujado con la punta hacia abajo

[8] Para detalles, ver el anexo Configuración del DBE

aparece a la derecha del campo de la propiedad y se mostrará una pantalla desde la cual, fácilmente, se puede elegir un archivo de imagen para el botón.

Los eventos de cada componente u objeto en Delphi, son acciones que suceden durante la ejecución de la aplicación, estos pueden ser muy diversos, como un click, el presionar una tecla específica, el pasar el cursor sobre un componente, el arrastrar algo de un objeto a otro, etc. Cada componente tiene una lista definida de eventos que puede reconocer, es decir, que no todos los eventos que existen pueden ser reconocidos por todos los componentes. Cuando un evento es reconocido por algún componente, se ejecuta el código que esté asociado a dicho evento; a este código se le conoce como "*event handler*".

Los eventos pueden asociarse y verse desde el Object Inspector, si se activa la pestaña destinada para esto. Por ejemplo, en la pantalla de esta primer prueba, se codificaron acciones en dos eventos de la forma, los eventos son:

OnActivate, que se dispara cuando la forma se activa, en este caso, cuando se ejecuta el programa, la única forma que este contiene entra en operación, lo que provoca que se "dispare" el evento mencionado. El código dentro del texto del programa, queda como sigue:

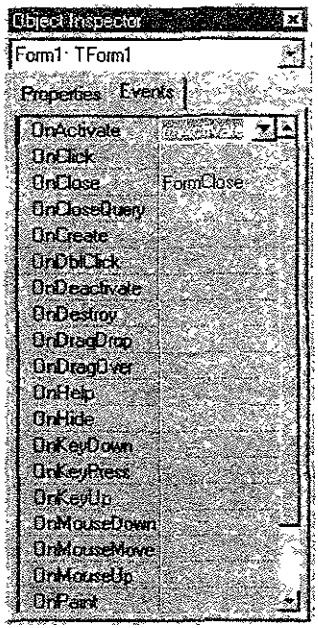
```
procedure TForm1.FormActivate(Sender: TObject);
begin
     Table1.Open;
end;
```

Este código debe poder ser reconocido por un codificador en Pascal, de manera sencilla, el código contiene lo siguiente, Se define un procedimiento llamado *FormActivate* que pertenece a la clase *Form1*, que es la forma en si, este procedimiento tiene un parámetro *Sender* de tipo *TObject*, que es un tipo con el que Delphi define todos sus objetos y componentes y se utiliza para saber desde dónde se disparó el evento. Finalmente dentro del procedimiento, sólo se pide abrir el componente *Table1*. De este modo, los datos que *Table1* tiene quedan a disposición del *DataSource*, para que desde éste se utilicen

OnClose, este evento funciona exactamente al contrario del anterior, ya que se "dispara" cuando la forma se cierra. Es decir si se da por terminado el programa (presionando la esquina superior derecha de la pantalla, la "X"), antes de desalojar la memoria y desaparecer de vista, se ejecuta el código que se encuentra en el *OnClose*. El código del evento es el siguiente

```
     procedure TForm1.FormClose(Sender: TObject; var
Action:TCloseAction);
     begin
          Table1.Close;
     end;
```

El código es muy similar al del evento *Activate* sólo que este se encarga de cerrar el *Table1*. Es importante notar que el evento *OnClose*, tiene un parámetro mas, y sirve para interrumpir el salir de la aplicación, pues en caso de que alguna condición no se cumpla, el parámetro *Action* puede regresar el valor de que ninguna acción se ejecute y así no permitir que se cierre la aplicación



El Object Inspector para la forma del ejemplo y sus eventos, se ve como lo muestra la imagen anterior. Nota: todos los eventos de la lista es capaz de reconocerlos la forma que sirve como pantalla[9].

Los Métodos son procedimientos o funciones asociadas a determinados componentes, por ejemplo, en los eventos de los que se ve el código, se usan los métodos del componente *Table:* **Open** y **Close,** estos para abrir y disponer de los datos de la base y para cerrar y dejar libres los datos. Entonces, los métodos ejecutan una serie de instrucciones predefinidas en Delphi que le facilitan al programador el uso de los componentes, ya que con instrucciones precodificadas se puede ahorrar muchas validaciones y ejecuciones parciales de código.

Toda la codificación del ejemplo puede verse en el apéndice de programas y en los discos entregados con el presente proyecto donde además de los ejemplos se incluyen el código fuente de la librería dinámica que da origen a este trabajo

[9] Para saber en que momento se "dispara" cada evento, se puede consultar la ayuda de Delphi o la bibliografía sugerida.

Con lo anterior se puede tener una primer idea de cómo usar y configurar los componentes, además de que se puede observar la diferencia entre algunos tipos de componentes, de forma que el programador se pueda ir familiarizando con el uso de estos. Es muy recomendable para aquel que quiera programar en Delphi, empiece a leer en la ayuda del propio lenguaje la utilización que se puede dar a cada componente, para saber en que caso se pueden utilizar y también como deben ser utilizados

1.1.2. ¿Cómo se codifica en Delphi?

El código en Delphi, como ya se ha mencionado, es muy familiar a la codificación en Pascal, para enunciar la forma de hacerlo, se tomará el código del ejemplo del "directorio personal" dado que es sencillo y sirve para enunciar las partes fundamentales

```
unit Unit1;
```

Es necesario definir el tipo de programa que se está codificando, en este caso se desarrolla una "unidad", que es un programa que forma parte de un proyecto en Delphi desde el cual puede ser accesado.

```
Interface
```

Al igual que en las TPU's de Pascal, se necesita de una interfaz que diga que parte del código puede ser accesada por otro programa.

```
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls,
Forms, Dialogs, Db, DBTables, Grids, DBGrids, StdCtrls, Buttons;
```

El uses también dirá que unidades o programas de Delphi o personales debe incluir la presente unidad para que funcione.

```
Type
```

En la sección de tipos, se definirán las variables, clases, objetos, etc. que incluya la unidad, en este caso, todas las definiciones fueron agregadas por Delphi, el programador únicamente se dedica a pegar componentes en la forma y a codificar dentro de los eventos que ya antes se mencionaron

```
TForm1 = class(TForm)
  DBGrid1: TDBGrid;
  DataSource1: TDataSource;
  Table1: TTable;
  BitBtn2: TBitBtn;
  BitBtn3: TBitBtn;
```

```
        BitBtn1: TBitBtn;
        procedure FormActivate(Sender: TObject);
        procedure    FormClose(Sender:    TObject;    var    Action:
TCloseAction);
      private
        { Private declarations }
      public
        { Public declarations }
      end;
```

En el ejemplo, sólo se define una clase (TForm1), que contiene un DbGrid, un DataSource, 3 BitBtn's, dos procedimientos genéricos, y ninguna declaración privada, ni tampoco pública. Nota: Fuera de la clase pueden definirse variables, procedimientos, objetos, etc. pero sólo pueden utilizarse en esta unidad, si se quiere compartir procedimientos, variables, o cualquier otra cosa, es necesario definirlos en la sección de "declaraciones públicas" para que desde cualquier otra unidad puedan ser referidas.

```
var
  Form1: TForm1;
```

En la sección de variables pueden definirse todas las variables que necesite la unidad, sin necesidad de que pertenezcan a la clase principal del programa.

```
Implementation
```

En la implementación se hace la codificación final de cada procedimiento, es decir se programa lo que la aplicación va a hacer durante su ejecución.

```
{$R *.DFM}
```

Siempre que la unidad contenga una forma, es necesario indicarle al compilador, mediante este mandato que la busque.

```
procedure TForm1.FormActivate(Sender: TObject);
begin
     Table1.Open;
end;

procedure   TForm1.FormClose(Sender:   TObject;   var   Action:
TCloseAction);
begin
     Table1.Close;
end;

end.
```

Finalmente, se ponen los procedimientos y el código que sea necesario para que la aplicación funcione como debe. Esta es la definición de los procedimientos.

En el apéndice del proyecto, se pueden ver algunos códigos más de programas, con lo que se puede empezar a hacer ejemplos para ir adquiriendo destreza en el uso del lenguaje También se dispone de los códigos fuentes del *dll* para que se pueda, desde Delphi, ver la forma en cómo se accesa a los procedimientos, y como este genera las definiciones y algunas codificaciones necesarias para el programa. Además, existe una breve introducción a la programación en Delphi, en la que se detalla mas la forma en como se debe codificar y también el uso de algunas instrucciones

1.2 ¿Qué es un archivo *dll*?

El nombre de este tipo de librerías proviene, como muchas otras cosas, de su nombre en inglés *Dynamic Link Library* (o Librerías de ligado dinámico) Los *dll's* son librerías compiladas de forma que sin importar en que lenguaje se hayan hecho, se puede ligar a otro programa o sistema en tiempo de ejecución, de manera sencilla. Este tipo de librerías aporta la ventaja que se pueden tener diferentes fragmentos de código, datos o recursos que pueden ser compartidos entre diferentes aplicaciones. Por ejemplo, si se necesita una parte de algún sistema que contenga cálculos complicados o que utilicen demasiados recursos del procesador, en un lenguaje cuyas habilidades no son las de hacer cálculos de alta precisión pero que desarrollar en él sea muy fácil y rápido, se puede optar por hacer la parte de cálculos en el lenguaje de programación C++, por ejemplo, y que este compile el código como un *dll*, de forma que desde el ambiente de desarrollo se pueda tener acceso a las rutinas de la librería dinámica, que el procesador ejecutará en la forma en como lo compiló C, ahorrando con esto tiempo de ejecución, recursos y probablemente la decisión de cambiar la plataforma de desarrollo

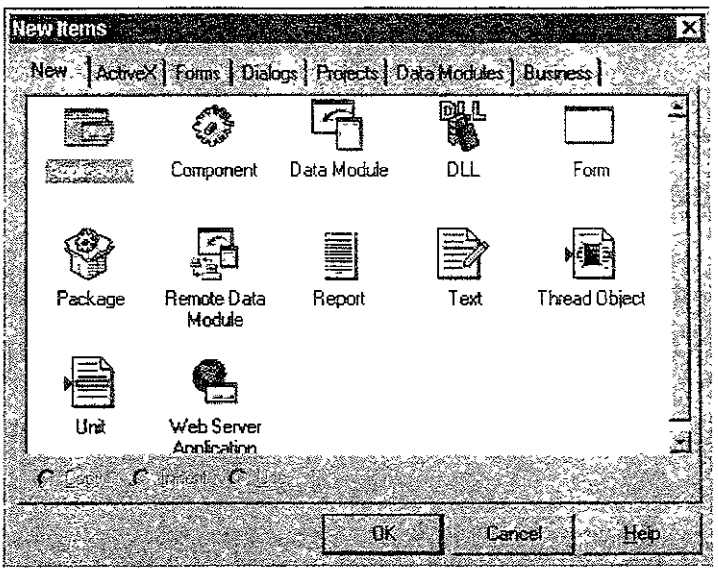
Si se desarrollan suficientes librerías de este tipo, se puede generar un API (*Application Program Interfaces*) como los de Windows, de forma que se tengan muchas rutinas que se repiten a lo largo de diferentes programas con el fin de incluir en otros fuentes librerías dinámicas, es decir, rutinas que no ocupan espacio en el programa EXE, sino que en el momento que un programa ejecutable las manda llamar, estas se accionan desde donde estén de forma que, ocupando un solo espacio, pueden ser referidas desde otros programas. Por si fuera poco, las *dll*'s pueden ser cargadas y descargadas de un programa a placer, lo que permite tener un control muy estricto acerca del espacio de memoria a utilizar, pues puede cargarse una *dll* para ejecutar un proceso único en un sistema que siempre debe estar funcionando y ser descargada en cuanto el proceso se concluye.

1.3. Conceptos fundamentales de un *dll*

En Delphi es relativamente fácil crear una librería de este tipo, de hecho, los programadores usuarios de Delphi, sólo necesitan hacer algunos ajustes a un archivo de proyecto normal (con extensión .dpr) para que sea compilado como librería dinámica. En caso de que se necesitaran mas unidades o formas, sólo se debe ir agregando una por una como si se tratara de un proyecto normal[10] El mismo procedimiento debe seguirse para

[10] Elegir File + New en el menú principal de Delphi y elegir el icono de Form o Unit.

generar un proyecto de tipo *dll*, sólo que se debe elegir el icono apropiado La pantalla en la que se eligen el tipo de programa que se desea agregar es la siguiente.



Una vez que se elige el tipo DLL, se genera un archivo de tipo .dpr que contiene el inicio de la codificación necesaria para la creación de la famosa librería. En el código que se presenta al inicio, se puede observar lo siguiente:

```
library Project1;
```

En este caso, ya no se necesita expresar el Unit que se usa en los proyectos comunes, pues lo que se desea generar es una librería.

```
{ Important note about DLL memory management: ShareMem must be
the first unit in your library's USES clause AND your project's
(select View-Project Source) USES clause if your DLL exports any
procedures or functions that pass strings as parameters or
function results. This applies to all strings passed to and from
your DLL--even those that are nested in records and classes.
ShareMem is the interface unit to the DELPHIMM.DLL shared memory
manager, which must be deployed along with your DLL. To avoid
using DELPHIMM.DLL, pass string information using PChar or
ShortString parameters. }

uses
  SysUtils,
  Classes;

begin
```

```
end.
```

El comentario hace notar que para hacer capaz de que el *dll* pueda enviar y recibir cadenas, es necesario que se ponga el ShareMem en la clausula USES y que además se debe incluir con el *dll* que se genere el DelphiMM.Dll para que la librería sea capaz de comunicar parámetros, resultados de funciones, incluso variables anidadas en registros o clases de la librería de tipo cadena Sin esto, la cantidad de cosas que ofrece Delphi para utilizar, podría resultar omitido

1.3.1 Código de Inicialización o DLLMain Entry Point.

Todo lo que se conoce como código de inicialización, es el código que se ejecuta cuando la librería es cargada, es decir, antes de que las rutinas del *dll* estén a disposición del programa que lo llama este código es ejecutado. Este código se encuentra entre el "begin" y el "end." del archivo .dpr y desde aquí se pueden inicializar las rutinas, si es necesario, correr algún tipo de código en lote o batch. Todo mientras la librería se carga a memoria, con esto se puede hacer que el *dll* corra algún proceso, sin necesidad de llamar a ninguna rutina específica, si no que corre de manera automática

En realidad lo que pasa al cargarse la librería, es que el procedimiento DllMain se ejecuta, como si fuera el evento OnActivate, siguiendo el estilo del lenguaje C, por lo tanto, este procedimiento puede ser codificado y puede mandarse llamar posteriormente. Además si se une a la sección de USES las cláusulas Windows y Forms. se puede tener acceso a algunos estados con los que se puede tener un manejo mas detallado de que ejecutar y cuando pase que evento. Los eventos que pueden ser detectados, son los siguientes:

Evento	Valor	Razón por la que se ejecuta	Posible uso
Dll_Process_Attach	1	Un procedimiento principal ha cargado el *dll* en su espacio de memoria. Se llama una sola vez por proceso	Para inicializar objetos o variables globales cuando el *dll* es cargado o ejecutar algún proceso de Batch
Dll_Thread_Attach	2	Un procedimiento a creado un "subprocedimiento" que depende de él (no se ejecuta para el procedimiento principal)	Ejecuta un proceso especial cuando ligas subsecuentes son activadas
Dll_Thread_Detach	3	Un "subprocedimiento" en el proceso cargado, ha sido desactivado	Si el *dll* incorpora código extra cuando es llamado por segunda vez, es aquí donde se puede liberar la memoria o iniciar banderas
Dll_Process_Detach	0	El *Dll*, esta siendo descargado de, al menos, uno de los	Se puede liberar cualquier objeto global, cerrar

		procesos principales en los cuales está cargado	archivos, o puertos que se estén utilizando

Para poder utilizar los diferentes eventos, se debe codificar como sigue:

```
library Project1;

uses
  ShareMem,
  Windows,
  SysUtils,
  Classes;

Procedure DLLMain(dwReason : Word);
Begin
   Case dwReason of
     Dll_Process_Atach : {Instrucciones };
     Dll_Process_Detach : {Instrucciones};
     Dll_Thread_Attach : {Instrucciones};
     Dll_Thread_Detach : {Instrucciones};
   End;
End;

Begin
   DllProc := @DLLMain;
   DllMain(dll_Process_Attach);
end.
```

Con la codificación anterior, se puede tener acceso a diferentes respuestas dependiendo de la forma en como sea cargada la librería, con esto se puede tener una librería para varios usuarios a la vez, como pretende ser Windows 95.

1.3.2. Protocolos para llamar funciones o Calling Conventions

El protocolo que se necesita para el uso de librerías dinámicas, tiene que ver con la forma en que se reciben y envían parámetros, pues como ya se mencionó la ejecución se hace en la forma en como lo haría el lenguaje original. Sin este tipo de protocolos, los programas y las librerías serían incapaces de entablar comunicación pues no hay otra cosa que se "digan" ambas partes, las rutinas se ejecutan en la librería y sólo se envían los resultados al programa que lo mandó llamar. Por esta razón, el programador que quiera que sus *.dll*'s sean utilizadas, debe ser muy específico al mencionar dichos protocolos, pues de otra forma sus rutinas serán inaccesibles.

Los protocolos especifican el orden en el que los parámetros son pasados a las funciones. También especifica, si se usan o no registros del CPU para pasar los parámetros Finalmente, el protocolo indica quien es el responsable de limpiar el stack de memoria donde sea cargado, el que lo manda llamar o la función que ha sido llamada

A continuación se mencionan los "Calling Conventions" que soporta Delphi y cómo funcionan:

Protocolo o Calling Convention	Orden de los parámetros	Limpieza del Stack
Register (Fast-Call)	Izquierda a derecha	La rutina que ha sido llamada
StdCall	Derecha a izquierda	La rutina que ha sido llamada
Pascal	Izquierda a derecha	La rutina que ha sido llamada
Cdecl	Derecha a izquierda	La rutina que llama
SafeCall	Derecha a izquierda	La rutina que ha sido llamada

Desde Delphi 2.0 el protocolo usado por default, es Register (Fast-Call), que es el mas eficiente de los protocolos, pues utiliza registros extendidos del CPU para pasar los tres primeros parámetros, haciéndolo mas rápidamente, si hubiera mas parámetros, estos son pasados siguiendo el protocolo Pascal.

Nota: La implementación del protocolo Fast-Call es soportada por productos Borland, únicamente, si se quiere usar este Convention y que la librería sea compatible con otros programas, se debe utilizar Register, que funciona exactamente igual que Fast-Call. De hecho la ayuda de Delphi no hace referencia a Fast-Call, en ella sólo se menciona Register.

Uno de los errores mas comunes en la codificación de *dll*'s es que se olvida precisar el tipo de protocolo que las funciones utilizarán, en caso de que el código de la librería sea compilado en Delphi 1, el protocolo que por default se utilizará es Pascal, sin embargo si esto pasa con alguna versión superior de Delphi, el default será Fast-Call, lo que puede traer problemas de compatibilidad del *dll*, para los usuarios de este. Por lo tanto es recomendable recordar siempre el hacer específico el protocolo a usar.

1.3.3. Exportando rutinas

Cuando Delphi guarda la librería en el disco, escribe el nombre de todas las rutinas marcadas para exportar y un "archivo imagen virtual" que sirve para cuando el *dll* es cargado en memoria por un programa, este sepa los nombres de las funciones y sus direcciones, pues sin esto el programa que utiliza la librería no es capaz de comunicarse con ella.

Un *Dll* exporta rutinas para que estas sean utilizadas por el sistema operativo, algún programa, otro *dll*, etc Estas rutinas son una interfaz con la que se pueden manipular objetos del *dll* de forma sencilla o simplemente como un grupo de funciones que llevan a cabo diversas, y en general, rutinas muy útiles. La forma mas simple para usar una *dll*, es que con esto se puede generar un "paquete" de rutinas que están relacionadas entre ellas, para que puedan ser utilizadas de forma "transparente" por otras aplicaciones.

Para poder exportar una rutina desde un *dll*, se necesita hacer algunos ajustes en el código del proyecto donde se esta generando la librería, es decir, la codificación es muy parecida a la de un proyecto normal en Delphi, pero incluye una sección mas denominada **Exports**.

Como primer paso se debe definir la función en la sección de **interface** de una unidad o después del uses del código del proyecto de la librería. Cabe mencionar, que en caso de que se tengan diferentes unidades en el proyecto, todas estas deben ser incluidas en el uses principal (del proyecto, archivo .dpr), poniendo primero el nombre de la unidad (Delphi siempre usa el nombre del archivo físico para nombrar sus unidades) seguido de la palabra reservada **in**, junto debe estar entre apóstrofes (') el nombre físico del archivo, incluyendo su extensión y finalmente entre llaves el nombre de la forma asociada a la unidad (lo que se debe escribir entre llaves, es el nombre que se le dio a la forma en su propiedad **Name**). Por ejemplo:

```
Uses

formulas in 'formulas.pas' {Formula};
```

En el proyecto que se esta presentando, todas las fórmulas y procedimientos necesarios para que el *dll* funcione se están codificando en la unidad *formulas*, esta está guardada en el archivo *formulas.pas* y la forma asociada a ella, se llama *Formula*.

El segundo paso, es agregar la función que se necesita dentro de la sección de **Exports** del proyecto. En algunos casos también se le pone un índice que diferenciará con un número a cada función, sin embargo esto no es necesario.

```
exports
        Factorial   {Index 1},
        Combinacion {Index 2},
        Pow         {Index 3};
Begin
End;
```

Para el caso de la presente librería se tiene como ejemplo tres rutinas definidas e implementadas en la unidad *formulas*, las tres deben indicarse en la sección de exports, tal como se describe y lo que está entre llaves ({x}), es opcional de escribir, pero en caso de que se quiera, se deben quitar éstas. Este número sirve para una rápida localización de las funciones por la aplicación que las utilizará, sin embargo el uso de estos índices no es recomendable pues el trazar o *debugear* el programa resultaría mas difícil, además hay gente que dice que Microsoft no recomienda el uso de éstos porque tarde o temprano desaparecerán de sus sistemas operativos, por lo que las *dll*'s que los usen, serán obsoletas.

El último paso, es codificar o darle la funcionalidad adecuada a la rutina en la sección *implementation* de la unidad en la que se definió la función, esta parte es igual que definir la función en Pascal. En los anexos del trabajo, está el código del *dll*, y ahí puede verse exactamente la codificación que se dio a las rutinas y al proyecto principal.

Los encabezados y definiciones de las funciones del ejemplo, son las siguientes

```
function Factorial(x : Integer) : Double; Register;
function Combinacion(n, r : Integer) : Double; Register;
function pow(x,y : Double) : Double; Register;
```

El código, es parte de la sección *implementation* de la unidad *formulas.pas* y como puede observarse, la definición es común a las definiciones en Pascal, sólo que al final se indica el protocolo de comunicación que utilizará el *dll*. Para las personas que ya hayan codificado un archivo *dll* en Delphi 1, les parecerá que falta la directiva *far* y *export*, sin embargo, Delphi 3 ya no necesita éstas, pero el compilador las soporta por ser compatible con versiones anteriores.

Con lo descrito hasta aquí, mas las indicaciones para usar Delphi agregadas en el anexo, se puede generar el código de un archivo *dll*, lo que haría falta, es generar la destreza suficiente del programador para adecuarse al uso de componentes y las nuevas instrucciones que se usan en este poderoso lenguaje. Ahora solo resta poner manos a la obra y empezar a practicar.

1.4. Usando el *dll*

Para poder usar las rutinas del *dll*, se deben hacer algunos pasos que resultarán sencillos a comparación de lo que Delphi debe hacer para poder utilizar una librería dinámica. Existen todo un grupo de funciones que radican en el Win32 API, y que hacen posible cargar en el mismo espacio de memoria de la aplicación que usará el *dll*, estas rutinas tienen que ver directamente con el manejo de memoria y en términos estrictos el uso de estas es transparente para el programador, pues no necesita hacer tanto manejo de funciones con parámetros referidos directamente a módulos y sus espacios, sólo debe hacer algunos ajustes al proceso de definición de sus unidades.

El proceso a seguir puede ser expresado como sigue:

1.4.1. Importando las rutinas de un *dll*

Después de que se crea una *librería dinámica* y sus funciones para exportar, es necesario cargar el *dll*, para poder usarlas. Después de que el código de una aplicación solicita y define las funciones que son exportadas, el código puede hacer uso de ella.

Si la aplicación simplemente tratara de llamar una función, este recibiría un error a cambio, pues el compilador no reconocería la función y, obviamente, no podría ni compilarla ni incluirla en el código que se dispone a ejecutar Así que es necesario poner una declaración en el programa fuente que incluirá la *dll* por cada una de las funciones a las que tendrá acceso; de hecho, el proceso de definición es normal al de una función en la que se escribirán líneas de código necesarias, sólo que en vez de poner todo el código necesario se hace la llamada a la función con los parámetros requeridos.

De hecho existen dos formas de accesar las rutinas de un *dll* desde Delphi, la forma en como se describió previamente es la forma **explícita**, pues hace una definición "estática" de la función utilizando algunas declaraciones diferentes La otra forma es la **implícita** que hace el uso de memoria paso a paso, lo que permite cargar y descargar el *dll* de la memoria, pero utilizando rutinas de manejo directo de memoria y utilizando mas pasos.

1.4.2. Forma implícita de importar rutinas

Para la forma implícita, la codificación de los procedimientos para que estos hagan la liga a la librería, es muy parecida a las comunes, pero utilizan algunas palabras extras que son traducidas por el *Linker* de Delphi haciendo el manejo de la memoria y su carga y liberación cuando se crea y se destruye la forma que contiene estas definiciones[11]

Las directivas que se necesitan escribir después de una definición normal de funciones o procedimientos, son:

Protocolo · Este igual que para exportar especifica la forma en como funcionará la limpieza del *stack* y el orden en como se recibirán los parámetros de la función. Los protocolos o Calling Conventions están definidos en la tabla 1 de la sección 1.3.2. Protocolos para llamar funciones o Calling Conventions

External : Que indica que la función que precede a este comando está y será ejecutada desde una librería dinámica. Lo que sigue después del *external* es el nombre físico de la *dll*, es decir el archivo que está guardado en el disco. El nombre debe ser especificado entre apóstrofes (') y no necesariamente debe llevar extensión.

Name : Después de esta cláusula debe ponerse el nombre real de la función, tal y como fue definida en el *dll*, Con esto se accesa directo a la función que se puede ejecutar así como a su dirección de memoria y todas las referencias que están incluidas en el archivo de referencia que acompaña al *dll*.

Algunos ejemplos de estas definición son:

```
    function Factorial(n:integer):double;register;external
'Pyetool.dll';
    function Comb (a,b:integer):double;register;external
'Pyetool.dll' name 'Combinacion';
```

[11] Las funciones que son utilizadas para cargar y descargar librerías directamente en el sistema operativo, son LoadLibrary y FreeLibrary, con sus respectivos parámetros y funciones complementarias, que en su mayoría regresan direcciones de memoria para que éstas sean usadas desde el programa, utilizando apuntadores de memoria, mientras se ejecuta

Estas definiciones están, originalmente, declaradas fuera de la definición de la clase principal de la unidad en que se utilizarán, y en el primer caso se ve que no existe la declaración *name*, esto se debe a que el nombre de la función es el original, es decir, en el archivo *Pyetool.dll* existe y se exporta la función Factorial y tiene como parámetro una variable de tipo *integer*. Por el otro lado, la función *Comb* no existe en el *dll*, por eso es necesario agregar el nombre original de la función pues de otra forma, el *Linker* no podría hacer uso de la función. Además se utiliza como protocolo el *Register*. Es muy importante saber con que tipo de protocolo se está exportando la rutina, pues en caso de que se defina uno diferente se corre el riesgo de tener resultados erróneos sin saberlo, pues el programa se compilará sin problemas.

Las definiciones deben hacerse en el orden en que se especifica en los ejemplos, de otro modo se incurre en un error. Deben estar incluidas en la sección *interface* del programa.

También se puede hacer referencia a la función por su índice, si este fue definido cuando se exportó la función, lo único que se debe incluir es en vez de **name** el mandato **index** y el número que tiene asignado la función. Como ya se dijo, no es recomendable utilizar este tipo de índices, pues al parecer están en peligro de extinción.

```
    function Factorial(n:integer):double;register;external
'Pyetool.dll' index 1;
    function Comb (a,b:integer):double;register;external
'Pyetool.dll' index 2;
```

Este caso es el mas sencillo, de hecho buena parte de los programadores utiliza este tipo de llamadas a *dll's*, sin embargo, la forma en como funciona hace que la librería esté cargada siempre en memoria, igual que el programa que la manda llamar. Este problema puede ser resuelto creando y destruyendo la forma en la que está definido el uso del *dll*, así que puede ser tan buena opción esta como la explícita, que se describe a continuación

1.4.2. Forma explícita de importar rutinas

Para este caso, se debe codificar un poco mas sin embargo puede resultar interesante saber el manejo de memoria y las funciones que se utilizan para cargar *dll's* de forma manual. En primer lugar se debe definir un **tipo**[12] específico capaz de apuntar a la dirección de memoria de un procedimiento. Por ejemplo:

```
Type
TProc = Procedure(nombre : Pchar); Pascal;
```

La definición anterior crea un lugar donde se puede asignar un procedimiento, tomado del *dll*. Posteriormente, en algún procedimiento específico se debe hacer la asignación de la rutina.

[12] Se refiere a un tipo de variable y sirve para especificar que tipo de datos contendrá una variable definida en Delphi, todas las variables que se quieran usar deben estar definidas y la definición consta de un nombre y un tipo, algunos tipos son string, integer, double, etc.

```
Procedure AsignaFuncion;
Var
   Proc : Tproc;
   PDll : Thandle;
   X : Double;
Begin
   PDll := LoadLibrary('ProbEst.dll')
If PDll <=0 then {Error}
```

(Aquí se puede utilizar la función GetLastError que dirá el número de error que ocurrió)

```
@Proc := GetProcAddress(PDll, 'Factorial');
x := Proc(10);
```

(x contendrá el factorial de 5, siempre y cuando no ocurra ningún error al asignar el procedimiento a Proc)

```
FreeLibrary(PDll);
End;
```

Como se puede observar, el código necesario para utilizar el *dll* de esta forma es mayor, pero también en un mismo procedimiento se carga y se libera de la memoria, por lo que dejaría de ocupar espacio en cuanto no se usara, de esta forma se cubre la problemática de tener que crear y destruir una forma en Delphi, ya que si esta es indispensable en el funcionamiento de un sistema, entonces no puede ser destruida, así sólo se ocupará la memoria del *dll* cuando el procedimiento se ejecute lo cual puede ser muy útil en algunos casos.

Con esto, debe poder hacerse un *dll* del tamaño que se necesite, utilizando codificación casi igual que la que se usa en Pascal, salvo algunas instrucciones nuevas y algunos mandatos diferentes que los que se conocían desde el inicio de Pascal como lenguaje.

Capítulo II – Conceptos estadísticos del *dll*

A continuación se presenta la forma en como se agruparon la teoría estadística, conocimientos en Delphi y capacidad de diseño y desarrollo de sistemas para obtener las rutinas del *dll* que permiten obtener algunos resultados estadísticos de forma fácil, y sobre todo, utilizando librerías dinámicas, que pueden ayudar en la creación de aplicaciones rápidas y poderosas de forma que el desarrollo no sea tan detallado o costoso. De esta manera se provee, con la librería, una variedad de herramientas de fácil acceso y buen rendimiento, en cuanto a tiempo de ejecución se refiere, para el desarrollador de software que lo requiera.

2.1. Conceptos Básicos y herramientas elementales

Antes de poder entrar en el detalle de las funciones, el código, la forma de utilización de las rutinas, etc. es necesario hacer una aproximación a lo que es la estadística y su utilización. Existen muchos libros especializados en hablar de la teoría estadística, así que para este proyecto se hará una recopilación de algunas partes que le dan forma al *dll* y la forma en como se utilizaron.

2.1.1. Introducción a la estadística

Aunque para algunos la estadística sea algo demasiado ajeno o se imaginen que jamás tendrán que lidiar con algo que tenga que ver con ella o les pueda parecer absurdo que existan personas que les guste y trabajen todos los días utilizando parte de sus herramientas, se sorprenderían al saber que cosas tan simples como el promedio de calificaciones obtenido en una materia, es uno de los estadísticos mas comunes. Desde los investigadores que determinan la resistencia de los diseños y materiales con que se hacen los autos a partir de la repetición de experimentos (como chocar autos prototipos), hasta las encuestas que determinan el "raiting" de una estación de radio, llevan en sus entrañas una muy fuerte relación y una base estrictamente formal respecto a la teoría estadística, todo cuidadosamente estudiado para que los resultados obtenidos tengan validez y aporten información utilizable y con parámetros que puedan medir la cantidad de precisión que maneja cada dato obtenido.

Existen algunas definiciones puntuales de estadística, se mencionarán algunas:

- Estadística es una rama de las matemáticas que se encarga de la recopilación, el análisis, la interpretación y la presentación de una gran cantidad de datos numéricos[13].
- Estadística es la rama del método científico que trata de los datos reunidos al contar o medir los las propiedades de alguna población[14]

[13] New Collegiate Dictionary de Webster
[14] Kendall y Stuart

- La Estadística trata con métodos para obtener conclusiones a partir de los resultados de los experimentos o procesos[15].

Y así como esas se pueden encontrar tantas definiciones como profesionales de la estadística existan, así que solo se dirá de ella que es una colección de herramientas con fundamento matemático que permiten sacar conclusiones válidas para conjuntos grandes de datos, a partir de un fragmento de ellos, diciendo también la cantidad de precisión que cada dato arrojado tiene A esta parte de las herramientas se le conoce como *estadística inferencial.* Por otro lado existe la parte de herramientas que se dedican a dar valores con los que se puede generalizar a un conjunto de datos pues toman en consideración a cada uno de los elementos de una población y se genera información con ellos.

Para algunos casos la estadística puede servir como un adivino, sólo que en este caso se expresará en vez de un resultado puntual, un rango posible de valores acompañado de ciertas probabilidades de error.

2.2. Estadística Descriptiva

La estadística descriptiva se ayuda de muchos modelos matemáticos que permiten obtener un solo valor que se puede utilizar para describir, en lo general, las características de una población o conjunto de datos del interés de alguna persona. También se apoya de gráficos en los que se pueden descubrir, a priori, tendencias, valores que se observan un mayor número de veces, promedios, etc. Ayudado de este tipo de herramientas la estadística descriptiva puede hacer de manera sencilla la "radiografía" de un grupo de interés

2.2.1. La media, la mediana y la moda

Los datos que estas herramientas nos proporcionan son sin duda medidas de resumen de las distribuciones de frecuencia que sigue una población particular, así que ahora se le prestará atención a las medidas que aportan información acerca de cómo se agrupan los datos de manera natural.

En primer lugar se describen las medidas de tendencia central, llamadas así porque indican de manera simple y desde diferentes puntos de vista, el punto medio o típico de una distribución. A estas medidas también se le conocen como medidas de localización pues no dicen cual es el alcance de un grupo de datos, ni que tan separados están éstos, pero si dicen cuál es al que tienden la mayoría.

Por ejemplo, estas medidas pueden ser las siguientes

La **media aritmética** de una distribución no es otra cosa que el promedio de la que la mayoría de las personas puede hablar, a este valor también se le conoce como esperanza

[15] Fraser

matemática o valor esperado de una distribución. Para definir el proceso numérico que sigue esta, se utilizará el ejemplo clásico de las calificaciones

Supóngase que se tiene un alumno desea saber en que semestre ha conseguido mejores resultados, en cuanto a calificaciones se refiere, si en el que acaba de terminar o en el anterior. Para analizar el problema se tiene lo siguiente

Semestre	Mat. 1	Mat 2	Mat 3	Mat 4	Mat 5	Mat 6	Mat 7	Mat 8	Mat 9	Mat 10
1	8	6	9	10	10	7	7	7	9	6
2	8	7	7	6	10	8	7	N.E.	N.E.	N.E.

Las materias no son las mismas y en las casillas que se muestra N.E. significa que la materia no existió en ese semestre.

Con esto se pueden hacer diferentes análisis, en primer lugar, se podría ver cuál fue la mejor y peor calificación de cada semestre y compararlas, sin embargo en ambos existen la mayor (10) y menor (6) calificaciones posibles en el sistema de calificación mas común en México, por lo que de primera vista parecería un empate en los resultados.

Un segundo criterio puede ser el pensar en la suma de las diferentes calificaciones obtenidas, teniendo así un solo número que comparar para decidir, así que del semestre 1 se obtiene un total de 79 y del semestre 2 el total es 53, con lo que se tiene que por aplastante ventaja, el semestre 1 es el que tiene mejor rendimiento. Razonando un poco al respecto se puede observar que la decisión tiene muchas cosas en contra pues aunque el resultado es mucho mayor, también se están considerando tres materias mas que en el semestre 2 no se cursan y que si en el 1, así que aunque ya existe un solo número por semestre este aún no es comparable.

Dado lo anterior, nació el concepto de *promedio* o *media aritmética*, con este no sólo se toma en cuenta el total de los puntos alcanzados, sino que también se cuenta el número de observaciones (en este caso calificaciones) que aportan datos a la suma, haciendo que no sólo se sumen las calificaciones, sino que además se dividan entre el total de datos, obteniendo un resultado balanceado dependiendo de la muestra obtenida. La *media* puede visualizarse como el número que puede sustituir a cada uno de los números que participan en su cálculo para obtener el mismo resultado al sumarlos. La *media* aunque pueda sustituir a todas las observaciones, no necesariamente existe dentro del número de valores posibles, así para el ejemplo los promedios se calculan así:

Semestre 1 = 79/10 = 7.9 y
Semestre 2 = 53/7 = 7.57

Ahora si existe un valor que tome en cuenta no sólo la suma total, sino que además se consiguió el valor que puede ser el representante de cada uno de los involucrados en el experimento, así, se puede ver que efectivamente el semestre 1 tuvo un mayor rendimiento, sin embargo la ventaja no es tan grande como se veía en la conclusión anterior.

Es importante notar que aunque la media puede ser un dato interesante, existe toda una serie de herramientas y un conjunto de fundamentos y teoría para conseguir estandarizar los valores que se obtienen de muestras diferentes para hacerlos comparables, de modo que no se recomienda hacer grandes deducciones a partir de un solo dato, a continuación se expresan algunos datos que pueden aportar mayor información acerca de la distribución de una población y que pueden acercar al que toma decisiones a un punto mas sensato.

Existen algunas diferencias en el cálculo de la media dependiendo de la disposición de los datos, sin embargo el *dll* utiliza la forma a la que puede ser reducida cualquiera de los casos en que se disponga de una población, solo se mencionará que en algunos casos se hace distinción de si los datos están o no agrupados, de si deben ser ponderados o deben crecer geométricamente, etc cada uno de los casos es una adaptación de la fórmula que a continuación se presenta y que no servían cuando el contar con una computadora era muy complicado, así que si se sabe plantear el problema de forma adecuada, se puede resumir simplemente a:

$$\mu = \frac{\sum_{1}^{n} x_i}{n}$$

Para obtener resultados se necesita saber lo siguiente:

- n : Que es el total de observaciones de una población o muestra.
- x_i : Que son cada uno de los datos que pertenecen a la población, en el ejemplo previo, x_i son cada una de las calificaciones por semestre y n es el total de calificaciones

A continuación se presenta la **mediana**, que es otra medida de tendencia central y que representa el valor que está exactamente a la mitad de las observaciones, sin considerar que haya datos que se repitan, simplemente es el dato que al ordenar todas las observaciones se encuentra justo a la mitad de ellas. Este punto es el parteaguas de una población, si se considera este punto como pivote, se sabe que arriba de el están la mitad de todos valores y de bajo de él la otra mitad

Existe una forma sencilla de saber que lugar de los datos ocupa la mediana, si se aplica la fórmula:

$$Med = \frac{n+1}{2}$$

Donde:

- n : Es el total de observaciones de la población o muestra

Así se puede saber, de los datos ordenados que lugar ocupa la mediana, como se divide entre 2 es fácil adivinar que en caso de que n sea par se tendrá un lugar "x 5" como resultado de la operación, esto se podrá ver mas claro en el ejemplo.

Si se tienen los datos siguientes:

Posición	1	2	3	4	5	6	7	8
Valor	2	4	6	8	10	12	14	16

Para este caso la mediana ocupa el lugar: (8+1)/2 = 4 5 – ésimo, es obvio que el lugar sólo expresa un resultado matemático, pues en la realidad no existen, dentro de los ordinales, fracciones, sólo números enteros. Para solucionar el problema se obtiene el promedio entre los valores 4° y 5°, por lo tanto la media será 9, y así existen 4 valores previos y 4 posteriores a ella.

Finalmente, la **moda** que es un termino muy usado y alrededor del cual mucha gente gasta fuertes cantidades de dinero e ilusiones, también es un valor de tendencia central y también es una de las herramientas con que cuenta la estadística para conseguir la radiografía de los datos que se mencionaba en un principio.

Al igual que en lo cotidiano, la moda es el dato que mas se repite dentro de un conjunto de ellos, no se calcula por medios aritméticos convencionales, lo único que se puede hacer es contar cada uno de los elementos diferentes y determinar así cuál es el que con mayor frecuencia se presenta, a este valor se le denomina moda.

Con el conocimiento de estos valores se puede formar una idea al respecto de cómo se vería la gráfica de un conjunto de datos antes de tener que hacer esta, se puede tener una aproximación al sesgo que los datos presentan, la altura, si tiene varios o sólo un pico, etc. aunque no son los datos mas adecuados para determinarlo, pueden servir para una primera aproximación a la toma de decisiones.

Codificación en Delphi.

Después de haber hecho el análisis y definición de las herramientas, ahora se programarán en Delphi adecuándolas de modo que se pueda utilizar desde una *dll*, para lo cual se tomó en cuenta lo siguiente.

En Delphi existe una forma fácil de crear listas de datos, pues existe un objeto predefinido que puede manejar cadenas o *strings* sin que el programador tenga que poner atención en los apuntadores que esta utiliza, así sólo es necesario llamar algunas rutinas del objeto para administrar y accesar los datos, así que se decidió tomar el *TStringList* predefinido en Delphi para almacenar temporalmente los datos que se necesitan operar para obtener los resultados estadísticos requeridos. En el código, sólo se agregó una línea en la sección de variables de la unidad, no de la clase. El código mas o menos es el siguiente.

```
unit formulas;
interface
uses
const
type
```

Esta parte de código normalmente se crea automáticamente por Delphi, aquí sólo se muestra como orientación extra para que sea mas claro donde esta definida la lista de cadenas, pues en el código de la *dll*, completa es mucho mayor.

```
var
{Xi, es la lista donde quedarán los datos que se reciben para
operar}
  xi : TStringlist;
```

Después de que se definió la lista, es necesario hacer algunas rutinas extras para poder manipular el contenido de la lista desde otro programa, pues esa es la intención de hacer la librería, así que se definieron las siguientes rutinas como interfaz

```
function LimpiaDatos : Boolean; register;
{Prepara la lista de cadenas para subir datos, es decir, la
deja en blanco}
begin
   try
      xi.Clear;
      result := true;
   except
      result := false;
   end;
end;
```

La mayor parte de estas rutinas de interfaz sólo harán acceso a las rutinas que el *TStringlList* ya tiene definidas pero que desde algún lenguaje distinto a Delphi serían inaccesibles, por ejemplo la rutina anterior sólo ejecuta uno de los métodos del objeto llamado *Clear* que se encarga de limpiar y reiniciar la memoria disponible para la lista, pero además regresa un valor para indicar si el resultado fue exitoso o no, por eso la rutina regresa un tipo *boleano* (falso o verdadero)

```
function IncluyeDatos (x : Double) : Boolean; register;
{Agrega los datos dándoles formato de cadenas}
begin
   try
      xi.add(FloattoStr(x));
      result := true;
   except
      result := false;
   end;
end;
```

Para este caso, también se accesa a una rutina mas del objeto, *add* que se encarga de agregar cadenas a la lista, actualizando el contador de elementos, el lugar en que la nueva cadena quedó y la lista en general. En caso de que el dato no sea un valor numérico o pase cualquier otra anomalía, entonces el resultado de la rutina será falso y el elemento no será agregado a la lista que se está manipulando.

```
function EliminaDato (n : Longint) : Boolean; register;
{Elimina los n últimos datos de la lista}
var
   i : Longint;
begin
   try
      i := 1;
      while i<=n do
      begin
         xi.delete(xi.count - 1);
         inc(i);
      end;
      result := true;
   except
      result := false;
   end;
end;
```

En este caso no solo se está utilizando el método del *StringList*, sino que además se le incluye cierto valor al enviar el parámetro para decir cuantos elementos se desean eliminar, para esta rutina pudo haberse utilizado una instrucción for y al parecer hubiera sido mas fácil, sin embargo si se desea aprovechar el máximo espacio posible dentro de la lista, se necesitarían variables de gran capacidad, pues la lista lo que realmente utiliza para almacenar son apuntadores de memoria, por lo que mientras la máquina en la que se ejecute la rutina tenga espacio, es posible almacenar datos en la lista. Por eso sólo se agregó el contador para determinar cuántos elementos se desean quitar, sin importar que lugar de la lista ocupan. Para este caso se utiliza el método *delete* que quita el elemento actualizando el contador, los apuntadores y la lista en general

```
function OrdenaDatos : Boolean; register;
{Hace que en la lista queden los datos ordenados}
begin
   try
      if not xi.sorted then xi.sort;
      result := true;
   except
      result := false;
   end;
end;
```

Finalmente, se utiliza una rutina mas que hace llamado a otro de los métodos del objeto, el *Sort*, que ordena los datos que están en la lista, dejándolos en ella misma y cambiando la propiedad *Sorted* a verdadero, por eso se hace la validación antes de ejecutar el comando, ya que si la lista está ordenada, entonces no es necesario volver a correr la rutina.

Una vez que se tiene dónde almacenar y manipular datos, se pueden generar ahora las rutinas que calculen los valores estadísticos necesarios

Primero se presenta el cálculo de la media, esta rutina utiliza un par mas que se encargan de tareas sencillas y muy repetitivas en la estadística, así la rutina es

```
function Media : Double; register;
{Calcula la media aritmética de los datos de la lista}
begin
   try
      result := sum / cuenta;
   except
      result := -1;
   end;
end;
```

Para este caso, se vuelve a recurrir al valor negativo como resultado pues lo que se espera de la rutina es un valor y no un estatus de éxito como las primeras del capítulo. Se puede observar que para calcular la media se utilizan dos rutinas que también fueron hechas para la *dll* nada mas, *sum* y *cuenta*, el código de ambas se presenta a continuación

```
function cuenta : LongInt; register;
{Regresa el número de datos que contenga la lista, si se está
familiarizado con el manejo del tipo
 TStringList, se puede evitar llamar a esta función}
begin
   try
      result := xi.count;
   except
      result := -1;
   end;
end;
```

La rutina *cuenta*, regresa el valor de la propiedad *count* del *StrmgList*, al igual que las rutinas anteriores sólo se está accesando un valor del objeto, pero en este caso no se regresa un estatus, sino que se devuelve el número de elementos que existen en la lista.

```
function sum : Double; register;
{Regresa la suma de los datos de la lista}
var
   i : Longint;
begin
   try
      result := 0;
      i := 0;
      while i < xi.count do
      begin
         result := result + strtofloat(xi[i]);
         inc(i);
      end;
```

```
    except
       result := -1;
    end;
end;
```

Por otro lado, la función *sum* de la *dll* regresa la suma de todos los valores de la lista, en este caso se vuelve a recurrir a un ciclo mediante la instrucción while con el fin de que se pueda tener acceso al mayor número de elementos posibles, si se pone atención en el código se podrá ver el inconveniente de utilizar un *StringList* para almacenar valores, cuando se hace esto, se debe cambiar de cadena a numérico el valor que lleve el elemento de la lista, por esa razón se utilizó la función predefinida en Delphi *StrtoFloat*

Para el caso de la mediana y la moda las rutinas fueron codificadas de la siguiente forma intentando optimizar el rendimiento de la *dll* pero sin hacer muy complejo el cálculo.

```
function Mediana : Double; register;
{Calcula la mediana de los datos de la lista}
begin
   try
      OrdenaDatos;
      if (cuenta mod 2) = 1 then
         result := StrtoFloat(xi[trunc(cuenta/2)+1])
      else
         result    :=    (StrtoFloat(xi[trunc(cuenta/2)])    +
StrtoFloat( xi[trunc(cuenta/2)+1]))/2;
   except
      result := -1;
   end;
end;
```

Nota Este código se modificó cuando se generó la rutina *Cuartil* que se verá mas adelante, sin embargo, la codificación se queda como ejemplo.

Para obtener la mediana, sólo se utilizaron dos rutinas previamente descritas, *OrdenaDatos* y *cuenta*, y lo demás es un poco de lógica de programación y sentencias que en Pascal también existen. Con el *if (cuenta mod 2) = 1* se determina si el número de elementos es impar, de ser así, solo se necesita el valor entero (para eso se utilizó *trunc*) de dividir el número de elementos entre 2 y sumar 1; en caso de que el número de elementos sea par, se necesita un promedio de los elementos que estén en medio de la lista, así es como se obtiene el cálculo de la mediana. Para la moda, la función es:

```
function Moda : String; register;
{Calcula la moda de los datos de la lista}
var
   i, n : LongInt;
   ldato, lcuenta : TStringlist;
   dato : String;
begin
   try
```

```
    {Primera sección de la rutina}
          OrdenaDatos;
          ldato := TStringlist.Create;
          lcuenta := TStringlist.Create;
          i := 0;
          n := 0;
          dato := xi[i];
          while i<xi.Count do
          begin
             if dato = xi[i] then
                inc(n)
             else
             begin
                ldato.add(dato);
                lcuenta.add(inttostr(n));
                n := 1;
                dato := xi[i];
             end;
             inc(i);
          end;
          ldato.add(dato);
          lcuenta.add(inttostr(n));
    {Segunda sección de la rutina}
          n := strtoint(lcuenta[0]);
          for i:=1 to lcuenta.count-1 do
             if              n<strtoint(lcuenta[i])              then
n:=strtoint(lcuenta[i]);
          i := 0;
          result := '';
          while i<lcuenta.Count do
          begin
             if n = StrtoInt(lcuenta[i]) then
                result := result + ldato[i] + ',';
             inc(i);
          end;
          ldato.free;
          lcuenta.free;
       except
          result := '';
       end;
    end;
```

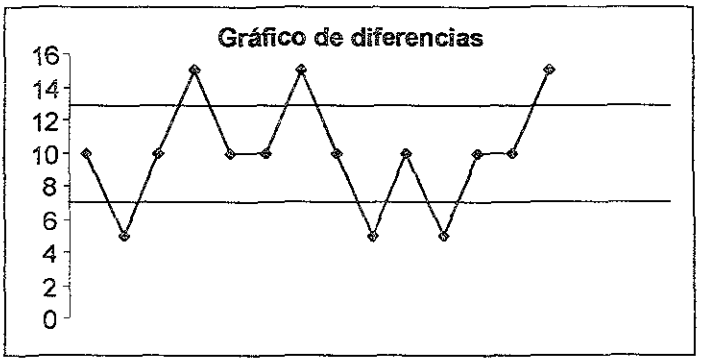
Para este cálculo, la rutina se dividió en dos partes, en el primero se utilizan un par de *StringList's* mas, para utilizarlos como de almacenamiento temporal, en la primer parte se lee la lista principal (xi) y se van guardando los valores leídos en las dos listas extras, en *ldato*, se guardan todos los elementos que son diferentes, pero sólo una vez, mientras que en *lcuenta*, se guarda el contador de elementos que se lleva por separado, teniendo al final dos listas para determinar qué elemento se repite cuántas veces. La segunda sección de la función determina el máximo de la lista de conteo para así buscar en las listas preliminares que elementos tienen el máximo de repeticiones. Es interesante ver en este segmento de código cómo se crea y libera el espacio en memoria para las listas temporales, sólo se

necesita usar *TStringList.Create* y *Free* asignados a una variable de tipo *TStringList*, para poder hacer uso de ella como tal

2.2.2. La varianza y la desviación estándar

Hasta ahora el proceso de descripción de datos puede estar claro, sin embargo también está incompleto. Los datos que se obtienen de las medidas de tendencia central son bastante ilustrativos, pero también pueden aportar problemas a un proceso de análisis, ya que sólo dicen valores típicos que no necesariamente son los mejores a considerar.

Para hacer un poco mas claro la importancia de conocer la varianza y la desviación estándar, supóngase que existen los siguientes datos obtenidos de haber medido las diferencias del tiempo en que un experto estima tardará resolver un problema y lo que se tarda en realidad. Estas diferencias pueden dejar fuera de competencia a un despacho de consultoría que base sus tarifas en proyectos contemplando penalizaciones por retardo del producto prometido, por lo tanto, para las personas encargadas del mercadeo del negocio puede ser de suma importancia saber si las diferencias en cuestión son o no significativas Los datos se mostrarán gráficamente:



De la gráfica se pueden deducir los datos que la forman, además se puede observar que los datos están acotados dentro de dos líneas (la mayoría) dados como límites de tolerancia de datos, es decir, los datos que queden dentro de las dos líneas son valores cuyas variaciones no son significativas, sin embargo se debe tener cuidado con los valores que no cumplen con este caso pues pueden meter en problemas a la persona que comprometa entregas con el cliente. Si se elabora una gráfica como la anterior se pueden ver los problemas, pero si sólo se tienen las medidas de tendencia central se tendría lo siguiente:

Los valores de tolerancia van desde el valor 7 hasta el 13, por otro lado la media de los datos graficados es = 10.0, la mediana = 10.0 y la moda = 10.0. De aquí se podrían tomar algunas decisiones que pusieran en riesgo los intereses del despacho del ejemplo, ya que al parecer los datos están controlados y dentro del rango aceptable de valores.

A partir de aquí nace la necesidad de conocer que tan homogéneos son los datos, ya que aunque los "valores resumen" dicen una cosa, la realidad es otra y significativamente distinta Así que ahora se tiene un nuevo grupo de medidas, las medidas de dispersión de los datos, éstas aportarán información importante y que detalla mas el esqueleto del grupo de datos que interese analizar.

La **varianza** sirve para determinar que diferencia tienen los datos contra la media, es decir, que tan alejados están los datos del valor típico, así lo que se calcula es el promedio de las diferencias al cuadrado de cada uno de los valores. En otras palabras, la varianza es la desviación absoluta promedio de los datos de una población con respecto a la media de la misma. La fórmula para el cálculo es·

$$\sigma^2 = \frac{\sum (x-\mu)^2}{N} = \frac{\sum x^2}{N} - \mu^2$$

De aquí se pueden deducir las siguientes notaciones·

- σ^2 : Varianza
- x : Cada uno de los datos de la población o muestra en experimentación.
- μ . La media de la población o muestra en experimentación
- N : Total de los elementos de la población o muestra en experimentación.

Para el cálculo de la **desviación estándar** sólo se necesita calcular la raíz cuadrada de la varianza, pues la d.e. es la raíz de la desviación absoluta promedio de los datos con respecto a la media. Así el cálculo es muy simple, cuando se conoce la varianza.

Para el caso del ejemplo, se puede recurrir a la opción de involucrar la desviación estándar de la muestra con los valores de tendencia central de la siguiente forma:

La σ^2 de los datos es = 10.714285, mientras la σ = 3.273268, de estos se puede hacer una suposición de distribución normal para los datos de la población[16], de donde se podrían conocer el peor y mejor de los casos de acuerdo a los datos manipulados. Para este caso los límites deberían ser, siguiendo la distribución normal, en el mejor de los casos 0.1802 y en el peor 19.8198. Así que definitivamente el proceso de donde salieron los datos, no es precisamente un caso muy estable o recomendable para utilizar el esquema de cobranza como se planteó, sin embargo conocer estos valores no habría sido posible sin conocer la dispersión de los datos.

Codificación en Delphi.

Para el caso de la varianza, se utilizó una fórmula resumida para su cálculo, la función incluye además de las que se explicaron en esta parte del trabajo, los grados de libertad utilizados para darle mayor precisión al cálculo en caso de estar utilizando una muestra de los datos. El código es el siguiente:

[16] De acuerdo a la distribución normal, si los datos se adecuan a su distribución de probabilidad, entonces el 99.73% de los datos estará contenido entre [μ-3σ, μ+3σ]

```
function Varianza(n : Longint): Double; register;
{Calcula la varianza de los datos con n grados de libertad}
begin
   try
      result := ((cuenta*sum2)-pow(sum,2))/(cuenta * (cuenta -
n));
   except
      result := -1;
   end;
end;
```

En este caso la codificación se simplificó mucho ya que sólo se utilizaron funciones que ya estaban definidas, sobre datos que ya deben estar dados de alta antes de llamar a esta función. Para este caso no parece una buena idea generar una rutina para el cálculo de la desviación estándar pues al saber la varianza sólo se necesita obtener su raíz cuadrada.

2.2.3. El mínimo, máximo y el rango

Estos valores pueden servir también en el caso anterior, si se conoce que el proceso es tan estable que los valores que se obtengan aquí, sean los únicos valores posibles, que quiere decir, no es necesario tener gran conocimiento en estadística para encontrar el significado de los valores que ahora se describen, por un lado, el **mínimo** es el valor más pequeño encontrado en la población o muestra que se está analizando, el **máximo** por el contrario es el valor mayor que se pueda encontrar en la misma población o muestra. Finalmente el **rango** es la diferencia entre el máximo menos el mínimo o la distancia que existe entre el valor mas pequeño y mas grande del grupo de datos.

Es fácil observar que si se puede garantizar que el proceso del ejemplo siempre tendrá esos valores como máximo y mínimo, se puede trabajar sin la suposición de normalidad en los datos y entonces se deberá trabajar en los casos mejor y peor (o *máximo* y *mínimo*) para que quedaran siempre en el *rango* de tolerancia para que el proceso sea rentable.

Codificación en Delphi.

Para estos casos las rutinas son las siguientes:

```
function Minimo(yi : TStringlist) : Double; register;
{Regresa el mínimo valor de un grupo de datos almacenado en la
lista que pasa como parámetro}
begin
   try
      OrdenaDatos;
      result := StrtoFloat(yi[0]);
   except
      result := -1;
   end;
end;
```

```
function Maximo(yi : TStringlist) : Double; register;
{Regresa el máximo valor de un grupo de datos almacenado en la
lista que pasa como parámetro}
begin
   try
      OrdenaDatos;
      result := StrtoFloat(yi[yi.count - 1]);
   except
      result := -1;
   end;
end;
```

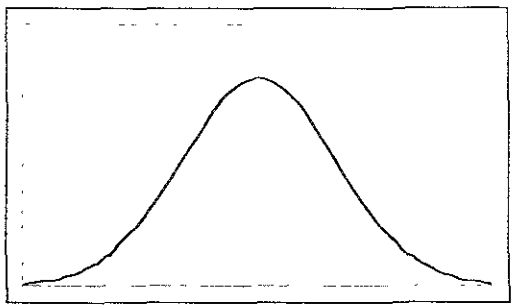
Para estas dos funciones la implementación fue muy sencilla, ya que sólo se deben ordenar los datos y escoger el primero (mínimo valor) o el último (máximo) pues la forma en como se ordenan los datos es ascendente. Así no se necesitó de nada mas que un par de instrucciones efectivas además del "seguro" contra errores.

```
function Rango : Double; register;
{Devuelve el rango de los datos}
begin
   try
      result := Maximo(xi) - Minimo(xi);
   except
      result := -1;
   end;
end;
```

El Rango también es un valor muy fácil de obtener cuando se dispone de las rutinas adecuadas para calcularlo, como se obtiene de la diferencia entre el máximo y el mínimo elemento, sólo se debe llamar a estas rutinas y restar los resultados. También la codificación es muy simple y no debería representar problemas al lector.

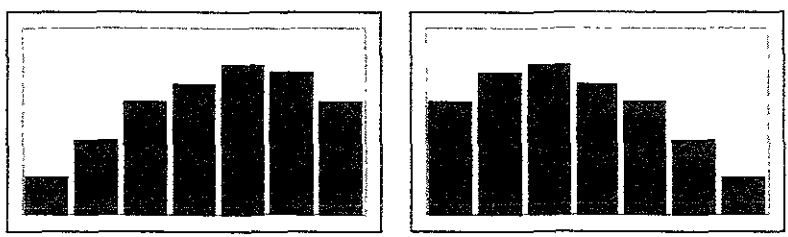
2.2.4. El Sesgo

En términos prácticos y siguiendo el uso común de la palabra, el Sesgo de una distribución se puede ilustrar como la falta de simetría en la gráfica de los datos que se desea analizar, es decir, cuando se tiene una gráfica de los datos en la que al partir a la mitad (sobre el eje x) la gráfica, se tienen como resultado dos partes idénticas, es decir, el 50% de los datos observados está justo del punto medio de la distribución hacia atrás y el otro 50% hacia adelante. Un caso clásico de este tipo de distribución es la distribución normal, la cual forma la famosa campana de Gauss al graficarse.



Esta es la campana de Gauss y esta es una distribución simétrica, si se desea partir a la mitad verticalmente, se puede ver fácilmente que la distribución formaría dos parte iguales.

El sesgo indica hacia dónde están los datos sesgados o asimétricos, y el concepto es sencillo de interpretar pues cuando la mayor parte de los datos observados está a la derecha del punto máximo de la distribución, se dice que el sesgo es a la derecha o que la distribución tiene asimetría positiva y viceversa. Con este dato se puede hacer una estimación mas precisa al respecto de los datos, pues ahora se tiene un dato acerca de dónde se aglomeran los datos, pudiendo tomar decisiones mas precisas con respecto a un fenómeno.



En las gráficas anteriores se pueden ver los tipos de asimetría, en el lado izquierdo se presenta un grupo de datos con asimetría negativa y la gráfica del lado derecho presenta asimetría positiva.

Para el cálculo de este factor en Delphi, se utilizó el método de los momentos centrales, es decir, se calculo el tercer momento para determinar el coeficiente de simetría, la forma teórica de este momento es:

$$\mu_3 = E(X-\mu)^3$$

Cuando para resolver el cubo de los elementos contra la media, se utiliza el coeficiente de Newton, entonces se puede llegar a la siguiente codificación:

```
    function Sesgo : Double; register;
    {Sesgo de los datos, para calcularlo se obtendrá el tercer
momento central}
    var
       i : Longint;
       m,v : Double;
    begin
    try
          result := 0;
          m := media;
          v := sqrt(Varianza(1));
          i := 0;
          while i < xi.count do
          begin
             result := result + pow((StrtoFloat(xi[i])-m)/v, 3.0);
             inc(i);
          end;
          result := result * xi.count / ((xi.count-1)*(xi.count-
2));
       except
          result := -1;
       end;
    end;
```

En este caso se hace una diferencia de cada uno de los elementos de la lista contra la media, la cual se calcula una sola vez igual que la varianza, pues si los datos son varios, la rutina podría tomar mas tiempo de ejecución del necesario. Después simplemente se multiplica por los factoriales del coeficiente mencionado para obtener el resultado esperado. Se puede notar que para calcular la raíz cuadrada de la varianza, se utilizó la función *sqrt()* que es una rutina con la que se cuenta desde el lenguaje Pascal.

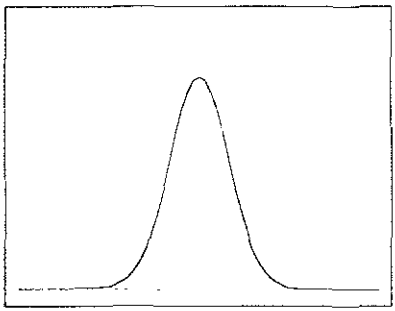
2.2.5. La Curtosis

Este concepto no es precisamente intuitivo como la mayoría de los anteriores pues la palabra no es muy común, sin embargo lo que significa es muy sencillo de comprender, la curtosis indica que tan puntiaguda es una distribución, el caso mas sencillo de mostrar es en una distribución insesgada pues en ella se puede ver al graficar que pasa exactamente con los datos. Los tipos diferentes de curtosis son los siguientes:

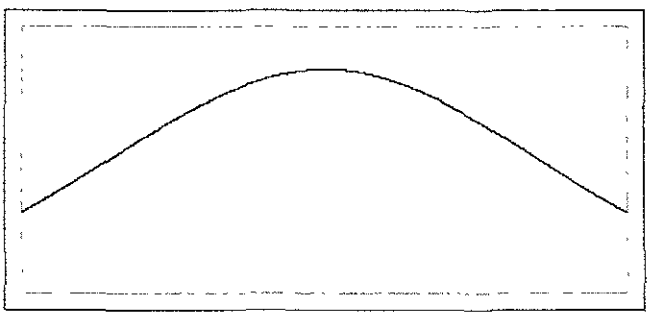
Curva Leptocúrtica, la palabra proviene del griego *lepto* que significa esbelto, por lo que la gráfica es muy puntiaguda; en este caso los datos son muy homogéneos pues la gran mayoría tiende a estar agrupado en un solo valor, con esto se consigue una distribución bastante puntiaguda.

Este dato puede ser muy útil cuando se desea decidir entre dos tipos de resultados y la mejor opción es en la que se tienen datos parecidos y bien agrupados; si se supone el caso de una distribución normal con propiedad leptocúrtica se tiene que el valor que se obtenga

como media de la muestra puede tener una fuerte tendencia a la media poblacional pues la gran mayoría de los datos estarían cerca de ella. Para algunos casos es muy importante tener los datos "juntos" ya que esto permite tomar decisiones mas precisas. Un ejemplo de este tipo de curva es el siguiente:



Curva Platicúrtica, cuyo origen también es griego pero su significado es plano o ancho, en este caso se tiene exactamente lo contrario a las curvas con características leptocúrticas pues si se supone, también, una distribución normal pero con propiedad platicúrtica se tiene que los datos están muy separados y mas bien heterogéneos con respecto a la media de la muestra. Para este caso no se puede suponer que la media muestral tienda a la media poblacional pues de operar los datos se obtendrán valores que mas bien tienden a otros valores. Existen casos en los que este tipo de distribuciones pueden resultar mejores opciones, sin embargo si se desea obtener procesos que estén bajo control, la propiedad platicúrtica es una fuerte indicación de que se está errando el camino deseado. La gráfica de una curva de este tipo es fácil suponerla, así la gráfica tiene la siguiente forma:



Para este caso se utilizó el cálculo del cuarto momento central de una distribución por lo que igual que el caso anterior se obtuvo un resultado mediante el coeficiente de Newton a partir de

$$\mu_4 = E(X-\mu)^4$$

así el código que se encarga de llevar a cabo este cálculo es:

```
    function Curtosis : Double; register;
    {Sesgo de los datos, para calcularlo se obtendrá el tercer
momento central}
    var
       i : Longint;
       m,v : Double;
    begin
    try
          result := 0;
          m := media;
          v := sqrt(Varianza(1));
          i := 0;
          while i < xi.count do
          begin
             result := result + pow((StrtoFloat(xi[i])-m)/v, 4.0);
             inc(i);
          end;
          result := result * (xi.count * (xi.count+1) /
((xi.count-1)*(xi.count-2)*(xi.count-3))) -
                    (3*sqr(xi.count-1)/((xi.count-2)*(xi.count-
3)));
       except
          result := -1;
       end;
    end;
```

En este caso, también se optimiza el uso de rutinas que se repetirán tantas veces como datos haya, por eso se manda llamar una sola vez las rutinas de la media y la varianza y al igual que en el tercer momento, se utiliza la rutina *sqrt()* para determinar la raíz cuadrada.

2.2.6. Los cuatro cuartiles

Este concepto tampoco es muy intuitivo gracias a la palabra con que se identifica, sin embargo el concepto tiene que ver sólo con orden y agrupación de datos, de acuerdo al porcentaje de datos que están entre algunos valores representativos en la muestra y generalmente pueden describir que tanto se "amontonan" los datos entre los valores representativos.

En términos muy simples se habla de cuartiles porque son los valores que parten en cuatro partes iguales a un conjunto de datos, es decir, entre cada uno de los cuartiles existe el 25% de los datos totales de la muestra. En el caso mas simple se tiene la mediana, la filosofía para encontrar la mediana es determinar cuál es el valor que se puede tomar como punto medio de los datos, quiere decir que, antes de la mediana está el 50% de los datos y después de ella el otro 50% del total de la muestra. Así se pueden agrupar los datos de distintas formas, pues si se desea partir en un mayor número de grupos a los datos, se pueden calcular deciles, centiles, etc. y partir los datos en 10, 100 o las partes iguales que se necesiten.

Así el cálculo de cada uno de estos valores depende directamente del número de partes que se desean, pero la forma de hacerlo es como el cálculo de la mediana, sólo que en vez de dividir entre dos, se divide entre el número de grupos que se desean.

Para este caso, sólo se codificó en Delphi el parámetro para determinar el número de cuartil que se desea obtener dentro de los datos, sin embargo, sería muy sencillo poner un parámetro para determinar la cantidad de partes que se desean en los datos. El código quedó de la siguiente forma:

```
function Cuartil(n : Integer) : Double; register;
//Sólo se recibe el número de cuartil que se desea obtener
var i : Longint;
    ampl : Double;
begin
     try
        if xi.count>1 then
        begin
           ampl := (xi.count - 1) / 4;
    {El total de elementos entre el número de grupos que se
quieren}
           result := StrtoInt(xi[trunc(ampl * n)]) +
              (StrtoInt(xi[trunc(ampl * n)+1]) -
StrtoInt(xi[trunc(ampl * n)])) * frac(ampl * n);
        end
        else
           result := StrtoFloat(xi[0]);
     except
        result := -1;
     end;
end;
```

Casi todo el código se ha estado utilizando a lo largo de la programación del *dll*, sin embargo ahora se incluyeron dos funciones de operación de números reales, *Trunc* que se encarga de devolver la parte entera de un número real, y *Frac*, que se encarga de devolver la parte fraccionaria del real. De ahí en fuera sólo se necesita hacer un poco de cálculos con la amplitud de los grupos y el cuartil que se desea.

Capítulo III – Referencias Probabilísticas del *dll*

Durante el siguiente capítulo, se mostrarán todas las rutinas y bases teóricas a partir de las cuales, se generó el código que forma la parte probabilística de la librería, con lo que se harán notar las fórmulas y distribuciones que están codificadas y que le dan vida al *dll* Se mostrará también la forma en cómo pueden ser utilizadas, específicamente, cada una de las rutinas, el orden de los parámetros, el protocolo que utiliza cada rutina, el nombre de las funciones, pues en esta librería no se tiene acceso a las rutinas por un índice. Al mismo tiempo se presentará la parte del código que genera los valores probabilísticos.

3.1. Conceptos básicos y herramientas elementales

Para poder continuar en el entendido que, los términos utilizados son familiares para el lector se hará una breve introducción a la teoría probabilística y después se hará el desglose por funciones.

3.1.1. Eventos

Un evento, en teoría de probabilidad, es todo aquello que ocurre de manera provocada o no y que se puede observar, medir o reproducir. En la gran mayoría de los casos un evento es el resultado de llevar a cabo algo, por ejemplo:

- Lanzar una moneda
- Pesar un libro
- Tocar a una persona que pueda sentir
- Fabricar un foco

Y como estos, muchos otros casos. Sin embargo lo interesante de un evento, que hace que la probabilidad tome forma, es la clasificación que se le puede dar a los eventos, pues de acuerdo al tipo de resultado, se pueden denominar:

Deterministas: Que son todos los eventos en que el resultado se conoce previamente y en el que las variaciones que se presenten no le importan al experimentador. Los ejemplos mencionados, se puede catalogar como *deterministas* si los resultados que se esperan son:

- La moneda caerá al suelo
- El libro pesará cierta cantidad
- La persona sabrá que ha sido tocada
- Se obtendrá un foco al terminar

No deterministas o aleatorios: Este es el caso opuesto a los anteriores, pues los resultados dependerán de factores que aunque son conocidos por el experimentador no pueden ser manipulados por él. Es decir, se puede conocer un número que represente las veces que se puede obtener un resultado deseado o *éxito*, pero dicho resultado, no puede ser

conocido previamente como en el caso determinista. Los ejemplos pueden ser considerados como no deterministas si se cambian como sigue.

- Caerá águila al caer la moneda
- El libro pesa mas que el peso promedio de los libros de una biblioteca casera
- Se sabrá la fuerza con que se toca a la persona
- Se obtendrá un foco defectuoso

Aunque los resultados en estos casos, no pueden saberse previamente, se pueden tener algunos datos que aporten mas información al respecto, con lo que se pueden tomar decisiones al respecto de ellos, siendo esta, la tarea de la probabilidad.

Es importante notar que para estos eventos se tienen algunas características que deben ser cubiertas, antes de poderse considerar un evento como *aleatorio* o *no determinista*.

a) Es posible repetir el experimento en forma infinita (si se quiere) sin cambiar, en sus fundamentos, las condiciones iniciales
b) Aunque no se puede saber exactamente el resultado particular, se puede describir el conjunto de todos los resultados posibles a ese experimento (al conjunto de todos los posibles resultados se le conoce como *espacio muestral*)
c) Aunque los eventos pueden parecer completamente inaccesibles de análisis, después de muchas repeticiones se presentan patrones que pueden ser expresados en un modelo matemático, de manera que a partir de él se puedan hacer conclusiones válidas acerca del experimento que se está realizando.

3.1.2. Frecuencia relativa y el concepto de probabilidad

Supóngase que se lanzó n veces una moneda y que el evento A sucede cuando se obtiene águila como resultado, y B cuando cae sol. Además considérense nA y nB como el número de veces que ocurrieron los eventos A y B, respectivamente. Entonces, la frecuencia relativa se calcula así:

$$f_A = \frac{nA}{n}$$

Lo que representa una tasa de ocurrencia del evento A, con esto se puede describir lo que pasó en una muestra al repetir n veces un experimento. Las propiedades de la frecuencia relativa son las siguientes:

a) $0 \leq f \leq 1$
b) $f = 1$ si y sólo si A ocurre siempre, en las n repeticiones
c) $f = 0$ si y sólo si A nunca ocurre, en las n repeticiones
d) Si A y B son eventos mutuamente excluyentes (que solo puede ocurrir uno a la vez) y f_{AUB} es la frecuencia relativa del evento AUB, entonces $f_{AUB} = f_A + f_B$.
e) f_A basada en n repeticiones tiende a la probabilidad real de A [P(A)] siempre y cuando n se acerca a infinito.

De este modo, se puede llegar a la generalización del concepto de probabilidad de un evento A [P(A)] a partir del concepto de frecuencia relativa, pero siempre y cuando esta última este estimada para un número infinito de casos o que la relación siempre se conserve a lo largo de todos los experimentos que se puedan repetir durante toda la vida. Si se razona un poco al respecto, se puede identificar que la frecuencia relativa representa la probabilidad de un evento una vez que se ha tomado una muestra entre los eventos de un experimento, lo cual es válido para esa muestra, sin embargo cuando se quiere un valor que sea verosímil no importando el número de veces que se repetirá el experimento, entonces se necesita de un proceso un poco mas complicado para llegar a este valor.

De hecho las propiedades del concepto de probabilidad son iguales que las que debe observar la frecuencia relativa, sólo que en vez del inciso e), se generaliza el inciso d), quedando de la siguiente forma:

d) Si $A_1, A_2, \ldots A_n$ son eventos mutuamente excluyentes, entonces, $P(A_1 \cup A_2 \cup \ldots A_n) = P(A_1) + P(A_2) + \ldots P(A_n)$

3.1.3. Métodos de enumeración

Para determinar la probabilidad de un experimento, es muy importante conocer, al menos en número, el espacio muestral o la cantidad de resultados posibles para dicho experimento Esto en casos de pocas alternativas puede resultar trivial, sin embargo hay casos en los que no se puede determinar éste hasta después de haber hecho un análisis un poco mayor del caso.

El *principio de multiplicación*, parte de la idea de tener dos eventos que pueden ocurrir, el primero (A)de n_1 formas distintas y el segundo (B) de n_2 formas distintas también, supóngase que existe un evento formado por ambos experimentos, de forma que se puede hacer que primero se lleve a cabo el experimento A y después el B y observar los resultados que se tengan. El espacio muestral consta, entonces, de (n_1 x n_2) eventos distintos. Si no sólo son dos eventos los involucrados, sino que se tienen m experimentos diferentes y que estos también pueden ser ejecutados uno por uno hasta que se obtenga un evento formado por los eventos de los m experimentos, entonces se puede calcular el número de eventos "compuestos" diferentes de forma análoga (n_1 x n_2 x ... n_m).

A partir de un concepto similar se puede definir el *factorial* que es cuando se toman como posibles resultados para un evento, la multiplicación de todos los subconjuntos de eventos que un experimento puede tener. La forma para su cálculo es:

$$n! = n(n-1)(n-2)...1$$

Nota. $0! = 1$

Si variamos un poco los supuestos del principio de multiplicación, podemos llegar al *principio de adición*. Este se presenta cuando el experimento A y el B no pueden aplicarse juntos, es decir, si se lleva a cabo A de una de las n_1 formas de hacerse las n_1-1 formas

restantes no pueden llevarse a cabo, ni tampoco las n_2 formas de hacer B. En este caso en vez de multiplicar n_1 y n_2, sólo se suman. Si existen mas de dos experimentos, las opciones para llevarlos a cabo sólo se suman y se obtendrá el número de opciones diferentes que se tienen.

Permutaciones, supóngase que se quiere ordenar un grupo de objetos de todas las formas posibles (nPn), además no importa la forma en cómo se escojan, el orden es un factor que haría diferentes el elegir los mismos elementos dos ocasiones diferentes. Por ejemplo, si se tiene {1,2,3} y se desea agrupar en todos los órdenes posibles, entonces se tienen las siguientes alternativas: 123, 132, 213, 231, 312 y 321. Así la respuesta es sencilla, pero si se tratara de un conjunto mas grande, se puede pensar de la siguiente forma: si se eligieran primero los valores que se presentarán primero, se tienen n opciones diferentes entre las cuales elegir, el segundo elemento puede ser 1 de las n-1 opciones restantes, el tercero 1 de las n-2 restantes ... hasta que al final sólo se puede poner la última opción no elegida. Así, se tiene que el calculo corresponde a la multiplicación siguiente:

$$n(n-1)(n-2)...1$$

Que como ya se observó, es la definición del factorial, así que se puede decir que las permutaciones posibles entre n objetos agrupándolos de n en n, será igual al factorial de n, en fórmula se vería así:

$$nPn = n!$$

Sin embargo si se quiere generalizar este caso, se puede pensar en agrupar los valores de r en r elementos, siempre y cuando $0 \leq r \leq n$. Este proceso sufre una variación, pues en este caso el primer elemento se puede elegir de entre n opciones, el segundo de entre n-1 opciones, sin embargo el r-ésimo elemento puede tener, solamente, $n - (r + 1)$ valores diferentes, con esto el factorial se cambia y obtiene la siguiente formula:

$$_nP_r = \frac{n!}{(n-r)!}$$

Con esto queda generalizado el concepto de la permutación.

Combinaciones. Este es un caso muy parecido a las permutaciones, sólo que ahora el orden en que se ponen los elementos que se eligieron, no importa, es decir aunque la permutación 12 es diferente de la 21, la combinación 12 es igual a la combinación 21. Para este caso, una vez que se eligen como permutación los r elementos, se aplica dividiendo el número de formas en que se pueden permutar los valores r!, para así descartar todas las posibles variaciones de orden en la combinación, así la forma de calcular una combinación es la siguiente:

$$_nC_r = \frac{n!}{r!(n-r)!}$$

Existe un caso mas para las permutaciones y es cuando *no todos los objetos son diferentes*, para este caso, supóngase que se tienen n objetos totales, pero que dentro de ellos se tienen n_1 de la clase 1, n_2 de la clase 2, .. de forma que $n_1 + n_2 + .. n_k = n$. Para este caso la forma en como se debe calcular el número total de permutaciones para los n objetos es:

$$_nP_n = \frac{n!}{n_1!n_2!\cdots n_k!}$$

De esta manera, se tienen las herramientas básicas para poder enumerar la cantidad diferente de casos que pueden existir en un mismo espacio muestral.

3.2. Distribuciones Discretas

Antes de entrar completamente en el terreno de las distribuciones de probabilidad, es necesario hacer algunas observaciones y definiciones previas, pues sin estas se podría caer en confusiones al intentar explicar lo que sigue.

3.2.1. Variable Aleatoria

El concepto de variable aleatoria (v.a.), parte de la necesidad de traducir en términos matemáticos y de funciones lo que se ha visto hasta el momento, pues la definición que se puede dar es: Para un experimento E con espacio muestral S; a la función X que asigna a cada uno de los elementos $s \in S$ un número real X(s), se le llama *variable aleatoria*. Esta v.a. cumple con las condiciones básicas de una función matemática y aunque no siempre importa saber de dónde toma el rango y como se calcula cada resultado de X(s), si es importante saber que cada v.a. puede definir perfectamente un experimento dado.

Un ejemplo de v.a. puede estar dado por: supóngase un caso en el que se lanzan dos monedas y se observan sus resultados, y que la v.a. estará definida por el número de "soles" que caen por cada vez que llevamos a cabo el experimento, entonces:

$S = \{(a,a),(a,s),(s,s)\}$[17]

Entonces:
X(aa) = 0,
X(as) = 1 y
X(ss) = 2

Otro ejemplo, en el que se puede tener una v.a. diferente y que no dependa de una suma o función matemática, puede ser expresada como: al fabricar lápices, se pueden tener dos casos diferentes, *defectuoso* o *no defectuoso*, con esto se podría asociar los casos con valores, haciendo que la v a. tome la forma:

[17] Nótese que los resultados (a,s) y (s,a) son iguales para lo que en el ejemplo interesa medir, por esa razón solo está expresada una de las combinaciones posibles

X("defectuoso") = 0 y
X("no defectuoso") = 1.

Así se pueden hacer muchas conjeturas y operaciones acerca de una v.a., además se suprime buena parte de la notación que se necesitaría para definir cada experimento.

3.2.2. Variables aleatorias discretas

Este tipo de clasificación nace por el tipo de manejo que se debe hacer de cada v.a., dependiendo de los valores que esta pueda tomar, (pues aunque se observó que cada v.a. puede estar definida por un rango arbitrario, también es posible asignar cada observación del espacio muestral como el rango de la v.a.), cuando el número posible de valores de X es finito, o al menos infinito numerable, la v.a. se conoce como *discreta* (v.a.d). Es decir, para este caso los valores posibles de X pueden expresarse en una lista que termina (si el conjunto es finito) o continua hasta el infinito pero los valores están bien determinados.

Ahora, se puede hacer una definición mas, cuando una v a.d., es asociada en cada valor por una función a algún valor de probabilidad, se tiene una *función de probabilidad*, es decir, si se puede asociar un valor $p(x_i)$ para cada $P(X=x_i)$ y $p(x_i)$ cumple:

a) p(xi) > 0 para toda i,

b). $\sum_{i=1}^{\infty} p(x_i) = 1$

entonces, $P(X=x_i)$ es una función de probabilidad; al conjunto de parejas ordenadas $\{(x_i,P(x_i)$, para toda i=1,2,...$\}$, se le conoce como distribución de probabilidad de X.

3.2.3. Distribución Binomial

Definición. Supóngase que se tiene un Experimento E, en el que se puede tener como resultado un éxito con probabilidad p o un fracaso con probabilidad 1 – p; además el experimento se repite n veces de forma independiente, entonces, si se quiere saber la probabilidad de tener x éxitos en un total de n repeticiones, se dice que se tiene una v.a. que sigue una distribución binomial. Nota: A cada una de las repeticiones individuales de E, se les conoce como *ensayos Bernoulli*.

Función de probabilidad. Si se tiene un experimento E, con probabilidad de éxito p y se desea saber la probabilidad de que ocurran k éxitos en n ensayos, ésta se calcula de la siguiente forma:

$$P(X=k)=\binom{n}{k}p^k(1-p)^{n-k}$$

Parámetros:

- n: Número de repeticiones del experimento, su rango es. {1,2,3,.. }
- k: Número de éxitos que se desean en las n repeticiones, s r.e: {0,1,2,. .n}
- p: Probabilidad de éxito del experimento E, su rango es: s.r.e: [0,1], y por lo tanto la probabilidad de fracaso es 1-p y por lo tanto s.r.e. igual al de p

Codificación: El código en Delphi, quedó como a continuación se muestra.

```
    function Binomial(n, x : Integer; p : Real) : Double;
register;
    begin
        try
           result := Combinacion(n,x)*pow(p,x)*pow(1.0-p,n-x);
        except
           result := -1;
        end;
    end;
```

Para los entendidos en Pascal, debe ser fácil describir la función, salvo por la estructura try – except, que lo único que hace es "intentar" ejecutar el código que está entre las dos palabras reservadas y en caso de error la función regresará –1.

Se hacen llamadas a dos funciones extras, la primera es *Combinacion*, que tiene como parámetros n, que es el total de valores de donde se obtendrán las combinaciones y x que dice de cuantos en cuantos se tomarán para obtener cada combinación. La otra es *pow* que recibe p que es la base y x que es el exponente al cual se elevará.

El concepto de los parámetros que se deben enviar al *dll* es: `n`, para determinar el total de repeticiones del experimento; `x`, que es la cantidad de éxitos que se quieren obtener y `p` que es la probabilidad de obtener un éxito.

3.2.4. Distribución Geométrica

Definición. Supóngase un experimento E en el cual el resultado que interesa es sólo cuando un evento determinado ocurre o no. Además, siguiendo con los supuestos de la distribución binomial, se tienen probabilidades de ocurrencia = p y de no ocurrencia = 1- p que permanecen constantes cada vez que el experimento se repite. En este caso, también las repeticiones son independientes unas de otras y las probabilidades permanecen constantes, entonces, si lo que interesa es saber cuantas repeticiones de E se necesitan para que ocurra el primer éxito, entonces se tiene una v.a. que sigue una distribución geométrica.

Función de probabilidad. Si se tiene un experimento E con probabilidad de éxito p y se desea saber la probabilidad de que el primer éxito suceda en la k-ésima repetición, entonces dicha probabilidad se calcula de la siguiente forma.

$$P(X=k)=(1-p)^{k-1}p$$

Parámetros:

- k . Número de repeticiones para que suceda el primer éxito, s r.e· {1,2,3 ..}
- p: Probabilidad de éxito del experimento E, s.r.e: [0,1], y por lo tanto la probabilidad de fracaso es 1-p y por lo tanto s.r.e. igual al de p

Codificación: El código en Delphi, quedó como a continuación se muestra.

```
function Geometrica(x : Integer; p : Real) : Double; register;
begin
     try
        result := pow((1-p),(x-1))*p;
     except
        result := -1;
     end;
end;
```

En este caso, al igual que el anterior, existe una sentencia try – except que permite "cachar" el error, de modo que el *dll* no presente fallas, sino que corra y en caso de haber recibido algún dato erróneo, entonces devuelva un valor ilógico.

En esta función, los parámetros que se envían son **x**, que determina el valor de las repeticiones que se desea hacer para que aparezca el primer éxito y **p** es la probabilidad de que el éxito suceda.

3.2.5. Distribución Binomial negativa o de Pascal

Definición. En caso de que la distribución geométrica se quiera generalizar y lo que interese sean las repeticiones contra la r-ésima ocurrencia del éxito, entonces se debe hablar de la distribución binomial negativa, ya que esta distribución procesa una v.a. que calcula la probabilidad de que hayan sucedido r éxitos en k repeticiones. Como puede resultar fácil la apreciación en caso de que se necesiten saber las repeticiones para que ocurra el primer éxito, se estará cayendo en los supuestos hechos para definir la distribución geométrica, sin embargo se puede llegar a la deducción de esta a partir de la distribución binomial de la siguiente manera.

Para que ocurran r éxitos en k repeticiones, es necesario que hayan ocurrido (r-1) éxitos en las (k-1) repeticiones de E previas, para calcular esta probabilidad se puede recurrir a la distribución binomial haciendo n = k-1 y x = r-1. El desglose quedaría así

$$P(X=r-1)=\binom{k-1}{r-1}p^{r-1}(1-p)^{k-r}$$

Por último y para hacer la distinción entre ambas distribuciones, se debe tomar en cuenta el k-ésimo éxito, pues este dará la diferencia entre ambas Si se toma en cuenta que la probabilidad del último éxito es independiente a lo que haya sucedido en las k-1 repeticiones anteriores y que dicha probabilidad es p, sólo se debe multiplicar en el desarrollo anterior p y así se obtendrá la probabilidad total. Así que ahora se tiene.

Función de probabilidad. Una vez que se incluye la última p al desarrollo, se tiene que para obtener la probabilidad de que ocurran r éxitos en k repeticiones se debe aplicar lo siguiente:

$$P(X=r)=\binom{k-1}{r-1}p^r(1-p)^{k-r}$$

Parámetros. De la que es fácil deducir de que datos se compone el resultado esperado

- r : Total de éxitos deseados después de haber realizado k repeticiones, s.r.e: {1,2,3, ...}
- k : Total de repeticiones necesarias para obtener r éxitos, s.r.e: {r, r+1,r+2,...}
- p: Probabilidad de éxito del experimento E, su rango es: s.r.e: [0,1], y por lo tanto la probabilidad de fracaso es 1-p y por lo tanto s.r.e. igual al de p.

Codificación. Esta distribución fue codificada de la siguiente forma.

```
    function BinomialN(x, n : Integer; p : Real) : Double;
register;
    begin
         try
            result := Combinacion(n-1, x-1)*pow(p,x)*pow(1-p, n-
x);
         except
            result := -1;
         end;
    end;
```

En esta rutina los parámetros que es necesario enviar son x que dice el total de éxitos que se desea obtener, n que menciona el total de repeticiones de E que se harán para obtener los éxitos y p que es la probabilidad de éxito del experimento.

3.2.6. Distribución Hipergeométrica.

En este caso, los supuestos de las distribuciones anteriores sufren algunos cambios, pues en este caso, la probabilidad de éxito no siempre se mantiene constante y por lo tanto, los experimentos no son independientes entre si. Supóngase que se tiene un lote de N artículos, de los cuales r son defectuosos (así que N-r son los no defectuosos), además, son

escogidos n artículos de dicho lote y mientras se va sacando cada elemento, éste no es devuelto a la totalidad de artículos; es lógico pensar que si los n objetos se toman de N, entonces $n \leq N$. Si lo que se desea saber es la probabilidad de obtener k artículos defectuosos de la manera descrita, entonces se tiene una v.a. con distribución hipergeométrica

Para obtener la fórmula a partir de la cual se obtiene la probabilidad, se puede pensar en lo siguiente: Para que se obtengan k artículos defectuosos después de tomar n artículos del lote, es necesario haber tomado los k artículos de los r defectuosos y n-k de los no defectuosos del lote. Por lo tanto:

Función de probabilidad. Para este caso se puede expresar el modelo matemático como sigue:

$$P(X=k)=\frac{\binom{r}{k}\binom{N-r}{n-k}}{\binom{N}{n}}$$

Parámetros: En este caso también se han tomado las variables con las que se definió la distribución por lo que debe ser muy fácil saber a que se refiere cada variable, de modo que:

- N : Total de artículos en el lote, s.r.e: {n, n+1, n+2,.. }
- n : Muestra del lote, por lo tanto $n \leq N$; s.r.e: {k, k+1, k+2, . .}
- k : Total de artículos defectuosos en la muestra, por lo tanto $k \leq n$; s.r.e: {0,1,2,...}
- r : Total de artículos defectuosos en el lote, por lo tanto $r \leq N$; s.r.e: {0,1,2,...}

Codificación. Una vez llevado a cabo la programación de la distribución, esta quedó así:

```
    function HiperGeometrica(x, Nt, n, Nd : Integer) : Double;
register;
    begin
       try
        result := Combinacion(Nd,x)*Combinacion(Nt-Nd,n-x) /
Combinacion(Nt,n);
         except
            result := -1;
         end;
    end;
```

En este caso los parámetros teóricos quedan de la siguiente manera en el *dll*, la **x** es el número de defectuosos que se obtendrán, **Nt** es el tamaño del lote, **n** es la muestra en la que interesa conocer el número de defectuosos y **Nd** es el total de artículos defectuosos dentro del lote.

Es bueno hacer mención de una de las consideraciones que se debe tomar en cuenta cuando se programa en Delphi, el compilador no es sensible a las diferencias entre mayúsculas y minúsculas, es por eso que en la codificación se usa la variable *Nt* para denotar el total de elementos, pues en caso de que se hubiera puesto *N* y la muestra como *n*, Delphi habría mandado error por identificadores duplicados

3.2.7. Distribución de Poisson.

Para definir esta distribución se recurrirá, nuevamente, a la distribución binomial que ya ha sido presentada previamente. Supóngase un experimento E cuyos eventos se presentan en intervalos (pueden ser de tiempo, de espacio o cualquier otra cosa que se pueda dividir en intervalos), si se plantea:

P(ocurrencia del evento en un intervalo) = p,
P(No ocurrencia del evento en un intervalo) = 1-p y
P(Mas de una ocurrencia en un intervalo) = 0.

Entonces, los intervalos que se necesitan son tan pequeños que sólo un evento puede ocurrir en ellos, pero son lo suficientemente grandes que la probabilidad de que ocurra dicho evento es mayor a cero. En este caso se tienen las bases para utilizar una distribución binomial sólo que para determinar n se necesita saber el tamaño del intervalo y por lo tanto la probabilidad de que el evento ocurra también se desconoce.

La v.a. que bajo estos supuestos calcule la probabilidad de tener k éxitos en un intervalo determinado, se dice que tiene una distribución de Poisson.

Función de probabilidad. Para que siga siendo válido el desarrollo de la binomial, supóngase $\lambda = np$, que se desea saber la probabilidad de obtener k éxitos y además, póngase a la distribución binomial en un límite cuando $n \to \infty$. De este modo se puede hacer un desglose como el siguiente:

$$= \lim_{n\to\infty} \binom{n}{k} p^k (1-p)^{n-k}$$

$$= \lim_{n\to\infty} \frac{n(n-1)(n-2)\cdots}{k!} \left(\lambda/n\right)^k \left(1-\lambda/n\right)^{n-k}$$

$$= \lim_{n\to\infty} \frac{\lambda^k}{k!} \left(1-\frac{\lambda}{n}\right)^n \frac{n(n-1)(n-2)\cdots}{n^k} \left(1-\frac{\lambda}{n}\right)^{-k}$$

$$= \frac{\lambda^k}{k!} \lim_{n\to\infty} \left(1-\frac{\lambda}{n}\right)^n \left(1-\frac{\lambda}{n}\right)^{-k} \left(1-\frac{1}{n}\right)\left(1-\frac{2}{n}\right)\cdots$$

$$si \lim_{n\to\infty} \left(1-\frac{\lambda}{n}\right)^n = e^{-\lambda} \Rightarrow$$

$$= \frac{\lambda^k}{k!} e^{-\lambda}$$

Parámetros. En este caso se recurre a un parámetro que si bien puede resultar confuso, nace de los parámetros básicos de la distribución binomial n y p.

- λ : Promedio de ocurrencias de éxito en un intervalo, esta nace de la multiplicación de n que es el total de repeticiones y p que es la probabilidad de que ocurra un éxito. De hecho este parámetro determina una tasa de ocurrencia, por lo que puede ser calculada u obtenida directamente.
- k : Es el total de éxitos deseados, s.r.e: {0,1,2,...}
- *e* : Que es la base de los logaritmos neperianos = 2.7182

Codificación. Al transcribir a Delphi la función quedó:

```
function Poisson(x : Integer; L : Real) : Double; register;
begin
     try
        result := pow(L,x)/Factorial(x)*exp(-L);
     except
        result := -1;
     end;
end;
```

Para este caso, sólo se deben pasar el número de éxitos que se desean obtener (x) y el promedio de éxitos por intervalo definido (L), con esto quedará lista la función para ejecutarse.

3.3. Distribuciones Continuas

La necesidad de utilizar rangos de valores finitos o infinitos como posibles valores de algún experimento, da lugar a este tipo de distribuciones; para este caso, se idealiza un concepto matemático que permite generalizar la aparición de un valor dentro de un número

infinito de valores incluidos en un rango, por ejemplo, si se desea conocer la variación que se tiene al generar las dosis de cierto medicamento que se le administrará a un grupo de pacientes, lo mas razonable es esperar diferencias muy sutiles en fragmentos de miligramo, lo que haría casi imposible calcular una probabilidad asociada a un punto tan pequeño, así los problemas se facilitan al adoptar probabilidades de obtener variaciones que correspondan a intervalos que convengan al interesado.

3.3.1. Variables aleatorias continuas

Supóngase el caso de X, una v.a. cuyos valores que se pueden adoptar pueden o interesan medirse en grupos, por ejemplo: todos los valores contenidos en $0 \leq x \leq 1$, de aquí los valores pueden ser {0, 0.01, 0.02, ... 0.99, 1.0}. A cada valor de estos se le puede asignar un valor no negativo que además al sumar cada uno se obtenga un 1 como total. Al verlo de esta manera, se tienen 101 valores posibles para la v.a. pero si generaliza y no se especifican los valores de modo que hablar del i-ésimo elemento sea ilógico, se tendría que la probabilidad para un valor puntual sería necesariamente 0, de modo que es necesario hacer referencias a intervalos de la v.a. de interés.

A este tipo de v.a. se les conoce como variables aleatorias continuas (v.a.c.) y para este tipo de variables, la función de probabilidad se conoce como función de densidad de probabilidad (f.d.p) que satisface con las condiciones siguientes.

$a). f(x) \geq 0, \forall x,$

$b). \int_{-\infty}^{\infty} f(x)dx = 1$

$c). \forall (a,b), donde -\infty < a < b < \infty \rightarrow P(a \leq x \leq b) = \int_{a}^{b} f(x)dx$

Así es como se definen las v.a.c. y sus f.d.p., y al igual que las discretas, pueden comportarse de distintas maneras lo que hace necesario el desglose de estas distribuciones de la siguiente manera.

3.3.2. La distribución Normal

Esta distribución puede que sea la mas importante de todas, pues de la forma en como se presentan sus resultados se comportan muchos experimentos cotidianos, de hecho cuando la distribución que sigue un experimento no se conoce, se utiliza esta para aproximar sus valores. Esta distribución tiene la forma de una campana, teniendo en los extremos superior e inferior de la escala, la probabilidad mas pequeña de ocurrencia y en el centro la mayor cantidad de datos posibles. De esta forma se comportan, por ejemplo, las calificaciones de un grupo escolar de rendimiento promedio, las velocidades que pueden alcanzar todas las motocicletas que se pueden conseguir en el mercado, los precios de los artículos que se pueden comprar en un supermercado, la habilidad intelectual de cualquier mexicano, etc. así como esos hay muchos casos en los que en los extremos hay muy pocas

observaciones. Por esto es que la distribución normal tiene grandes aplicaciones e importancia

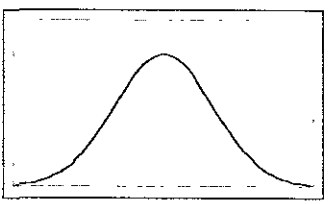
Función de densidad La f.d.p. queda definida como sigue

$$f(x)=\frac{1}{\sqrt{2\pi}\sigma}e^{\left(-\frac{1}{2}\left[\frac{x-\mu}{\sigma}\right]^2\right)},-\infty<x<\infty$$

Parámetros. En este caso los parámetros quedan expresados de la siguiente forma:

- μ : Que es el valor promedio de la distribución, alrededor del cual se agrupan la mayor cantidad de valores, s.r.e. $(-\infty, \infty)$[18].
- σ : Que se refiere es la desviación estándar de la distribución, en múltiplos de esta se medirán, preferentemente, los rangos alrededor de la media. s.r.e. $\sigma > 0$
- e : Que es la base de los logaritmos neperianos = 2.7182
- π : Que es el número 3.1416, aproximadamente.

Gráfica. La gráfica que describe a la distribución normal es de la siguiente forma:



Una forma simplificada de hacer referencia a la distribución normal o *gaussiana* es mediante una N, especificando los parámetros entre paréntesis, por ejemplo N(0,1) significa una distribución normal con media 0 y desviación estándar 1; Estos parámetros son los que comúnmente se utilizan en las tablas de probabilidad ya que es posible cambiar la escala en base a la denominada *notación z*, que lo que hace es, simplemente, cambiar los puntos de referencia de los valores que interesan a la distribución pues en base a cualquier media diferente de cero y a una desviación estándar diferente de 1, se pueden "transportar" valores de una escala a otra, la forma mas simple de hacerlo es:

$$z=\frac{x-\overline{\mu}}{\sigma}$$

Codificación. Para esta distribución el código en Delphi quedo como se muestra a continuación.

```
function Normal(x : Real) : Double; register;
//Se presuponen m=0 y s=1, i.e.: N(0,1)
begin
   try
```

[18] Para mayor información acerca de la media y la desviación estándar se puede referir al siguiente capítulo del proyecto o a libros básicos de estadística.

```
        result  :=  (1/pow(2*Pi,1/2))*Integra('exp((-1/2)*x^2)',  -
4, x);
      except
         result := -1;
      end;
   end;
```

Para encontrar la probabilidad, según la distribución normal, del intervalo que va desde $-\infty$ hasta x, se utilizaron las funciones y simplificaciones siguientes: El único valor que la rutina recibe es x, que es el número (en notación z) hasta el cual se debe calcular la probabilidad, ya que se presuponen para este caso una $\mu = 0$ y $\sigma = 1$. Además se utiliza una constante definida por el usuario de nombre Pi, que equivale al π de la f.d.p. y se utilizan dos funciones extras, que se estarán utilizando en las demás distribuciones continuas, por un lado *Pow* que calcula en base a exponenciales y logaritmos el valor del primer parámetro elevado al segundo; y la otra función *Integra*[19] que recibe una función en la notación expresada y le calcula el valor de la integral desde el segundo parámetro[20] hasta el tercero.

Nota: El código de ambas rutinas se puede encontrar en el anexo donde se presenta, además, todo el código del *dll* y de los ejemplos que lo utilizan.

3.3.3. La distribución Exponencial

Esta distribución tiene como característica particular que solo toma valores positivos y es por esto que su principal uso es la de trabajar con tiempos, es decir, se puede utilizar para calcular tiempos de espera (o probabilidades de esperar determinado tiempo) a partir del tiempo promedio de una fila, por ejemplo. Es muy utilizada también en la teoría de confiabilidad para determinar el tiempo en que se presentará la primer falla de algún artículo a partir de una tasa de fallas constante en un tiempo determinado. Con esta distribución se pueden simular los tiempos de ocurrencia en experimentos que siguen una distribución poisson, pues funcionan a partir de una tasa de ocurrencia en un espacio determinado. Hay que notar que para que una distribución sea del tipo Exponencial, se debe tener una tasa constante de éxitos.

Función de densidad. La f.d.p. está determinada por:

$$f(x) = \alpha e^{-\alpha}, \forall x > 0$$

En donde α es la tasa de ocurrencia del error.

---

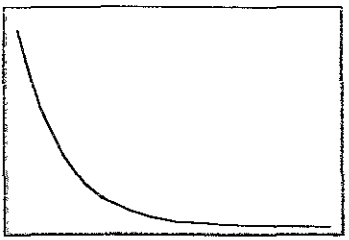
[19] La función integra utiliza rutinas de un componente que conseguido en internet en la página www.xmission.com/~renates/delphi.html, puesto ahí por su autor Renate Schaaf, con dirección electrónica: Internet: renates@xmission.com schaaf@math.usu.edu para uso no lucrativo o educacional.

[20] Se tomó –4 como el valor de $-\infty$ pues antes de este número existen menos del 0.01% de los casos que la distribución puede modelar.

Parámetros. Para alimentar esta función se deben conocer los siguientes valores.

- α : Que define una razón de fallas en un intervalo determinado; s.r.e. $\alpha>0$
- e Que es la base de los logaritmos neperianos = 2 7182

Gráfica. La gráfica que describe a la distribución exponencial es de la siguiente forma.



Codificación. El código de la *dll* que se encarga de calcular la distribución exponencial es el siguiente.

```
function Exponencial(x, a : Real) : Double; register;
begin
     try
        result                    :=                    a*Integra('exp(-
1*'+floattostr(a)+'*x)',0,x);
     except
        result := -1;
     end;
end;
```

Para que el *dll* funcione sólo hay que enviar el valor de `x` que es la cota superior de la integral de la f.d p. y `a` que es el valor de la tasa de éxitos que se tiene para la distribución.

3.3.4. La distribución Gamma

Antes de expresar los usos y definiciones de esta distribución, es necesario definir la función Gamma que es un mecanismo matemático ampliamente utilizado y que además tiene gran utilidad en muchos otros campos.

La función Gamma está definida como sigue:

$$\Gamma(p) = \int_0^\infty x^{p-1} e^{-x} dx, \forall p > 0$$

Si esta expresión se integra (por partes puede resolverse fácilmente) se obtiene una expresión que sin duda resultará familiar, pues cuando p es un entero positivo, se puede llegar a la conclusión que

$$\Gamma(p) = (n-1)!$$

Sin embargo cuando p no es un entero positivo, como se necesita para la generalización anterior, entonces se debe hacer la integral poniendo la cota superior en un límite que tiende al infinito, así se puede decir que la función gamma es una generalización del cálculo del factorial.

Definición. Cuando se trata de definir la distribución gamma, se debe pensar en una suma de r distribuciones exponenciales cuyo parámetro es α, si se quiere saber la distribución de probabilidad que modela dicha suma, entonces la respuesta puede ser encontrada en esta distribución que también es conocida como distribución *Erlang*. Esta distribución tiene múltiples usos en el modelado de fenómenos, pues dependiendo de los parámetros, la función puede tomar formas muy diversas, lo que la hace muy versátil cuando se trata de encontrar funciones que se apeguen lo mas posible a los datos que se han observado de cierto experimento.

Función de densidad. La función que describe la mencionada suma de variables está descrita a continuación.

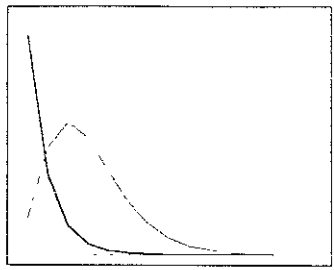
$$f(x) = \frac{\alpha}{\Gamma(r)}(\alpha x)^{r-1}e^{-\alpha x}, \forall x > 0$$

Si se sustituye en r (que es la cantidad de v.a. exponenciales que se suman) el valor de 1, se tiene como resultado la distribución exponencial, de modo que se puede ver a la exponencial como un caso de la distribución gamma, sin embargo sin la primera, esta última no podría ser explicada de una forma tan sencilla.

Parámetros. Los valores ya fueron expresados, sin embargo una definición mas puntual puede ser la siguiente:

- α : Que define una razón de fallas en un intervalo determinado; s.r.e. $\alpha>0$ Este valor es el mismo parámetro que para la distribución exponencial.
- r : Que es el parámetro que le da la forma a la distribución, s.r.e $r>0$
- e : Que es la base de los logaritmos neperianos = 2.7182
- Γ : Que es la función gamma que se definió antes que la distribución

Gráfica. La gráfica que describe a la distribución gamma, puede tener varios perfiles, como se mencionó antes, pero se las formas mas representativas que se pueden obtener son las siguientes:



Codificación. El código de la *dll* que se encarga de calcular la distribución gamma es el siguiente.

```
function Gamma(x, a, n : Real) : double; register;
begin
     try
if alfa <> 1 then
        result := (1/pow(beta,alfa) * G(alfa)) *
Integra('(x)^('+floattostr(alfa)+'-1)*exp(-
x/'+floattostr(beta)+')',0,x)
        else
        result := (1/pow(beta,alfa)*G(alfa))*Integra('exp(-
x/'+floattostr(beta)+')',0,x)
        except
        result := -1;
        end;
end;
```

Para este caso, se necesita enviar la **x** que es hasta donde se evaluará la integral de la f.d.p., el valor que indica la tasa de éxitos para las exponenciales y que está determinado por **a** y por último, **n** que es el parámetro de forma de la distribución. En este caso también se utiliza la función *G(x)* que se encarga de calcular el valor de la función gamma para x.

3.3.5. La distribución Beta

La importancia de la distribución Beta radica, en la propiedad que tiene para generar varias formas distintas, puede ser utilizada en muestras pequeñas y en casos donde se necesita calcular límites de tolerancia donde la hipótesis de una distribución normal no necesariamente se cumple. Esto hace que esta distribución sea una herramienta muy especializada, pues sólo en casos muy especiales se utiliza, pero también muy poderosa pues con las distribuciones comunes no se podría hacer lo que esta puede conseguir.

Función de densidad. Para describir la v.a. con distribución Beta se utiliza un par de distribuciones Gamma de la siguiente forma:

$$x = \frac{x_1}{(x_1 + x_2)}$$

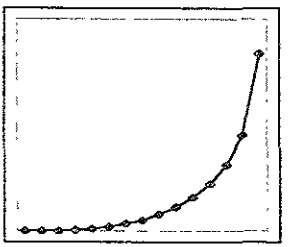
Donde x_1 toma la primer distribución Gamma con parámetros α_1 y r_1 y x_2, por supuesto, es la segunda Gamma con parámetros α_2 y r_2. De aquí nace la f.d.p. como sigue.

$$f(x, r_1, r_2) = \begin{cases} \frac{\Gamma(r_1 + r_2)}{\Gamma(r_1)\Gamma(r_2)}, \forall 0 < x < 1 \\ 0, e.o.c \end{cases}$$

Parámetros. Para este caso se dividen de la siguiente forma

- r_1 : Que es el parámetro de forma de la primer distribución Gamma; s.r.e. $r_1 > 0$
- r_2 : Que funciona igual, pero para la segunda Gamma; s.r.e. $r_2 > 0$
- Γ : Que es la función Gamma ya mencionada antes.

Gráfica. La gráfica que describe a la distribución Beta, puede tener varios perfiles, como se mencionó antes, y como es de esperarse, dependiendo de los parámetros será la forma que adopte la distribución, así que a continuación se presentan las combinaciones de parámetros que dan las formas características.



Codificación. El código de la *dll* que se encarga de calcular la distribución Beta es el siguiente.

```
    function Beta(x, B1, B2 : Real) : Double; register;
    //Para 0<x<1 y B1 y B2 > 0
    begin
         try
    if (B1<>1) and (B2<>1) then
            result := G(B1 + B2) / (G(B1) * G(B2)) * Integra
('x^('+floattostr(B1)+'-1)*(1-x)^('+floattostr(B2)+'-1)',0,x)
         else if (B2<>1) then
            result := G(B1 + B2)/(G(B1)*G(B2))*Integra('(1-x)^('+
floattostr(B2)+'-1)',0,x)
         else if (B1<>1) then
            result := G(B1 + B2)/(G(B1)*G(B2)) *Integra ('x^('
+floattostr(B1)+'-1)',0,x)
```

```
        else
            result := G(B1 + B2)/(G(B1)*G(B2))*Integra('1',0,x);
        except
            result := -1;
        end;
end;
```

Se puede notar que se trata, casi, de un caso generalizado de la versión Gamma pues la forma de evaluarla es muy parecida, los cambios radican en que ahora no se utilizan exponenciales para el cálculo de valores. Para que esta función "corra", se necesita enviar la **x** que dice, en el intervalo de 0 a 1, hasta donde se evaluará la integral. Además es necesario conocer **B1** y **B2** que son los parámetros de forma de la nueva distribución.

3.3.6. La distribución Triangular

Este es un caso de distribución de probabilidad en que los eventos que se conocen son pocos, sin embargo se puede hacer una rápida referencia de la forma en como se comportan dichos eventos aportando los valores *pesimista*, *optimista* y *verosímil*. Con esto se puede formar una distribución cuya probabilidad máxima corresponde al dato verosímil, y los extremos son los valores óptimo y pésimo esperados. Así se tiene la gráfica en forma de triángulo[21].

Función de densidad. Dado que el área está definida por dos rectas que nacen de los valores extremos y que se cruzan en el punto mas verosímil, la función estará expresada en dos partes como a continuación se presenta:

$$f(x)=\begin{cases}\dfrac{2(x-O)}{(v-o)(p-o)}, \forall o<x<v \\ \dfrac{2(p-x)}{(p-v)(p-o)}, \forall v<x<p\end{cases}$$

Parámetros. Los parámetros de este caso, pueden deducirse muy fácilmente a partir de la definición, pues este caso de distribución sólo utiliza la lógica como fundamento teórico, así que los parámetros quedan de la siguiente forma.

- o : Que es el valor óptimo esperado del experimento
- p : Que es el valor pesimista del experimento
- v : Que es el valor verosímil del caso

Para este caso, el rango de las variables puede ser cualquier número dentro de los números reales, pero deben cubrir con que o < v < p. Además se tiene la altura del triangulo

[21] Es importante notar, que aunque esta distribución requiere sólo tres datos para funcionar adecuadamente, puede ser usada en casos complicados en los que no se desea tener mayor complicación para obtener resultados funcionales, es decir, aunque se pueden utilizar otras distribuciones de probabilidad mas complicadas en ocasiones la distribución triangular puede ser preferida para simplificar el experimento.

con un valor igual a 2 / (p - o), que resulta de despejar de la fórmula para calcular el área de un triangulo.

Codificación. El código del *dll* quedó como sigue:

```
    function Triangular(x, o, l, p : Real) : Double; register;
    //Se hace el algoritmo desde aquí, pues la función está
definida en dos partes diferentes
    var
       i      : Integer;
       x0, dx, y, sum1, sum2, xi, xn  : Real;
    begin
    try
            Application.CreateForm(TFormuladlg, Formuladlg);
            dx := (x-o) / Iteraciones;
            x0 := o; xi := 0; xn := 0;
            sum1 := 0; sum2 := 0;
            for i:=0 to Iteraciones do
            begin
               if (x0 + i*dx) <= l then
               begin
                  if (o<>0) then
                     Formuladlg.funcion.expression := '(2*(y-
'+floattostr(o)+'))/(('+floattostr(l-o)+')*('+floattostr(p-o)+'))'
                  else
                     Formuladlg.funcion.expression :=
'(2*y)/(('+floattostr(l-o)+')*('+floattostr(p-o)+'))';
               end
               else
               begin
                  if p<>0 then
                     Formuladlg.funcion.expression :=
'(2*('+floattostr(p)+'-y))/(('+floattostr(p-l)+')*('+floattostr(p-
o)+'))'
                  else
                     Formuladlg.funcion.expression := '(-
2*y))/(('+floattostr(p-l)+')*('+floattostr(p-o)+'))';
               end;
               y := x0 + i*dx;
               if (i>0) and (i<iteraciones) then
               begin
                  if (i mod 2) <> 0 then
                     sum1 := sum1 +
(Formuladlg.Funcion.TheFunction(0,y,0))
                  else
                     sum2 := sum2 +
(Formuladlg.Funcion.TheFunction(0,y,0));
               end
               else
               begin
                  if i=0 then
```

```
            xi := Formuladlg.Funcion.TheFunction(0,y,0)
         else
            xn := Formuladlg.Funcion.TheFunction(0,y,0);
      end;
   end;
   result := (dx / 3) * (xi + xn + 4*sum1 + 2*sum2);
except
   result := -1;
end;
```

Para este caso se utiliza el mismo algoritmo para integrar las anteriores funciones, la diferencia está en que este caso la función está definida en dos partes, así que en vez de complicar la rutina *Integra*, se codificó aquí mismo el cálculo. La llamada a la función que evalúa una función cualquiera, puede ser utilizada desde otra aplicación llamando a la misma función que se llama en este procedimiento, pues esta hace la interacción con el componente que se encarga de hacerlo.

Anexo 1. El código fuente del *Dll*

A continuación se presenta el código completo y final de las rutinas que conforman la librería, el objetivo de incluirla es que se vea cómo están ligadas todas las rutinas y se presentan y ponen a disposición para que se les hagan las modificaciones que se crean pertinentes o sirvan de formato general para que cada quien pueda codificar sus propias rutinas en base a las definiciones y protocolos que yo utilicé.

En caso de que no se cuente con los fuentes de manera magnética este anexo será muy útil pues se puede generar la misma librería u otra con rutinas diferentes a partir de un ejemplo con comentarios y anotaciones hechos a partir del momento en que se codificó el proyecto. Esto no solo sirve para el caso en que no se cuente con los archivos que generan el dll, sino que además se puede ir viendo el proceso de codificación de manera mas clara y adecuada para aprendizaje.

El archivo de proyecto.

Como se menciona a lo largo del trabajo, el compilador de Delphi funciona por proyectos, es decir, existe un programa central que administra y da los pasos necesarios para que cada unidad o programa específico se ejecute cuando se tiene que ejecutar, en este caso el archivo con extensión .DPR tiene lo siguiente:

```
library Pyetool;

{ Important note about DLL memory management: ShareMem must be
the
     first unit in your library's USES clause AND your project's
(select View-Project Source) USES clause if your DLL exports any
procedures or  functions that pass strings as parameters or
function results. This  applies to all strings passed to and from
your DLL--even those that  are nested in records and classes.
ShareMem is the interface unit to  the DELPHIMM.DLL shared memory
manager, which must be deployed along  with your DLL. To avoid
using DELPHIMM.DLL, pass string information  using PChar or
ShortString parameters. }

uses
  ShareMem,
  SysUtils,
  Classes,
  formulas in 'formulas.pas' {Formuladlg};

exports
       Factorial,
       Combinacion,
       Pow,
       Binomial,
       Geometrica,
```

```
        BinomialN,
        HiperGeometrica,
        Poisson,
        Normal,
        Exponencial,
        Gamma,
        GNormal,
        GGamma,
        Integra,
        G,
        Beta,
        Triangular,
        Weibull,
        LimpiaDatos,
        IncluyeDatos,
        EliminaDato,
        OrdenaDatos,
        cuenta,
        sum,
        sum2,
        Media,
        Mediana,
        Moda,
        Varianza,
        Minimo,
        Maximo,
        Rango,
        Sesgo,
        Curtosis,
        Cuartil,
        lista;
begin
     xi := TStringList.Create;
end.
```

Para este caso, se expresa

1. El nombre del *dll*, en este caso Pyetool.dll, que está al lado de la cláusula library.
2. El siguiente texto en inglés está generado automáticamente por Delphi, si cualquiera da la instrucción de crear una nueva librería, este código será agregado en comentario tal y como se presenta aquí.
3. La cláusula Uses le pide, como en Pascal, que se incluyan las librerías declaradas, para este caso, las tres primeras se incluyeron automáticamente para que el *dll* pudiera ser ejecutado y sólo *formulas* fue donde se codificaron las rutinas que le dan vida a esta librería. Es importante notar que si se está generando el código y se le pide a Delphi una nueva unidad, este genera la línea necesaria para que esta unidad se incluya en el proyecto, si la unidad ya existe y se desea utilizar no como librería, sino como código, entonces basta con llamar la opción *Project* del menú y de este tomar la opción *Add to project*. .

4. En la cláusula de Exports se pone el nombre de las rutinas que serán exportadas y que deben estar definidas en alguna parte del código de la librería, para este proyecto hay rutinas que existen y que no están en esta parte por considerarse de uso exclusivo del *dll*, sin embargo, si se desea tener acceso a ellas, se pueden incluir aquí y ser usadas como las otras desde otra aplicación
5 Finalmente se utilizó dentro del código del programa la creación de xi, esta lista de cadenas es un tipo predefinido de Delphi que se utilizó para la administración de los datos estadísticos, está definida dentro de *formulas.pas* y se crea aquí para que cada vez que sea llamada la librería se reserve un espacio de memoria para la lista

El archivo *formulas.pas*

En este caso, esta es la única unidad extra agregada al proyecto, como ella se pueden incluir todas las que sean necesarias y para poder utilizarlas se necesita solo que estén declaradas en la parte del *Uses* del archivo DPR, en esta parte del proyecto está paso a paso lo que cada rutina hace, aquí se pueden agregar rutinas que a cada quien le interesen para que estén en un solo *dll*, el código es el siguiente:

```
{ Nota:
        La instrucción TRY de delphi, intenta ejecutar el
código que está hasta el el Except, en caso de que se cometa un
error, se ejecutará lo que diga el except. La mayor parte de las
funciones contienen esta instrucción para evitar errores mayores
durante el uso de la DLL, en todos los casos, si se encuentra un
error se devolverá cero.}
unit formulas;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls,
Forms, Dialogs,
  StdCtrls, LeerFunciones, Express;

const
   Pi = 3.14159265359;
   Iteraciones = 2048;
   {Debe ser un número par pues es requisito en el modelo de
Simpson 1/3, recomendable una potencia de 2}

type
  TFormuladlg = class(TForm)
    Funcion: TExpress;
    Memo1: TMemo;
  private
    { Private declarations }
  public
    { Public declarations }
```

```
      end;
      {Funciones básicas}
      function Factorial(x : Integer) : Double; register;
      function Combinacion(n, r : Integer) : Double; register;
      function pow(x,y : Double) : Double; register;
      function Integra(f : String; a, b : Real) : Double;
register;
      function G(p : Real) : Double; register;
      {Distribuciones de probabilidad}
      function Binomial(n, x : Integer; p : Real) : Double;
register;
      function Geometrica(x : Integer; p : Real) : Double;
register;
      function BinomialN(x, n : Integer; p : Real) : Double;
register;
      function HiperGeometrica(x, Nt, n, Nd : Integer) : Double;
register;
      function Poisson(x : Integer; L : Real) : Double; register;
      function Normal(x : Real) : Double; register;
      function Exponencial(x, a : Real) : Double; register;
      function Gamma(x, beta, alfa : Real) : double; register;
      function Beta(x, B1, B2 : Real) : Double; register;
      function Triangular(x, o, l, p : Real) : Double; register;
      function Weibull(x, alfa, beta : Real) : Double; register;
      {Generación de Variables Aleatorias}
      function GNormal(m,s : Real) : double; register;
      function GGamma(a : Real;k : Integer) : double; register;
      {Funciones Estadísticas Para generar el espacio en donde
quedarán registrados los datos a los que se les aplicarán las
funciones  se utilizará una estructura de Delphi que permite
manejar listas de cadenas, pero se almacenarán valores primero se
pondrán las rutinas que administren los datos}
      function LimpiaDatos : Boolean; register;
      function IncluyeDatos(x : Double) : Boolean; register;
      function EliminaDato(n : Longint) : Boolean; register;
      function OrdenaDatos : Boolean; register;
      function cuenta : LongInt; register;
      function sum : Double; register;
      function sum2 : Double; register;
      function Media : Double; register;
      function Mediana : Double; register;
      function Moda : String; register;
      function Varianza(n : Longint): Double; register;
      function Minimo : Double; register;
      function Maximo : Double; register;
      function max(yi : TStringlist) : Double; register;
      function Rango : Double; register;
      function Sesgo : Double; register;
      function Curtosis : Double; register;
      function Cuartil(n : Integer) : Double; register;
      function lista : String; register;
    var
```

```
    Formuladlg: TFormuladlg;
    {Xi, es la lista donde quedarán los datos que se reciben
para operar de manera estadística}
    xi : TStringlist;

implementation

{$R *.DFM}

function Factorial(x : Integer) : Double; register;
//Sólo se envía el valor entero del que se desea calcular el
factorial
begin
     if x>=0 then
     begin
        {Cálculo del factorial usando recursividad}
        if x>1 then result := x * factorial(x-1)
        else result := 1;
     end
     else
        {No hay factorial para números negativos}
        result := 0;
end;

function Combinacion(n, r : Integer) : Double; register;
//número total de objetos (n), tomados de r en r
begin
     try
        {Cálculo de una combinación: C(a,b)=a!/(b!(a-b)!)}
        result := (Factorial(n))/(Factorial(r)*(Factorial(n-
r)));
     except
        {en caso de error devuelve -1}
        result := -1;
     end;
end;

function pow(x,y : Double) : Double; register;
//Potencia para x^y
begin
     try
        {Para bases negativas, sólo con exponente entero}
        if x<0 then
        begin
           result := exp(y * ln(abs(x)));
           if (trunc(y) mod 2) = 1 then
              result := result * -1;
        end
        else
           result := exp(y * ln(x));
     except
        result := -1;
```

```
        end;
    end;

    function Integra(f : String; a, b : Real) : Double; register;
    var
       i : Integer;
       inc, sum1, sum2 : Double;
       x0,x : Real;
    begin
        {Rutina para integrar funciones}
        try
           with Formuladlg do
           begin
              //Se crea la pantalla dónde están los componentes
que evaluan las funciones
              Application.CreateForm(TFormuladlg, Formuladlg);
              {Uso del componente para evaluar funciones obtenido
de internet}
              Funcion.Expression := f;
              //Se calcula el incremento en x para cada iteración
              x0 := a;
              result := 0;
              inc := 0;
              if a<=b then
                 inc := (b-a) / Iteraciones
              else
                 result := -1;
              if result <> -1 then
              begin
                 result := 0;
                 sum1 := 0;
                 sum2 := 0;
                 //Se evalua para cada punto e incremento el área
de su rectángulo
                 for i:=1 to Iteraciones-1 do
                 begin
                    {Se hace el llamado a la función, a partir
del cálculo del valor y su incremento}
                    x := x0 + inc;
                    if (i mod 2) <> 0 then
                       sum1 := sum1 + Funcion.TheFunction(x,0,0)
                    else
                       sum2 := sum2 + Funcion.TheFunction(x,0,0);
                    x0 := x0 + inc;
                 end;
                 x := a;
                 result := result + Funcion.TheFunction(x,0,0);
                 x := b;
                 result := result + Funcion.TheFunction(x,0,0);
                 result := (inc/3) * (result + 4*sum1 + 2*sum2);
              end;
              Formuladlg.Free
```

```
            end;
         except
            result := -1;
         end;
    end;

    function G(p : Real) : Double; register;
    begin
         try
            {Para las iteraciones y la función, 50 ya se puede
considerar infinito}
            if p<>1 then
               result := Integra('x^('+floattostr(p-1)+')*exp(-
x)',0,50)
            else
               result := Integra('exp(-x)',0,30)
         except
            result := -1;
         end;
    end;

    function Binomial(n, x : Integer; p : Real) : Double;
register;
    //Probabilidad, según la distribución binomial, de que de n
experimentos, x sean
    //éxitos, si la probabilidad de éxito es p
    begin
         try
            {Cálculo de la probabilidad = C(n,x)(p^x)(1-p^n-x)}
            result := Combinacion(n,x)*pow(p,x)*pow(1.0-p,n-x);
         except
            result := -1;
         end;
    end;

    function Geometrica(x : Integer; p : Real) : Double; register;
    //Probabilidad, de que a la x-ésima repetición del
experimento, obtengamos el primer
    //éxito
    begin
         try
            {Geometrica = (q^y-1)(p)}
            result := pow((1-p),(x-1))*p;
         except
            result := -1;
         end;
    end;

    function BinomialN(x, n : Integer; p : Real) : Double;
register;
    //Probabilidad de que después de n ensayos, se obtengan x
éxitos, si se tiene una
```

```
    //probabilidad p
    begin
         try
            {Binomial negativa = C(n-1, x-1)(p^x)(1-p^n-x)}
            result := Combinacion(n-1, x-1)*pow(p,x)*pow(1-p, n-
x);
         except
            result := -1;
         end;
    end;

    function HiperGeometrica(x, Nt, n, Nd : Integer) : Double;
register;
    //Probabilidad de que, de n artículos tomados de Nt, x sean
éxitos si hay Nd
    //artículos exitosos
    begin
         try
            {Hipergeométrica = C(Nd,x)C(Nt-Nd,n-x)/C(Nt,n)}
            result := Combinacion(Nd,x)*Combinacion(Nt-Nd,n-
x)/Combinacion(Nt,n);
         except
            result := -1;
         end;
    end;

    function Poisson(x : Integer; L : Real) : Double; register;
    begin
         try
            {Poisson = (L^x)/(x!) e^-L}
            result := pow(L,x)/Factorial(x)*exp(-L);
         except
            result := -1;
         end;
    end;

    function Normal(x : Real) : Double; register;
    //Se presuponen m=0 y s=1, i.e.: N(0,1)
    begin
         try
            {Normal = (1/(2Pi)^1/2S) e^((-1/2)((x-m)/s)^2)}
            result := (1/pow(2*Pi,1/2))*Integra('exp((-1/2)*x^2)',
-4, x);
         except
            result := -1;
         end;
    end;

    function Exponencial(x, a : Real) : Double; register;
    begin
         try
            {exponencial = ae^-ax}
```

```
            result := a*Integra('exp(-
1*'+floattostr(a)+'*x)',0,x);
         except
            result := -1;
         end;
     end;

     function Gamma(x, beta, alfa : Real) : double; register;
     //beta es el parámetro de escala y alfa el de forma
     begin
          try
             if alfa <> 1 then
                result :=
(1/pow(beta,alfa)*G(alfa))*Integra('(x)^('+floattostr(alfa)+'-
1)*exp(-x/'+floattostr(beta)+')',0,x)
             else
                result := (1/pow(beta,alfa)*G(alfa))*Integra('exp(-
x/'+floattostr(beta)+')',0,x)
          except
             result := -1;
          end;
     end;

     function Beta(x, B1, B2 : Real) : Double; register;
     //Para 0<x<1 y B1 y B2 > 0
     begin
          try
             if (B1<>1) and (B2<>1) then
                result := G(B1 +
B2)/(G(B1)*G(B2))*Integra('x^('+floattostr(B1)+'-1)*(1-
x)^('+floattostr(B2)+'-1)',0,x)
             else if (B2<>1) then
                result := G(B1 + B2)/(G(B1)*G(B2))*Integra('(1-
x)^('+floattostr(B2)+'-1)',0,x)
             else if (B1<>1) then
                result := G(B1 +
B2)/(G(B1)*G(B2))*Integra('x^('+floattostr(B1)+'-1)',0,x)
             else
                result := G(B1 +
B2)/(G(B1)*G(B2))*Integra('1',0,x);
          except
             result := -1;
          end;
     end;

     function Triangular(x, o, l, p : Real) : Double; register;
     //Se hace el algoritmo desde aquí, pues la función está
definida en dos partes diferentes
     var
        i      : Integer;
        x0, dx, y, sum1, sum2, xi, xn  : Real;
     begin
```

```
        try
           Application.CreateForm(TFormuladlg, Formuladlg);
           dx := (x-o) / Iteraciones;
           x0 := o; xi := 0; xn := 0;
           sum1 := 0; sum2 := 0;
           for i:=0 to Iteraciones do
           begin
              if (x0 + i*dx) <= 1 then
              begin
                 if (o<>0) then
                    Formuladlg.funcion.expression := '(2*(y-
'+floattostr(o)+'))/(('+floattostr(1-o)+')*('+floattostr(p-o)+'))'
                 else
                    Formuladlg.funcion.expression :=
'(2*y)/(('+floattostr(1-o)+')*('+floattostr(p-o)+'))';
              end
              else
              begin
                 if p<>0 then
                    Formuladlg.funcion.expression :=
'(2*('+floattostr(p)+'-y))/(('+floattostr(p-1)+')*('+floattostr(p-
o)+'))'
                 else
                    Formuladlg.funcion.expression := '(-
2*y))/(('+floattostr(p-1)+')*('+floattostr(p-o)+'))';
              end;
              y := x0 + i*dx;
              if (i>0) and (i<iteraciones) then
              begin
                 if (i mod 2) <> 0 then
                    sum1 := sum1 +
(Formuladlg.Funcion.TheFunction(0,y,0))
                 else
                    sum2 := sum2 +
(Formuladlg.Funcion.TheFunction(0,y,0));
              end
              else
              begin
                 if i=0 then
                    xi := Formuladlg.Funcion.TheFunction(0,y,0)
                 else
                    xn := Formuladlg.Funcion.TheFunction(0,y,0);
              end;
           end;
           result := (dx / 3) * (xi + xn + 4*sum1 + 2*sum2);
        except
           result := -1;
        end;
     end;

     function Weibull(x, alfa, beta : Real) : Double; register;
     begin
```

```
        {Hay versiones diferentes para la Weibull en excel y en
el de videgaray, tal vez no se incluya}
        try
           if alfa <> 1.0 then
              result := (alfa / pow(beta,alfa)) *
Integra('(x^('+floattostr(alfa)+'-1)*exp(-((x/'+
                      floattostr(beta)+')^'+floattostr(alfa)+'))',
0, x)
           else
              result := (alfa / pow(beta,alfa)) * Integra('exp(-
((x/'+floattostr(beta)+')^'+
                      floattostr(alfa)+'))', 0, x);
        except
           result := -1;
        end;
    end;

    function GNormal(m,s : Real) : double; register;
    //Generación de normales con media = m y desviación estándar =
s
    var
       i : Byte;
       r : Double;
    begin
        try
           r := 0;
           for i:=1 to 12 do
              r := r + random;
           r := r - 6;
           result := s*r + m;
        except
           result := -1;
        end;
    end;

    function GGamma(a : Real;k : Integer) : double; register;
    //Dónde 1/ka es la media de las k's distribuciones
independientes, con k
    //v.a. diferentes
    var
       i : Integer;
    begin
        try
           result := 0;
           for i:=1 to k do
              result := result + ln(random);
           result := result / (-k*a);
        except
           result := -1;
        end;
    end;
```

```
    function LimpiaDatos : Boolean; register;
    {Prepara la lista de cadenas para subir datos, es decir, la
deja en blanco}
    begin
       try
          xi.Clear;
          result := true;
       except
          result := false;
       end;
    end;

    function IncluyeDatos(x : Double) : Boolean; register;
    {Agrega los datos dándoles formato de cadenas}
    begin
       try
          {Como se esta usando una lista de cadenas, se guardan
con ceros a la izquierda para que al ordenar
          una cadena sea como cuando se hace con números, si no
las ordena sin tomar en cuenta la posición
          decimal de cada número}

xi.add(FormatFloat('00000000000000000000.00000000000000000000',x))
;
          result := true;
       except
          result := false;
       end;
    end;

    function EliminaDato(n : Longint) : Boolean; register;
    {Elimina los n últimos datos de la lista}
    var
       i : Longint;
    begin
       try
          i := 1;
          while i<=n do
          begin
             xi.delete(xi.count - 1);
             inc(i);
          end;
          result := true;
       except
          result := false;
       end;
    end;

    function OrdenaDatos : Boolean; register;
    {Hace que en la lista queden los datos ordenados}
    begin
       try
```

```
        if not xi.sorted then xi.sort;
        result := true;
      except
        result := false;
      end;
    end;

    function cuenta : LongInt; register;
    {Regresa el número de datos que contenga la lista, si se está
familiarizado con el manejo del tipo
     TStringList, se puede evitar llamar a esta función}
    begin
      try
        result := xi.count;
      except
        result := -1;
      end;
    end;

    function sum : Double; register;
    {Regresa la suma de los datos de la lista}
    var
      i : Longint;
    begin
      try
        result := 0;
        i := 0;
        while i < xi.count do
        begin
          result := result + strtofloat(xi[i]);
          inc(i);
        end;
      except
        result := -1;
      end;
    end;

    function sum2 : Double; register;
    {Regresa la suma de los datos al cuadrado de la lista}
    var
      i : Longint;
    begin
      try
        result := 0;
        i := 0;
        while i < xi.count do
        begin
          result := result + pow(strtofloat(xi[i]),2);
          inc(i);
        end;
      except
        result := -1;
```

```
   end;
end;

function Media : Double; register;
{Calcula la media aritmética de los datos de la lista}
begin
   try
      result := sum / cuenta;
   except
      result := -1;
   end;
end;

function Mediana : Double; register;
{Calcula la mediana de los datos de la lista}
begin
   try
      result := cuartil(2);
   except
      result := -1;
   end;
end;

function Moda : String; register;
{Calcula la moda de los datos de la lista}
var
   i, n : LongInt;
   ldato, lcuenta : TStringlist;
   dato : String;
begin
   try
      OrdenaDatos;
      ldato := TStringlist.Create;
      lcuenta := TStringlist.Create;
      ldato.clear;
      lcuenta.clear;
      i := 0;
      n := 0;
      dato := xi[0];
      repeat
         if dato = xi[i] then
            inc(n)
         else
         begin
            ldato.add(dato);
            lcuenta.add(inttostr(n));
            n := 1;
            dato := xi[i];
         end;
         inc(i);
      until  i=xi.Count;
      ldato.add(dato);
```

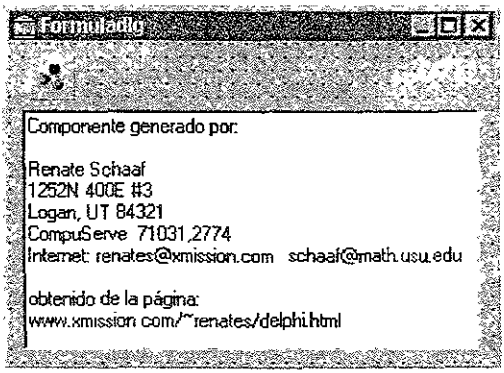
```
            lcuenta.add(inttostr(n));
            n := strtoint(lcuenta[0]);
            for i:=1 to lcuenta.count-1 do
               if n<strtoint(lcuenta[i]) then
n:=strtoint(lcuenta[i]);
            i := 0;
            result := '';
            while i<lcuenta.Count do
            begin
               if n = StrtoInt(lcuenta[i]) then
                  result := result +
FloattoStr(StrtoFloat(ldato[i])) + ',';
               inc(i);
            end;
            ldato.free;
            lcuenta.free;
         except
            result := '';
         end;
      end;

      function Varianza(n : Longint): Double; register;
      {Calcula la varianza de los datos con n grados de libertad}
      begin
         try
            result := ((cuenta*sum2)-pow(sum,2))/(cuenta * (cuenta -
n));
         except
            result := -1;
         end;
      end;

      function Minimo : Double; register;
      {Regresa el mínimo valor de un grupo de datos almacenado en la
lista que pasa como parámetro}
      begin
         try
            OrdenaDatos;
            result := StrtoFloat(xi[0]);
         except
            result := -1;
         end;
      end;

      function Maximo : Double; register;
      {Regresa el máximo valor de un grupo de datos almacenado en la
lista que pasa como parámetro}
      begin
         try
            OrdenaDatos;
            result := StrtoFloat(xi[xi.count - 1]);
         except
```

```
            result := -1;
         end;
      end;

      function Max(yi : TStringlist) : Double; register;
      {Regresa el máximo valor de un grupo de datos almacenado en la
lista que pasa como parámetro}
      begin
         try
            if not yi.sorted then yi.sort;
            result := StrtoFloat(yi[yi.count - 1]);
         except
            result := -1;
         end;
      end;

      function Rango : Double; register;
      {Devuelve el rango de los datos}
      begin
         try
            result := Maximo - Minimo;
         except
            result := -1;
         end;
      end;

      function Sesgo : Double; register;
      {Sesgo de los datos, para calcularlo se obtendrá el tercer
momento central}
      var
         i : Longint;
         m,v : Double;
      begin
         try
            result := 0;
            m := media;
            v := sqrt(Varianza(1));
            i := 0;
            while i < xi.count do
            begin
               result := result + pow((StrtoFloat(xi[i])-m)/v, 3.0);
               inc(i);
            end;
            result := result * xi.count / ((xi.count-1)*(xi.count-
2));
         except
            result := -1;
         end;
      end;

      function Curtosis : Double; register;
```

```
    {Sesgo de los datos, para calcularlo se obtendrá el tercer
momento central}
    var
       i : Longint;
       m,v : Double;
    begin
       try
          result := 0;
          m := media;
          v := sqrt(Varianza(1));
          i := 0;
          while i < xi.count do
          begin
             result := result + pow((StrtoFloat(xi[i])-m)/v, 4.0);
             inc(i);
          end;
          result := result * (xi.count * (xi.count+1) /
((xi.count-1)*(xi.count-2)*(xi.count-3))) -
                    (3*sqr(xi.count-1)/((xi.count-2)*(xi.count-
3)));
       except
          result := -1;
       end;
    end;

    function Cuartil(n : Integer) : Double; register;
    //Sólo se recibe el número de cuartil que se desea obtener
    var
        ampl : Double;
    begin
         try
            OrdenaDatos;
            if xi.count>1 then
            begin
               ampl := (xi.count - 1) / 4; {El total de elementos
entre el número de grupos que se quieren}
               result := StrtoFloat(xi[trunc(ampl * n)]);
               if n<4 then
                  result := result + (StrtoFloat(xi[trunc(ampl *
n)+1]) - StrtoFloat(xi[trunc(ampl * n)])) * frac(ampl * n);
            end
            else
               result := StrtoFloat(xi[0]);
         except
            result := -1;
         end;
    end;

    function lista : String; register;
    var
       i : Longint;
    begin
```

```
        for i:=0 to xi.count - 1 do
           result := result + FloattoStr(StrtoFloat(xi[i])) +
','
    end;

    end.
```

Este código está disponible también como archivo electrónico, pero por si no tiene acceso a él, se presenta tal y como quedó la parte del programa. Existe otra parte del código que es la forma o ventana donde está "dibujado" el componente que hace posible la evaluación de funciones, la forma se ve de la siguiente manera:



Como se puede observar, se creó un campo en que se puso toda la información disponible acerca del autor del componente y la página donde este se encuentra, además de algunos otros también muy útiles, cabe mencionar que el autor del componente no busca lucrar con ellos, sino mas bien, generar un banco de componentes que hagan rutinas matemáticas y que sirvan para proyectos educativos o no lucrativos, si su deseo es contactarlo, la información está disponible.

Este código está a disposición del que lo necesite y si se debe alterar o agregar cosas, cada programador está en la libertad de hacerlo.

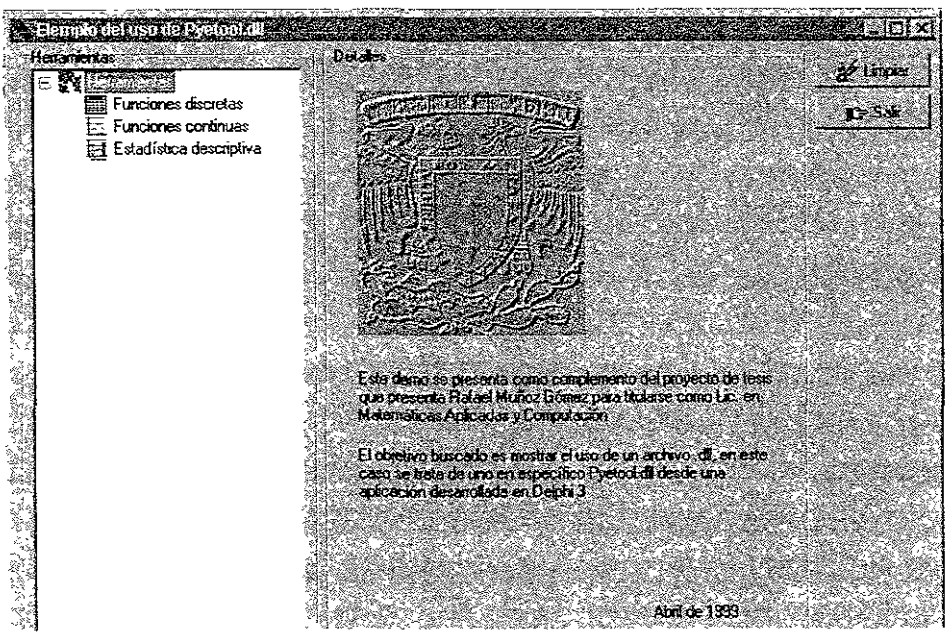
Anexo 2. Una aplicación que use el *Dll* y su código fuente.

Para que una librería de ligado dinámico tenga sentido, es necesario que alguna aplicación la utilice, entonces, se muestra en el presente anexo, la utilería que se desarrolló como ejemplo para mostrar como ligar la librería, como tener acceso a sus rutinas y como liberarla de memoria después. En este caso se presenta la forma de utilizar un dll en Delphi (no importa en que lenguaje se haya creado la librería), para hacer uso de este tipo de librerías en otros lenguajes, habrá que hacer el estudio de las rutinas equivalentes a las que aquí se muestran.

Se presenta además, el funcionamiento de la aplicación de ejemplo, así como la manera en como, desde el fuente, se hace el acceso a los valores, se pasan los parámetros a la librería y se manejan los resultados que la rutina nos devuelve.

El *objetivo del anexo* es mostrar los fuentes de una aplicación en Delphi que usa una librería de ligado dinámico, su interacción con el dll desarrollado y mostrar el funcionamiento de la aplicación de ejemplo.

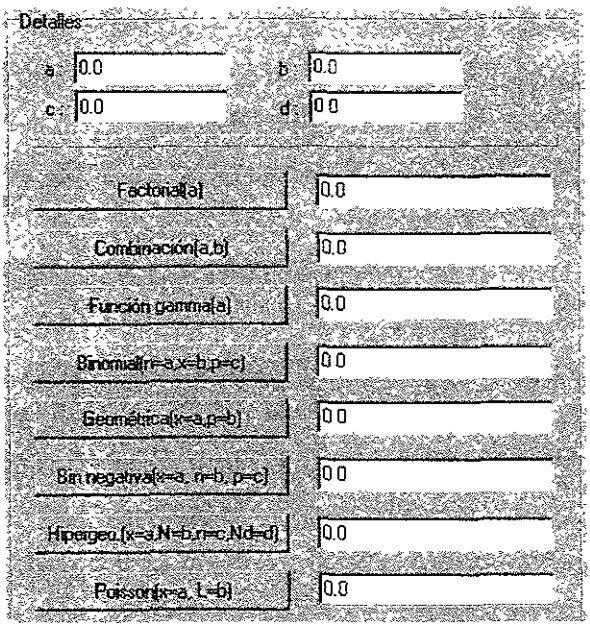
Dentro del proyecto está una aplicación de ejemplo, que puede ser ejecutada individualmente y este se encarga de poner a trabajar las rutinas de la librería, la forma en que se ve y como se usa es la siguiente:



Esta es la primer pantalla, que sirve como presentación para esta aplicación, en el lado izquierdo se tiene una especie de menú modernista, pues las opciones no están como

se acostumbran, mas bien van hacia abajo y la parte de la derecha presentará la pantalla que se necesite de acuerdo a la opción elegida en el menú. Hasta la derecha se encuentran dos botones que estarán disponibles desde cualquier opción del menú y que sirven para borrar los campos donde quedan los resultados de las evaluaciones (Limpiar) y para salir de la aplicación.

La siguiente pantalla se encarga de controlar las rutinas de probabilidad de variables aleatorias discretas, la pantalla se ve como sigue:



En esta pantalla se pueden calcular distintos valores de acuerdo a los parámetros que se pongan en los cuatro cuadros superiores y etiquetados como a, b, c y d. Cada botón de la parte de abajo tiene a su cargo una rutina distinta, por ejemplo, *Factorial(a)* calculará el factorial del valor que se ponga en el recuadro con la etiqueta a, *Combinación(a,b)* calculará la combinación tomando de a en a en un total de b elementos, estos valores, también deberán ponerse en los recuadros de arriba. Así cada botón tiene una rutina específica y necesita que se capturen los valores que el mismo botón indica, de lo contrario se cometerá un error de ejecución. La pantalla de funciones continuas es muy parecida y su funcionamiento es prácticamente el mismo, así que no se mostrará aquí y se ilustrará en seguida la pantalla de estadísticas.

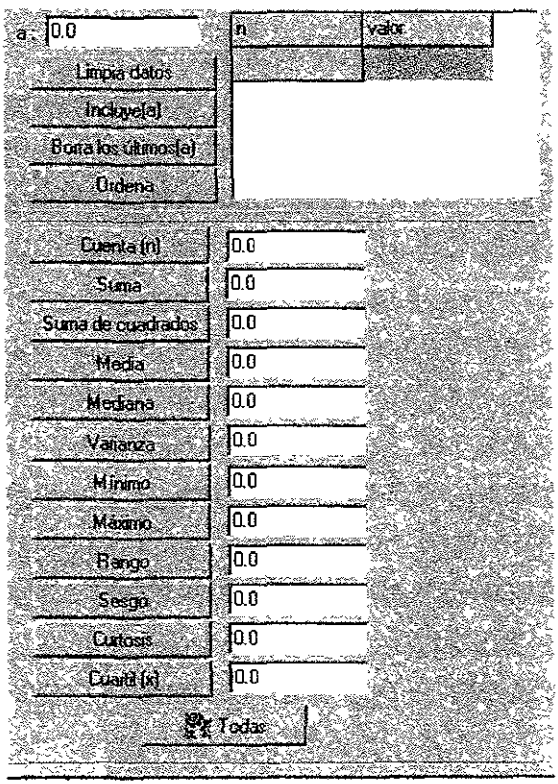
Para esta pantalla se presenta una forma distinta de manejar los parámetros, pues en este caso los parámetros puede ser todo un conjunto completo de datos, el *Dll* segirá almacenando datos en memoria hasta que esta se acabe o por seguridad no le permita seguir almacenando, entonces se tienen al principio las rutinas para la administración de los datos, si se ven en orden se ven de la siguiente forma:

1 Limpia datos, que crea una lista vacía de datos, para empezar a incluir los que sean necesarios, en este caso también está ligada la administración a un visor o grid, que va incluyendo todos los valores que se almacenan, esta es una forma de cortesía pues el *dll* no está preparado para mostrar así los datos, esta es funcionalidad de la pantalla que lo usa

2. Incluye(a), cada vez que se necesite dar de alta un dato en el grid, se debe poner el dato requerido en el recuadro con la etiqueta *a* y debe hacerse clic en esté botón, lo mismo que en el punto anterior, se ve en el grid sin embargo el que realmente importa que lo incluya es la lista de cadenas del *dll* pues de ahí se obtendrán los *resultados esperados*.

3 Borra los últimos(a), si se desea borrar un elemento, esté tendrá que ser del final y el parámetro que se solicita es el número de elementos a borrar, los cuales se irán quitando del último al primero, como en una estructura de pila.

4. Todos los demás botones tienen a su cargo distintas funcionalidades, cada una es clara por la descripción del botón, y funcionan de manera similar a las pantallas anteriores.



Esta es la aplicación a manera de demostración que se incluye en el proyecto sin afán de competir con las herramientas especializadas, lo único que se muestra es un ejemplo de

cómo se puede utilizar una librería dinámica para ser llamada desde una aplicación, cada quien podrá utilizar la librería en sus propios desarrollos y podrá incluir muchas mas facilidades de las que esta aplicación tiene, pero también se pueden usar las rutinas como formato fijo para mandar llamar las rutinas de cualquier *Dll* y así utilizarlas en aplicaciones personales.

El código Fuente

La parte del archivo de proyecto es la siguiente:

```
program Demodll;

uses
  Forms,
  Evalua in 'Evalua.pas' {FrmEjemplos};

{$R *.RES}

begin
  Application.Initialize;
  Application.CreateForm(TFrmEjemplos, FrmEjemplos);
  Application.Run;
end.
```

A diferencia del proyecto de la librería, en este caso se especifica que se generará un programa, sin embargo lo demás es muy parecido al anterior.

En cuanto al código de la pantalla donde se lleva a cabo toda la interfaz con el usuario, el código es el siguiente:

```
    {Para casi todas las rutinas expresadas en este Demo, se
utiliza el acceso implícito a las rutinas del dll, salvo las
rutinas, de la librería, que son utilizadas por otras rutinas
(Integra, por ejemplo), de modo que se puede tener un ejemplo de
como utilizar por ambas definiciones
     Rafael Muñoz Gómez}
    unit Evalua;

    interface

    uses
      Windows, Messages, SysUtils, Classes, Graphics, Controls,
Forms, Dialogs,
      ComCtrls, StdCtrls, ExtCtrls, Buttons, Grids;

    type
      //De la pantalla de funciones discretas
```

```
      TFactorial = function(x : Integer) : Double; register;
      TCombinacion = function(n, r : Integer) : Double; register;
      TFGamma = function(p : Real) : Double; register;
      TBinomial = function(n, x : Integer; p : Real) : Double;
register;
      TGeometrica = function(x : Integer; p : Real) : Double;
register;
      TBinNegativa = function(x, n : Integer; p : Real) : Double;
register;
      THiper = function(x, Nt, n, Nd : Integer) : Double;
register;
      TPoisson = function(x : Integer; L : Real) : Double;
register;
      //De la pantalla de funciones continuas
      TNormal = function(x : Real) : Double; register;
      TExponencial = function(x, a : Real) : Double; register;
      TGamma = function(x, beta, alfa : Real) : double; register;
      TBeta = function(x, B1, B2 : Real) : Double; register;
      TTriangular = function(x, o, l, p : Real) : Double;
register;
      TWeibull = function(x, alfa, beta : Real) : Double;
register;
      //De la pantalla de estadísticas
      TCuenta = function : LongInt; register;
      TDoble = function : Double; register;
      TVarianza = function(n : Longint): Double; register;
      TCuartil = function(n : Integer) : Double; register;

      TFrmEjemplos = class(TForm)
        GbHtas: TGroupBox; {Este componente enmarca al treeview
donde está el desgloce de las herramientas}
        tvMenu: TTreeView; {Este componente tiene enumeradas las
opciones a manera de menú}
        GbDet: TGroupBox;
        NbInterfaz: TNotebook;
        Image1: TImage;
        StaticText1: TStaticText;
        StaticText2: TStaticText;
        StaticText3: TStaticText;
        Label2: TLabel;
        Label4: TLabel;
        Label5: TLabel;
        Label6: TLabel;
        Eda: TEdit;
        Edb: TEdit;
        Edc: TEdit;
        Edd: TEdit;
        Panel1: TPanel;
        bFactorial: TButton;
        bCombinacion: TButton;
        bFGamma: TButton;
        bBinomial: TButton;
```

```
    bGeom: TButton;
    bBinNegativa: TButton;
    bHiper: TButton;
    bPoisson: TButton;
    EdFact: TEdit;
    EdComb: TEdit;
    EdFGamma: TEdit;
    EdBinomial: TEdit;
    EdGeom: TEdit;
    edBNeg: TEdit;
    edHiper: TEdit;
    edPoisson: TEdit;
    Label7: TLabel;
    EdAc: TEdit;
    Label8: TLabel;
    edBc: TEdit;
    Label9: TLabel;
    edCc: TEdit;
    Label10: TLabel;
    edDc: TEdit;
    Panel2: TPanel;
    bIntegra: TButton;
    edIntegra: TEdit;
    Label3: TLabel;
    edFuncion: TEdit;
    bNormal: TButton;
    bExpon: TButton;
    bGamma: TButton;
    bBeta: TButton;
    bTriangular: TButton;
    bWeibull: TButton;
    EdNormal: TEdit;
    EdExponencial: TEdit;
    edGamma: TEdit;
    edBeta: TEdit;
    edTriangular: TEdit;
    edWeibull: TEdit;
    SgDatos: TStringGrid;
    Label1: TLabel;
    edDato: TEdit;
    bLimpiar: TButton;
    bIncluye: TButton;
    bBorra: TButton;
    Panel3: TPanel;
    bOrdena: TButton;
    bCuenta: TButton;
    bSuma: TButton;
    bSum2: TButton;
    bMedia: TButton;
    bMediana: TButton;
    bVarianza: TButton;
    bMin: TButton;
```

```
bMax: TButton;
bRango: TButton;
bSesgo: TButton;
bCurtosis: TButton;
bCuartil: TButton;
EdCuenta: TEdit;
edSuma: TEdit;
EdSum2: TEdit;
edMedia: TEdit;
BbTodas: TBitBtn;
edMediana: TEdit;
edVar: TEdit;
edMin: TEdit;
edMax: TEdit;
edRango: TEdit;
edSesgo: TEdit;
edCurtosis: TEdit;
edCuartiles: TEdit;
ImageList1: TImageList;
BbLimpiar: TBitBtn;
bbSalir: TBitBtn;
procedure tvMenuChange(Sender: TObject; Node: TTreeNode);
procedure EdaKeyPress(Sender: TObject; var Key: Char);
procedure bFactorialClick(Sender: TObject);
procedure bCombinacionClick(Sender: TObject);
procedure bFGammaClick(Sender: TObject);
procedure bBinomialClick(Sender: TObject);
procedure bGeomClick(Sender: TObject);
procedure bBinNegativaClick(Sender: TObject);
procedure bHiperClick(Sender: TObject);
procedure bPoissonClick(Sender: TObject);
procedure bIntegraClick(Sender: TObject);
procedure bNormalClick(Sender: TObject);
procedure bExponClick(Sender: TObject);
procedure bGammaClick(Sender: TObject);
procedure bBetaClick(Sender: TObject);
procedure bTriangularClick(Sender: TObject);
procedure bWeibullClick(Sender: TObject);
procedure EdaExit(Sender: TObject);
procedure FormActivate(Sender: TObject);
procedure bLimpiarClick(Sender: TObject);
procedure bIncluyeClick(Sender: TObject);
procedure bBorraClick(Sender: TObject);
procedure bOrdenaClick(Sender: TObject);
procedure bCuentaClick(Sender: TObject);
procedure bSumaClick(Sender: TObject);
procedure bSum2Click(Sender: TObject);
procedure bMediaClick(Sender: TObject);
procedure bMedianaClick(Sender: TObject);
procedure bVarianzaClick(Sender: TObject);
procedure bMinClick(Sender: TObject);
procedure bMaxClick(Sender: TObject);
```

```
        procedure bRangoClick(Sender: TObject);
        procedure bSesgoClick(Sender: TObject);
        procedure bCurtosisClick(Sender: TObject);
        procedure bCuartilClick(Sender: TObject);
        procedure BbTodasClick(Sender: TObject);
        procedure BbLimpiarClick(Sender: TObject);
        procedure bbSalirClick(Sender: TObject);
      private
        { Private declarations }
        procedure LimpiaGrid(var n : Longint);
        procedure AgregaGrid;
      public
        { Public declarations }
      end;
      {Aquí se quedarán las funciones de manera explícita, pues se
utilizan desde distintas funciones}
      function Integra(f:String; a, b:real) : Double; register;
external 'Pyetool.dll';
      function LimpiaDatos : Boolean; register; external
'Pyetool.dll';
      function IncluyeDatos(x : Double) : Boolean; register;
external 'Pyetool.dll';
      function EliminaDato(n : Longint) : Boolean; register;
external 'Pyetool.dll';
      function OrdenaDatos : Boolean; register; external
'Pyetool.dll';
      function cuenta : LongInt; register; external 'Pyetool.dll';
      function sum : Double; register; external 'Pyetool.dll';
      function sum2 : Double; register; external 'Pyetool.dll';
      function Cuartil(n : Integer) : Double; register; external
'Pyetool.dll';
    var
      FrmEjemplos: TFrmEjemplos;

    implementation

    {$R *.DFM}

    procedure TFrmEjemplos.tvMenuChange(Sender: TObject; Node:
TTreeNode);
    begin
        NbInterfaz.ActivePage := Node.Text;
    end;

    procedure TFrmEjemplos.EdaKeyPress(Sender: TObject; var Key:
Char);
    begin
        if not (key in ['0'..'9','.','-',#8,#13]) then key := #0;
    end;

    procedure TFrmEjemplos.bFactorialClick(Sender: TObject);
    var
```

```
      func  : TFactorial;
      pDll  : THandle;
      total : Double;
   begin
        pDll := LoadLibrary('pyetool.dll');
        if pDll <=0 then Messagedlg('Error al intentar abrir la
librería, checar que esté en el mismo lugar que '+
           'el ejecutable.', mtError, [mbOk], 0);
        @func := GetProcAddress(pDll, 'Factorial');
        total := func(strtoint(eda.text));
        edFact.text := floattostr(total);
        FreeLibrary(PDll);
   end;

   procedure TFrmEjemplos.bCombinacionClick(Sender: TObject);
   var
      func  : TCombinacion;
      pDll  : THandle;
      total : Double;
   begin
        pDll := LoadLibrary('pyetool.dll');
        if pDll <=0 then Messagedlg('Error al intentar abrir la
librería, checar que esté en el mismo lugar que '+
           'el ejecutable.', mtError, [mbOk], 0);
        @func := GetProcAddress(pDll, 'Combinacion');
        total := func(strtoint(eda.text),strtoint(edb.text));
        edComb.text := floattostr(total);
        FreeLibrary(PDll);
   end;

   procedure TFrmEjemplos.bFGammaClick(Sender: TObject);
   var
      func  : TFGamma;
      pDll  : THandle;
      total : Double;
   begin
        pDll := LoadLibrary('pyetool.dll');
        if pDll <=0 then Messagedlg('Error al intentar abrir la
librería, checar que esté en el mismo lugar que '+
           'el ejecutable.', mtError, [mbOk], 0);
        @func := GetProcAddress(pDll, 'G');
        total := func(strtofloat(eda.text));
        edFGamma.text := floattostr(total);
        FreeLibrary(PDll);
   end;

   procedure TFrmEjemplos.bBinomialClick(Sender: TObject);
   var
      func  : TBinomial;
      pDll  : THandle;
      total : Double;
   begin
```

```
        pDll := LoadLibrary('pyetool.dll');
        if pDll <=0 then Messagedlg('Error al intentar abrir la
librería, checar que esté en el mismo lugar que '+
           'el ejecutable.', mtError, [mbOk], 0);
        @func := GetProcAddress(pDll, 'Binomial');
        total                                                    :=
func(strtoint(eda.text),strtoint(edb.text),strtofloat(edc.text));
        edBinomial.text := floattostr(total);
        FreeLibrary(PDll);
    end;

    procedure TFrmEjemplos.bGeomClick(Sender: TObject);
    var
       func  : TGeometrica;
       pDll  : THandle;
       total : Double;
    begin
        pDll := LoadLibrary('pyetool.dll');
        if pDll <=0 then Messagedlg('Error al intentar abrir la
librería, checar que esté en el mismo lugar que '+
           'el ejecutable.', mtError, [mbOk], 0);
        @func := GetProcAddress(pDll, 'Geometrica');
        total := func(strtoint(eda.text),strtofloat(edb.text));
        edGeom.text := floattostr(total);
        FreeLibrary(PDll);
    end;

    procedure TFrmEjemplos.bBinNegativaClick(Sender: TObject);
    var
       func  : TBinNegativa;
       pDll  : THandle;
       total : Double;
    begin
        pDll := LoadLibrary('pyetool.dll');
        if pDll <=0 then Messagedlg('Error al intentar abrir la
librería, checar que esté en el mismo lugar que '+
           'el ejecutable.', mtError, [mbOk], 0);
        @func := GetProcAddress(pDll, 'BinomialN');
        total                                                    :=
func(strtoint(eda.text),strtoint(edb.text),strtofloat(edc.text));
        edBNeg.text := floattostr(total);
        FreeLibrary(PDll);
    end;

    procedure TFrmEjemplos.bHiperClick(Sender: TObject);
    var
       func  : THiper;
       pDll  : THandle;
       total : Double;
    begin
        pDll := LoadLibrary('pyetool.dll');
```

```
        if pDll <=0 then Messagedlg('Error al intentar abrir la
librería, checar que esté en el mismo lugar que '+
           'el ejecutable.', mtError, [mbOk], 0);
        @func := GetProcAddress(pDll, 'HiperGeometrica');
        total                                                    :=
func(strtoint(eda.text),strtoint(edb.text),strtoint(edc.text),strt
oint(edd.text));
        edHiper.text := floattostr(total);
        FreeLibrary(PDll);
    end;

    procedure TFrmEjemplos.bPoissonClick(Sender: TObject);
    var
       func  : TPoisson;
       pDll  : THandle;
       total : Double;
    begin
        pDll := LoadLibrary('pyetool.dll');
        if pDll <=0 then Messagedlg('Error al intentar abrir la
librería, checar que esté en el mismo lugar que '+
           'el ejecutable.', mtError, [mbOk], 0);
        @func := GetProcAddress(pDll, 'Poisson');
        total := func(strtoint(eda.text),strtofloat(edb.text));
        edPoisson.text := floattostr(total);
        FreeLibrary(PDll);
    end;

    procedure TFrmEjemplos.bIntegraClick(Sender: TObject);
    begin
        try
           edIntegra.text                                        :=
floattostr(Integra(edFuncion.text,strtofloat(edAc.text),strtofloat
(edBc.text)));
        except
           Messagedlg('Es posible que la función tenga valores no
soportados, ver la referencia y créditos '+
           'del componente que evalua las funciones en el texto
del proyecto para mas información.', mtError, [mbOk], 0);
        end;
    end;

    procedure TFrmEjemplos.bNormalClick(Sender: TObject);
    var
       func  : TNormal;
       pDll  : THandle;
       total : Double;
    begin
        pDll := LoadLibrary('pyetool.dll');
        if pDll <=0 then Messagedlg('Error al intentar abrir la
librería, checar que esté en el mismo lugar que '+
           'el ejecutable.', mtError, [mbOk], 0);
        @func := GetProcAddress(pDll, 'Normal');
```

```
        total := func(strtofloat(edAc.text));
        edNormal.text := floattostr(total);
        FreeLibrary(PDll);
    end;

    procedure TFrmEjemplos.bExponClick(Sender: TObject);
    var
      ` func  : TExponencial;
       pDll  : THandle;
       total : Double;
    begin
        pDll := LoadLibrary('pyetool.dll');
        if pDll <=0 then Messagedlg('Error al intentar abrir la
librería, checar que esté en el mismo lugar que '+
           'el ejecutable.', mtError, [mbOk], 0);
        @func := GetProcAddress(pDll, 'Exponencial');
        total                                                    :=
func(strtofloat(edAc.text),strtofloat(edBc.text));
        edExponencial.text := floattostr(total);
        FreeLibrary(PDll);
    end;

    procedure TFrmEjemplos.bGammaClick(Sender: TObject);
    var
       func  : TGamma;
       pDll  : THandle;
       total : Double;
    begin
        pDll := LoadLibrary('pyetool.dll');
        if pDll <=0 then Messagedlg('Error al intentar abrir la
librería, checar que esté en el mismo lugar que '+
           'el ejecutable.', mtError, [mbOk], 0);
        @func := GetProcAddress(pDll, 'Gamma');
        total                                                    :=
func(strtofloat(edAc.text),strtofloat(edBc.text),strtofloat(edCc.t
ext));
        edGamma.text := floattostr(total);
        FreeLibrary(PDll);
    end;

    procedure TFrmEjemplos.bBetaClick(Sender: TObject);
    var
       func  : TBeta;
       pDll  : THandle;
       total : Double;
    begin
        pDll := LoadLibrary('pyetool.dll');
        if pDll <=0 then Messagedlg('Error al intentar abrir la
librería, checar que esté en el mismo lugar que '+
           'el ejecutable.', mtError, [mbOk], 0);
        @func := GetProcAddress(pDll, 'Beta');
```

```
        total                                                    :=
func(strtofloat(edAc.text),strtofloat(edBc.text),strtofloat(edCc.t
ext));
        edBeta.text := floattostr(total);
        FreeLibrary(PDll);
    end;

    procedure TFrmEjemplos.bTriangularClick(Sender: TObject);
    var
       func  : TTriangular;
       pDll  : THandle;
       total : Double;
    begin
        pDll := LoadLibrary('pyetool.dll');
        if pDll <=0 then Messagedlg('Error al intentar abrir la
librería, checar que esté en el mismo lugar que '+
           'el ejecutable.', mtError, [mbOk], 0);
        @func := GetProcAddress(pDll, 'Triangular');
        total                                                    :=
func(strtofloat(edAc.text),strtofloat(edBc.text),strtofloat(edCc.t
ext),strtofloat(edDc.text));
        edTriangular.text := floattostr(total);
        FreeLibrary(PDll);
    end;

    procedure TFrmEjemplos.bWeibullClick(Sender: TObject);
    var
       func  : TWeibull;
       pDll  : THandle;
       total : Double;
    begin
        pDll := LoadLibrary('pyetool.dll');
        if pDll <=0 then Messagedlg('Error al intentar abrir la
librería, checar que esté en el mismo lugar que '+
           'el ejecutable.', mtError, [mbOk], 0);
        @func := GetProcAddress(pDll, 'Weibull');
        total                                                    :=
func(strtofloat(edAc.text),strtofloat(edBc.text),strtofloat(edCc.t
ext));
        edWeibull.text := floattostr(total);
        FreeLibrary(PDll);
    end;

    procedure TFrmEjemplos.EdaExit(Sender: TObject);
    begin
        try
           if (sender as TEdit).Text <> '' then
              (sender          as          TEdit).Text          :=
formatfloat('###,###,##0.0######',strtofloat((sender             as
TEdit).Text))
           else
              (sender as TEdit).Text := '0.0';
```

```
        except
           Messagedlg('Error al intentar interpretar el número
tecleado.', mtError, [mbOk], 0);
        end;
     end;

     procedure TFrmEjemplos.FormActivate(Sender: TObject);
     begin
        with sgDatos do
        begin
           Cells[0,0] := 'n';
           Cells[1,0] := 'valor';
        end;
        tvMenu.Fullexpand;
     end;

     procedure TFrmEjemplos.LimpiaGrid(var n : Longint);
     var
        i : Longint;
     begin
        with sgDatos do
        begin
           if n<rowcount-1 then
              n := (rowcount) - n
           else
              n := 1;
           for i:=n to Rowcount-1 do
           begin
              cells[0,i] := '';
              cells[1,i] := '';
           end;
           if n=1 then Rowcount := n+1 else Rowcount := n;
        end;
     end;

     procedure TFrmEjemplos.AgregaGrid;
     begin
        with sgDatos do
        begin
           if (rowcount > 2) or (cells[1,1]<>'') then
              Rowcount := Rowcount + 1;
           cells[0,rowcount-1] := inttostr(rowcount-1);
           cells[1,rowcount-1] := edDato.text;
        end;
     end;

     procedure TFrmEjemplos.bLimpiarClick(Sender: TObject);
     var
        n : Longint;
     begin
        LimpiaDatos;
        n := sgDatos.rowcount;
```

```
     LimpiaGrid(n);
     if (sender <> bbLimpiar) then edDato.Setfocus;
end;

procedure TFrmEjemplos.bIncluyeClick(Sender: TObject);
begin
     IncluyeDatos(StrtoFloat(EdDato.Text));
     AgregaGrid;
     edDato.Setfocus;
end;

procedure TFrmEjemplos.bBorraClick(Sender: TObject);
var
   n : Longint;
begin
     if trunc(strtofloat(edDato.text))>0 then
     begin
        EliminaDato(trunc(strtofloat(edDato.text)));
        n := trunc(strtofloat(edDato.text));
        LimpiaGrid(n);
     end;
     edDato.Setfocus;
end;

procedure TFrmEjemplos.bOrdenaClick(Sender: TObject);
var
   i,j : Longint;
   temp : String;
begin
     Ordenadatos;
     with sgDatos do
     begin
        for i:=1 to rowcount - 2 do
           for j:=i+1 to rowcount - 1 do
              if cells[1,i] > cells[1,j] then
              begin
                 temp := cells[1,j];
                 cells[1,j] := cells[1,i];
                 cells[1,i] := temp;
              end;
     end;
end;

procedure TFrmEjemplos.bCuentaClick(Sender: TObject);
begin
     edCuenta.text := floattostr(cuenta);
end;

procedure TFrmEjemplos.bSumaClick(Sender: TObject);
begin
     edSuma.text := floattostr(sum);
end;
```

```
    procedure TFrmEjemplos.bSum2Click(Sender: TObject);
    begin
         edSum2.text := floattostr(sum2);
    end;

    procedure TFrmEjemplos.bMediaClick(Sender: TObject);
    var
       func  : TDoble;
       pDll  : THandle;
       total : Double;
    begin
         pDll := LoadLibrary('pyetool.dll');
         if pDll <=0 then Messagedlg('Error al intentar abrir la
librería, checar que esté en el mismo lugar que '+
            'el ejecutable.', mtError, [mbOk], 0);
         @func := GetProcAddress(pDll, 'Media');
         total := func;
         edMedia.text := floattostr(total);
         FreeLibrary(PDll);
    end;

    procedure TFrmEjemplos.bMedianaClick(Sender: TObject);
    var
       func  : TDoble;
       pDll  : THandle;
       total : Double;
    begin
         pDll := LoadLibrary('pyetool.dll');
         if pDll <=0 then Messagedlg('Error al intentar abrir la
librería, checar que esté en el mismo lugar que '+
            'el ejecutable.', mtError, [mbOk], 0);
         @func := GetProcAddress(pDll, 'Mediana');
         total := func;
         edmediana.text := floattostr(total);
         FreeLibrary(PDll);
    end;

    procedure TFrmEjemplos.bVarianzaClick(Sender: TObject);
    var
       func  : TVarianza;
       pDll  : THandle;
       total : Double;
    begin
         pDll := LoadLibrary('pyetool.dll');
         if pDll <=0 then Messagedlg('Error al intentar abrir la
librería, checar que esté en el mismo lugar que '+
            'el ejecutable.', mtError, [mbOk], 0);
         @func := GetProcAddress(pDll, 'Varianza');
         total := func(1); //Con un grado de libertad
         edVar.text := floattostr(total);
         FreeLibrary(PDll);
```

```
    end;

    procedure TFrmEjemplos.bMinClick(Sender: TObject);
    var
       func  : TDoble;
       pDll  : THandle;
       total : Double;
    begin
         pDll := LoadLibrary('pyetool.dll');
         if pDll <=0 then Messagedlg('Error al intentar abrir la
librería, checar que esté en el mismo lugar que '+
            'el ejecutable.', mtError, [mbOk], 0);
         @func := GetProcAddress(pDll, 'Minimo');
         total := func;
         edmin.text := floattostr(total);
         FreeLibrary(PDll);
    end;

    procedure TFrmEjemplos.bMaxClick(Sender: TObject);
    var
       func  : TDoble;
       pDll  : THandle;
       total : Double;
    begin
         pDll := LoadLibrary('pyetool.dll');
         if pDll <=0 then Messagedlg('Error al intentar abrir la
librería, checar que esté en el mismo lugar que '+
            'el ejecutable.', mtError, [mbOk], 0);
         @func := GetProcAddress(pDll, 'Maximo');
         total := func;
         edMax.text := floattostr(total);
         FreeLibrary(PDll);
    end;

    procedure TFrmEjemplos.bRangoClick(Sender: TObject);
    var
       func  : TDoble;
       pDll  : THandle;
       total : Double;
    begin
         pDll := LoadLibrary('pyetool.dll');
         if pDll <=0 then Messagedlg('Error al intentar abrir la
librería, checar que esté en el mismo lugar que '+
            'el ejecutable.', mtError, [mbOk], 0);
         @func := GetProcAddress(pDll, 'Rango');
         total := func;
         edRango.text := floattostr(total);
         FreeLibrary(PDll);
    end;

    procedure TFrmEjemplos.bSesgoClick(Sender: TObject);
    var
```

```
      func  : TDoble;
      pDll  : THandle;
      total : Double;
   begin
        pDll := LoadLibrary('pyetool.dll');
        if pDll <=0 then Messagedlg('Error al intentar abrir la
librería, checar que esté en el mismo lugar que '+
           'el ejecutable.', mtError, [mbOk], 0);
        @func := GetProcAddress(pDll, 'Sesgo');
        total := func;
        edSesgo.text := floattostr(total);
        FreeLibrary(PDll);
   end;

   procedure TFrmEjemplos.bCurtosisClick(Sender: TObject);
   var
      func  : TDoble;
      pDll  : THandle;
      total : Double;
   begin
        pDll := LoadLibrary('pyetool.dll');
        if pDll <=0 then Messagedlg('Error al intentar abrir la
librería, checar que esté en el mismo lugar que '+
           'el ejecutable.', mtError, [mbOk], 0);
        @func := GetProcAddress(pDll, 'Curtosis');
        total := func;
        edCurtosis.text := floattostr(total);
        FreeLibrary(PDll);
   end;

   procedure TFrmEjemplos.bCuartilClick(Sender: TObject);
   var
      i : Smallint;
   begin
        for i:=1 to 4 do
        begin
           edCuartiles.text      :=      edCuartiles.text      +
floattostr(cuartil(i))+',';
        end;
   end;

   procedure TFrmEjemplos.BbTodasClick(Sender: TObject);
   begin
        bCuentaClick(nil);
        bSumaClick(nil);
        bSum2Click(nil);
        bMediaClick(nil);
        bMedianaClick(nil);
        bVarianzaClick(nil);
        bMinClick(nil);
        bMaxClick(nil);
        bRangoClick(nil);
```

```
     bSesgoClick(nil);
     bCurtosisClick(nil);
     bCuartilClick(nil);
end;


procedure TFrmEjemplos.BbLimpiarClick(Sender: TObject);
begin
     //Pantalla de funciones discretas
     Eda.text := '0.0';
     Edb.text := '0.0';
     Edc.text := '0.0';
     Edd.text := '0.0';
     EdFact.text := '0.0';
     EdComb.text := '0.0';
     EdFGamma.text := '0.0';
     EdBinomial.text := '0.0';
     EdGeom.text := '0.0';
     edBNeg.text := '0.0';
     edHiper.text := '0.0';
     edPoisson.text := '0.0';
     //Las funciones continuas
     EdAc.text := '0.0';
     edBc.text := '0.0';
     edCc.text := '0.0';
     edDc.text := '0.0';
     edIntegra.text := '0.0';
     EdNormal.text := '0.0';
     EdExponencial.text := '0.0';
     edGamma.text := '0.0';
     edBeta.text := '0.0';
     edTriangular.text := '0.0';
     edWeibull.text := '0.0';
     //De estadística descriptiva
     edDato.text := '0.0';
     bLimpiarClick(sender);
     EdCuenta.text := '0.0';
     edSuma.text := '0.0';
     EdSum2.text := '0.0';
     edMedia.text := '0.0';
     edMediana.text := '0.0';
     edVar.text := '0.0';
     edMin.text := '0.0';
     edMax.text := '0.0';
     edRango.text := '0.0';
     edSesgo.text := '0.0';
     edCurtosis.text := '0.0';
     edCuartiles.text := '0.0';
end;

procedure TFrmEjemplos.bbSalirClick(Sender: TObject);
begin
```

```
    close;
end;

end.
```

Todo el código es este y se puede apreciar la mayor parte de él son definiciones, las rutinas pesadas de cálculo están siendo utilizadas y dejadas en libertad cada vez que ya no se usan mas, salvo algunas que están definidas de manera explícita al inicio del código, la gran mayoría de las rutinas del *Dll* son usadas y la memoria se libera una vez que ya terminaron de ejecutarse. Este código también puede ser utilizado como muestra para proyectos personales y a partir de ahí generar rutinas mucho mas adecuadas a cada problema.

Anexo 3. Apuntes sobre un curso de Delphi

Este anexo fue recopilado por el Lic. Jorge Alberto Pérez Torres para llevar a cabo un curso de Delphi en la compañía Asesores en Tecnología y Logística, de donde se tomó para incluirlo como parte de los fundamentos de la programación en Delphi, esperando sea de mucha ayuda lo que a continuación se describirá, pues sirve como datos adicionales al respecto de la programación en Delphi y en sí del funcionamiento en general del lenguaje

Este curso funcionó con un grupo de estudiantes que se encontraban en el caso que yo planteo como objetivo del proyecto, teniendo resultados muy alentadores, esa es la razón para incluirlos en el desarrollo del presente proyecto.

Su objetivo es aclarar un poco mas el funcionamiento del lenguaje y ampliar la introducción que se hizo a Delphi.

I. Fundamentos de la programación orientada a objetos (POO)

La evolución de estilos de codificación ha estado dada por los niveles de organización al programar.

Por su nivel de organización podemos reconocer tres etapas características:

Programación spaghetti.
Programación estructurada.
Programación orientada a objetos.

Las bibliotecas de programas son el primer paso hacia la POO.

La POO nos provee de una serie de herramientas y conceptos que nos permiten plasmar de una manera más ordenada nuestras soluciones:

Encapsulamiento: Organización de métodos y variables con características afines para poder ser reproducidos en instancias.

Se busca generar entes aislados e independientes que interactúen con otros (como en el mundo real).

Herencia. Facultad de generar encapsulaciones a partir de otras encapsulaciones

Polimorfismo: Múltiple utilización o aprovechamiento de una encapsulación.

Clase: Definición de una encapsulación.

Objeto: Instancia de una clase.

Las clases son portables, reutilizables e implican un nivel alto de abstracción.

La POO ha permitido el auge de los lenguajes visuales. Los componentes de la ventanas están organizados bajo un árbol de herencias que permite compartir funcionalidad entre los elementos visuales del ambiente gráfico

Existe una gran cantidad de herramientas visuales de desarrollo en el mercado, pero no todas son lenguajes de programación y solamente algunos poseen compiladores en el sentido estricto de la palabra. Por ejemplo, Delphi es un compilador y VB es un intérprete, es decir, no genera código nativo sino macros que son compiladas en C para generar un ejecutable.

II. Conceptos generales acerca de Windows

Ambiente gráfico

La necesidad de contar con ambientes gráficos para desarrollar funciones en la computadora parte del concepto de interfaz.

Interfaz (entre caras): Son todos los medios con los cuales contamos para interactuar con el usuario. Puede haber interfaces gráficas (GUI, Graphic User Interface) o no gráficas (llamadas comúnmente de texto).

Windows es un ambiente gráfico (es decir, una GUI), y se encuentra basado en una serie de fundamentos tecnológicos y conceptuales que buscan una fácil adaptabilidad al uso por parte del usuario. Dos conceptos fundamentales en cuanto a la implementación de la funcionalidad de Windows (independientemente de la parte gráfica en sí, que Windows hace casi transparente a los programadores) son los **eventos** y los **mensajes**.

Mensajes: Estados que proporciona el sistema operativo para la interacción entre aplicaciones con éste y entre ellas mismas.

Eventos: Son definidos como los medios de notificación para la recepción de mensajes. Por ejemplo, cuando se mueve el ratón se activan una serie de mensajes que pueden ser atrapados por diferentes eventos que nos permiten al programar saber qué es lo que está pasando del otro lado de nuestra faz. No forzosamente son activados por el usuario, ya que otras aplicaciones nos pueden enviar mensajes.

API (Application Program Interface) de Windows: Son el cúmulo de funciones y procedimientos básicos que permiten a un programador elaborar aplicaciones para este ambiente gráfico a partir de los elementos gráfico-visuales que lo componen

La MFC (Microsoft Foundation Classes) es una jeraquía de clases que provee las clases que componen toda la funcionalidad del ambiente gráfico y por supuesto se encuentran programados en un esquema orientado a objetos, que de hecho fue la manera de integrar más fácilmente toda la funcionalidad inherente a la interfaz.

Existe una gran cantidad de elementos visuales, que son objetos y tienen asociados a si mismos eventos y poseen sus propias variables y métodos (recordemos que un método es una función o procedimiento asociado a una clase) A continuación enumeraremos los más comunes

Botón (Button)
Label
Edit
Combo box
List box
Check box
Radio button
Speed button
Tab set

Una combo box y una list box se parecen, pero el primero siempre implica una selección y el segundo puede presentar múltiples selecciones, además que el primero ahorra más espacio visual

Un check box y un radio button permiten elegir entre opciones, pero el primero se utiliza para selecciones no excluyentes y en el radio button las opciones son mutuamente excluyentes per se.

En un tab set puede haber incluídos muchos otros controles.

Todos los controles están supeditados a una ventana, la cual se define como el espacio en el cual se constriñen los componentes visuales de la interfaz de usuario final.

Existen básicamente dos tipos de ventanas. Las modales y las no modales. Las primeras normalmente se componen de diálogos, que son ventanas que solicitan información al usuario (por eso se dice que "dialogan" con él). La manera más fácil de reconocer si una ventana es modal o no es verificar si es posible o no capturar información en otra ventana d la aplicación sin tener que cerrar la ventana en que nos encontramos. Cuando le pedimos información al usuario asumimos que nuestras ventanas serán modales. Las ventanas no modales son todas las demás.

Existen también dos tipos de aplicaciones, a saber, MDI (Multiple Document Interface) y SDI (Single Document Interface). Las primeras engloban aplicaciones que permiten al usuario tener varias copias de una ventana al mismo tiempo y por lo tanto hacen uso de ventanas o modales (aunque conviven con ventanas modales). Ejemplos de éstas los tenemos en Excel, Word para Windows, Power point, etcétera. Las segundas son hechas sólo con ventanas modales. Curiosamente prácticamente la mayoría del software hecho a la medida es SDI. Puede haber híbridos

III. Terminología POO en Delphi

Componentes: Objetos provistos por la herramienta de desarrollo con funcionalidad básica para poder desarrollar aplicaciones. En el caso de Delphi la cantidad y calidad de componentes que incluye permite construir aplicaciones robustas sin necesidad de adquirir componentes en el mercado

Clases: Como habíamos mencionado son la definición de una encapsulación. Delphi por definición declara una clase al momento que empezamos a crear una ventana. En dicha clase Delphi declara los componentes que vamos colocando y cuando nosotros les asociamos los eventos a atrapar declara funciones asociadas a estos. El programador puede declarar todos los métodos y variables que necesite para darle toda la funcionalidad necesaria a las ventanas.

No es forzoso que asociemos una clase a declaraciones de ventanas y/o componentes. Podemos declarar nuestras propias clases si lo juzgamos necesario, y por lo tanto podemos hacer programas en los cuales no existan ventanas.

Sintaxis.

Delphi está basado en el Object Pascal, por lo cual alguien familiarizado con Pascal encontrará sencilla su adaptación al lenguaje. La sintaxis de las clases es simple ya que el mismo lenguaje nos la proporciona. Solamente es importante recordar que podemos hacer uso de las palabras reservadas **private** y **public** para que otros programas tengan o no acceso a variables y métodos de un objeto declarado a partir de nuestra clase Normalmente los métodos y variables asociados a la implementación de funcionalidad no deben ser públicos a otros programas si no queremos encontrarnos con sorpresas desagradables. A

El operador de ámbito que nos permite acceder a los métodos y variables de un objeto es el punto. Por lo tanto, al implementar código en delphi debemos tener cuidado en siempre hacer que nuestros métodos y variables sean parte de la clase y no los declaremos por separado, como era costumbre en Pascal.

Un ejemplo del uso del operador de ámbito es:

```
Lista.Items.Clear;
```

Lista es un objeto de la clase ListBox que posee una propiedad (objeto) llamado Items (de la clase TStrings) que a su vez posee un método denominado Clear que limpia el contenido de la lista d cadenas que mantiene. Como vemos es punto es el operador que nos permite acceder a las propiedades o métodos asociados al componente u objeto.

Ya que mencionamos las propiedades, estas se definen en Delphi como aquellos miembros de una clase que pueden ser afectados o modificados en su estructura interna por medio de métodos Puede haber variables u objetos como propiedades

Los métodos son aquellas funciones o procedimientos asociados a una clase Un ejemplo típico para ilustrar la diferencia entre método y clase es el siguiente

Tabla.Open;

Tabla.Active.= True;

Las dos líneas de código son análogas, puesto que el resultado de ejecutarlas es el mismo (abren una tabla de una base de datos) La diferencia radica en el hecho de que la primera se hace por medio de una llamada al método Open del objeto tabla de la clase TTable y la segunda es afectando a la propiedad Active para el mismo objeto de la misma clase.

Eventos

Delphi tiene implementado el atrapado o "cachado" (catch) de los eventos más comunes para una mayor facilidad de programación. De hecho muy raras veces es necesaria la implementación del cachado de eventos no contemplados por el lenguaje. Por ejemplo, en un componente como una caja de edición (Edit) existen eventos como la captura de tecla, el paso del ratón por el control, el dar clic o doble clic con éste mismo dispositivo, el seleccionar el control, etcétera. Esto nos permite elegir los eventos que nos interesa atrapar y decirle a la aplicación qué es lo que queremos que haga en cada caso.

Como implicaciones de la herencia, la encapsulación, y el polimorfismo en Delphi, tenemos que todos los componentes que incluyen se encuentran en una jerarquía de clases denominada Visual Component Library (VCL).

La VCL es un estrato más que nos sirve como interfaz entre el programador y el API de Windows y que tiene como fin de que no nos preocupemos por muchos detalles repetitivos y engorrosos al programar y estemos atentos solamente a los algoritmos. Este es el valor agregado más importante que se le puede dar a un programador.

El tener una jerarquía de clases no solamente provee de una facilidad de desarrollo a los que implementan un lenguaje visual (en este caso Delphi), sino que permite que los desarrolladores de aplicaciones puedan hacer uso de dicha jerarquía para utilizar las potencialidades del lenguaje. El TStrings es un ejemplo diáfano de este caso.

IV. Dentro de Delphi

Extensiones de archivos

pas	Código fuente en Object Pascal
dfm	Delphi Form. Fuente asociado a la forma

dpr Delphi Project. Es el archivo que tiene el control de todas las unidades que se ocupan en el proyecto

A nivel código objeto:

dcu Delphi compiled Unit Es la unidad compilada de Delphi.

obj Código objeto. Habrá tantos OBJ como tantas unidades PAS se tengan. Es posible compilar para generar dcus u objs

exe Este es el que se puede correr. Tiene como base el *.dpr

dll Biblioteca de ligado dinámico. Es código que se puede ejecutar al ser llamado por un ejecutable y puede ser compartido al mismo tiempo por varias aplicaciones.

res Archivo de recurso. Se refiere a los objetos gráficos que componen las ventanas de nuestra aplicación

No olvidar que hay dos maneras de generar código objeto: Compile y Build All. La primera solamente revisa qué programas han sido compilados viendo el archivo de cabeceras precompiladas (extensión dsm). En el segundo caso siempre se compilarán todos los programas. Si no existe el fuente, pero existen los archivos objeto o las dcus entonces podremos compilar el programa.

Sintaxis básica en Delphi

Un programa en Delphi tiene dos secciones principales: INTERFACE y el IMPLEMENTATION. En la primera tenemos la declaración de nuestras clases, tipos, constantes y variables a utilizar por nuestra aplicación y en el segundo su implementación. En el primer caso de los métodos hablamos de que su declaración en la clase se conoce como el prototipo de la función. La cláusula USES en esta sección declara comúnmente bibliotecas de programas y en la segunda unidades que interactúan con la que estemos implementando.

Almacenamiento de datos en memoria y su representación real

Algunos tipos de datos son Real, Double, Integer, Word, Byte, Boolean, String, Char, Smallint. Cada uno pudiendo ocupar 8, 12 ó 16 bits, pero con diferente representación en el mundo real (ver la ayuda de Delphi al respecto).

Es importante en estos casos entender cómo es que un tipo u otro se almacena en la memoria

La palabra "HOLA" por ejemplo.

POSICIONES EN MEMORIA

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

x: array[50] of char;

H O L A \0

x: string[20],

4 H O L A

De lo anterior de deduce que es más rápido leer de un string que de un arreglo porque se conocede antemano exactamente cuantos caracteres se van a leer. De este modo podemos almacenar hasta 255 caracteres.

No olvidemos que en Delphi 3 existe un nuevo tipo llamado AnsiString que permite la declaración de cadenas de 65536 caracteres, y no como el antiguo tipo String que solamente permitía almacenar hasta 255 caracteres.

Creación de formas

Existen dos maneras de crear formas. al momento de iniciar la aplicación y cuando se deba abrir una ventana específica. La primera provoca que nuestra aplicación ocupe más memoria (cada ventana que creamos ocupa cierta cantidad de memoria) pero facilita al programador su implementación y la ejecución es un poco más rápida. Las formas no modales deben forzosamente ser creadas de esta manera. Las formas modales pueden ser creadas y destruidas durante cualquier momento del programa. Esto sucede porque al crear una forma y mostrarla de manera modal la ejecución del programa que la llamó se interrumpe hasta salir de la forma llamada, por lo que al liberar la memoria no hay problema. En caso de un llamar al método Show el problema estriba en que la ejecución no se interrumpe y por lo tanto la liberación de memoria se ejecuta inmediatamente después de que el programa alcanza a mostrar la ventana.

Para crear una ventana en memoria entramos al menú Project, elegimos Options y en la ceja del TabSet con el nombre Forms dejamos en Auto-Create Forms, la forma principal y en Available Forms dejamos las demás formas. En el código se escriben las siguientes declaraciones:

```
x:=Tx.Create(Self),
x.ShowModal;
x.Free;
```

Suponiendo que x es el sufijo asociado a nuestra clase y el nombre del objeto Es importante hacer énfasis en que podríamos tener un objeto que se llamara "y" aunque su clase sea "Tx".

Show es el método para mostrar ventanas no-modales
ShowModal es el método para mostrar ventanas modales

Eventos comunes de utilización en las formas:

Form Create. Es un evento que crea la forma.

On Create. Se ejecuta cuando Windows reserva espacio y el objeto es generado en memoria

On Activate. No se pone ahí porque tiene que repintar algunas cosas.

On Close. Se activa cuando se cierra la ventana.

Cuando se está corriendo el programa la vista que tenemos es una ventana.

Cuando no se está corriendo el programa la vista que tenemos es una forma.

Es importante considerar que cuando no sepamos qué es lo que un evento está atrpando hagasmo uso de la ayuda o ejecutemos el programa utilizando el debugger para ver qué eventos se ejecutan.

Métodos de utilización común

Para adicionar datos a una lista (utilizando el método Add de la propiedad Items de la clase TStrings) se utiliza la declaración siguiente:

x.Items.Add(y)

En donde x. TListBox ó x: TComboBox y y: String.

Para borrar datos de una lista se utiliza la siguiente declaración:

x.Items.Delete(i)

En donde x: TListBox ó x. TComboBox y i: Índice del elemento a borrar.

No olvidar que estos métodos se asocian a la clase TStrings y no tanto a los componentes que se enuncian

El elemento "i" puede ser encontrado en el caso de que hablemos del elemento seleccionado se encuentyra en la propiedad ItemIndex. Esta propiedad no está a nivel de TStrings, sino del componente.

Si la lista no contiene elementos ItemIndex devuelve un -1.

Todas las listas de cadenas empiezan desde el elemento cero.

En TStrings Existe una propiedad que nos devuelve el número de elementos que hay en una lista. Esta es Count. Clear es el método que borra todos los elementos de la lista

Como TStrings es la misma clase en el caso de una ComboBox y una ListBox (y cualquier componente que lo contenga), entonces puedo hacer asignaciones completas, como por ejemplo

ListBox.Items := ComboBox.Items

Propiedades misceláneas:

La propiedad que impide capturar datos en una ComboBox es Style en su opción csDropDownList

La propiedad que permite a un botón hacerse visible es la siguiente·

Button.Visible:= True;

Todas las propiedades que se pueden asignar desde el Object Inspector pueden asignarse en código.

Cuando accedemos a un elemento de una lista, arreglo o cadena (variable en memoria) se utilizan corchetes corchetes. Los paréntesis son para llamados a métodos con que tengan parámetros.

Variable: [ ]
Método: ( )

16 bits y 32 bits

Delphi 1 0 compila a 16 bits, Delphi 2.0, 3.0 y 4.0 compilan a 32 bits.

Windows 3. Trabaja a 16 bits, Windows 95 trabaja a 32 bits.

Un programa a 32 bits corre más rápido que uno a 16 bits. Un programa hecho a 32 bits no puede correr a 16 bits, es decir, no es compatible con Windows 3.1 o versiones

anteriores Un programa hecho para 16 bits si puede correr en Windows 95 a pesar de trabajar a 32 bits En Windows 98 existirán algunas restricciones.

Manejo de excepciones

Una excepción es un error atrapado por el lenguaje de programación que puede detener la ejecución normal de un programa. Podemos dejar que dicha excepción se detenga el programa o podemos decirle qué hacer cuando ésta ocurra.

Existe la cláusula o instrucción **try**. Hay

```
A)                    B)
try                   try

   {Código asociado}      {Código asociado}

   finally                  except

{Código asociado}      {Código asociado}

end;                  end;
```

En el caso A en caso de que haya o no una excepción se ejecutará el código asociado a finally (por ejemplo, queremos liberar memoria de objetos pase lo que pase).

En el caso B solamente se ejecutará el código asociado a except cuando en efecto haya ocurrido una excepción en la aplicación.

V. Controles asociados a memoria

Estos controles (componentes que interactúan con el usuario final) son utilizados para manipular datos en memoria. Esto no significa que no se pueda guardar la información almacenada en ellos, más bien, que no existe dentro de su funcionalidad una manera de relacionar propiedades y/o métodos directamente a bases de datos.

VI. Manipulando datos

Controles asociados a datos

Estos controles nos sirven para implementar funcionalidad de acceso y modificación a elementos de una base de datos

Un caso de DBMS: SQL Anywhere

Por lo general las Bases de Datos residen en un Servidor con características técnicas muy especificas. Sin embargo, Sybase nos permite manipular los datos no necesariamente desde un servidor, pues tiene la bondad de conceder hacerlo a un nivel de PC. Esto es una gran ventaja, ya que se puede manipular una sola base de datos por separado sobre varias PC's y sin la necesidad de contar con un servidor. Además a comparación de otros DBM su costo es bastante accesible.

Cada vez que se compra un manejador, este contiene un servicio, el cual permite incluir n bases de datos.

Una vez que se acceda a un servicio, (por ejemplo **SQL-RvOper (dba) On RvOper)**, como el DBA se pueden hacer varias operaciones según los permisos contenidos para el usuario.

Con el comando F7 se muestran las tablas que contiene el sistema, las cuales tienen dos nombres separados por un punto

SYS.Claves
DBA.Clavess
DBO.Clavesss

Los caracteres que se encuentran antes del punto son los que conforman el nombre del usuario que creó la tabla. Después del punto se escribe el nombre de la tabla.

Es necesario que siempre que se realice una sentencia de SQL, el nombre de la tabla cuente con el nombre del usuario por cual fue creada, a menos que las tablas utilizadas hallan sido creadas por el usuario con el cual accedamos al DBM.

El comando Rollback regresa las tablas como estaban antes.

ODBC (Open Database Connectivity)

Utilizando ODBC podemos relacionar un manejador de base de datos cualquiera con un lenguaje de programación cualquiera. Los manejadores de base de datos implementan las funciones y cláusulas de SQL estándar (ver anexo 1) de al manera que mejor les conviene a sus desarrolladores. Cada manejador tiene características propias en instrucciones y aplicaciones para sus usuarios, pero debe cumplir los requisitos de estandarización en el SQL general. La construcción interna de un select es diferente dependiendo del manejador (por las estructuras de datos internas que maneje, la manera en que maneje los datos, la programación en sí, etc.), por lo cual si cada lenguaje de programación quisiera hacer un select como tal debería llamar a la función especifica con los parámetros, inicializaciones de ambiente y objetos en memoria por cada manejador que se utilizase. Por esto aparece DBC, que estandariza en funciones generales las llamadas del lenguaje de programación hacia el MBD y a su vez establece estándares a los proveedores de dicho software para que

pueda hacer los llamados a dlls que estos deben proveer y así hacer transparente a los desarrolladores de la implementación, quedando en manos de los proveedores de la industria de una manera estándar

El principal problema de ODBC es su lentitud (muy comprensible si entendemos las implicaciones detas traducciones de funciones), por lo cual ciertos proveedores de software MDB se unen a las empresas que desarrollan lenguajes de programación con el fin de que se pueda proveer al desarrollador de drivers nativos (es decir, acceso a funciones nativas para que el acceso a datos sea más rápido en lugar de pasar por ODBC) por esto el alto precio de las versiones cliente/servidor.

(No olvidar el diagrama dibujado en clase).

VII. Tópicos Selectos

Desarrollo concurrente de aplicaciones

Normalmente en el desarrollo de una aplicación concurre más de un programador. Esto implica que n programadores trabajen en un mismo proyecto pero afectando programas diferentes. En primera debe tratarse al máximo que todos compartan los mismos fuentes en la red, y las compilaciones se hagan de manera local. En segunda puede haber casos de programas estratégicos en los que sea mejor hacer una copia de manera local y modificarlos para que después de probarlos. Muchas veces se piensa que ésta es la mejor manera de trabajar, pero no es así ya que cuando se deben integrar las versiones (dejar en un proyecto todos los fuentes que se consideran válidos para la aplicación) se tienen dolores de cabeza más frecuentes.

No se debe echar al saco roto el consejo de respaldar los programas. Tampoco se debe exagerar en este caso porque a veces tenemos muchas versiones de la misma aplicación y no podemos distinguir cuál es el respaldo que nos es útil. La mejor manera de respaldar es colocando un directorio bajo el mismo lugar en que lo tenemos (o en otro disco pero con el mismo nombre) pero con fechas que digan de cuándo es el respaldo. Si se puede colocar información adicional en un archivo de bitácora (en texto para ser fácilmente consultado) se tienen muchos elementos para saber si dicho respaldo nos es o no útil.

Estandarización de las aplicaciones

Existen estándares en dos vías: a nivel codificación y a nivel interfaz.

Cuando codificamos debemos preguntar si algunas funciones que nos parezcan genéricas ya existen el bibliotecas programadas dentro de la organización y por ende utilizarlos. En caso de que no sea así lo ideal es construirla e incorporarla a la biblioteca que nos parezca adecuada (funciones matemáticas, utilerías, uso de cadenas, etc.). Si tampoco existe una biblioteca que nos parezca la apropiada entonces debemos crear una nueva Nunca debemos dejar estas funciones en los programas de la aplicación porque su hechura se repetirá n veces.

Los estándares varían en función de las organizaciones. Lo que se debe hacer es preguntarlos y seguirlos. Detalles como las sangrías y la manera de colocar los comentarios al código son vitales aunque pudieran parecer triviales.

A nivel interfaz existen botones con nombres, tamaños e imágenes predefinidas para funciones específicas de los sistemas. También hay estándares en los menúes, el orden de los controles y tamaños de campos. Dichos estándares son normalmente seguidos por analogía y en otros casos existen documentos formales al respecto.

Diseño de ventanas y modelos entidad-relación

Cuando dibujamos ventanas de captura el mismo diseño de la ventana nos permite identificar entidades y relaciones entre estas. Por ejemplo, si tenemos una combo en la que se guarden agentes y debajo tenemos otras combos asociadas a éste (tal vez una de nivel de sueldo) entonces estamos asociando las dos en una tercer entidad, y su existencia la enuncia la pantalla. Con la práctica es más fácil ir encontrando estas relaciones y por lo tanto debemos buscar que estas relaciones sean evidentes cuando dibujemos pantallas o nos las dibujen. Esto hace también que el usuario note estas relaciones de manera más clara y por lo tanto su aclimatación a los sistemas será más inmediata.

Los 11 mandamientos de la codificación

1. Divide los problemas en pequeñas piezas.
2. Usa nombres de variables que signifiquen algo y no generen confusión
3. Usa bibliotecas de funciones siempre que sea posible.
4. No rehagas código ya hecho, optimízalo si es necesario.
5. No sacrifiques la claridad y la eficacia por pequeñas ganancias en eficiencia.
6. Reemplaza expresiones repetitivas por llamadas a funciones comunes.
7. Trata de evitar la ambigüedad.
8. No te enfrasques en un error de codificación.
9. No te limites a reutilizar el código, reorganízalo.
10. Haz a tus comentarios coincidir con el código que explican.
11. Pregunta si no sabes.

Anexo 4: Petición de datos con SQL

Utilizando DBE de Delphi.

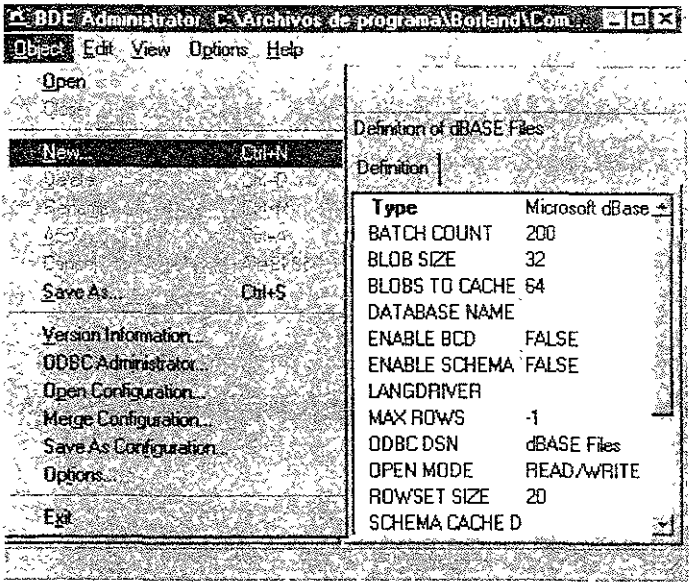
El objetivo de este anexo es mostrar la configuración desde Delphi del DBE (DataBase Engine) pues es este el que permite conexión con cualquier base de datos en el mercado, siempre que se cuente con los controladores de la base. Además se mostrarán las sentencias básicas de SQL estándar y como se pueden usar en los componentes Query de Delphi. La intención no es generar expertos en bases de datos, sino mostrar el uso de las sentencias mas comunes y útiles en este ya tan generalizado lenguaje de peticiones y que puede ser de mucha utilidad para los desarrolladores.

Para el caso del capítulo 1 en el que se tiene un directorio personal, a continuación se muestra la forma en que se configuró la base de datos que se utiliza, así como el uso que se puede dar a los componentes TQuery con los datos.

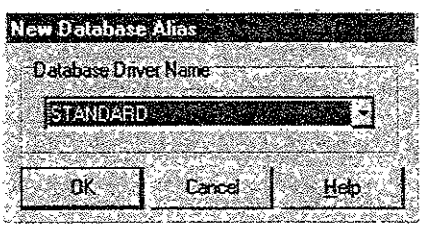
Primero, veremos el lugar desde donde se deben configurar las tablas de datos para su utilización. La aplicación que tiene Delphi para su configuración es el DBE o DataBase Engine, la cual podemos activar desde el menú de inicio (en Windows 95 o posterior), la carpeta donde esté instalado Delphi y después eligiendo BDE Administrator. En una instalación sin modificar los valores por default, se tendría que accesar:

Inicio > Delphi 3 > BDE Administrator

Y dentro de la pantalla, se tendrá que solicitar un nuevo controlador o Alias de base de datos para que lo reconozca Delphi. En el caso del directorio, se utilizó el alias: Telefonos.

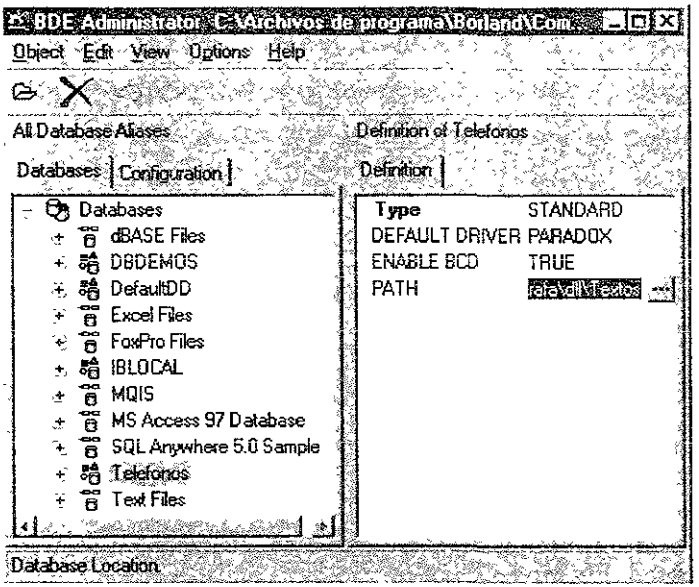


Una vez que se eligió crear un alias, se presentará una ventana para elegir el *Driver* de la base de datos que queremos usar, en este caso se utilizó el estándar, pues las tablas se crearon en Paradox, que es la forma estándar en Delphi, de usar tablas. La pantalla es la siguiente:



Una vez elegida la opción, se creará un nuevo elemento en el arbol de Alias de la pantalla del DBE, en este se pondrá el nombre que elijamos para el controlador, quedando la pantalla lista para seguir configurando las opciones restantes. Para el ejemplo, solo falta poner el driver a manejar, que será *PARADOX*, si queremos o no compartir los datos con otras aplicaciones (Enabled BCD), a la cual le pondremos el valor *True*, y finalmente, la ruta en donde encontrará las tablas que se usarán. En cada caso diferente para los drivers de las bases de datos, los valores pueden cambiar, sin embargo básicamente se necesita especificar la ruta donde esté la base o el driver ODBC que se debe usar para accesar a los datos

La vista de la pantalla, al final de configurarla es como a continuación.



Una forma de saber que el alias que se acaba de dar de alta está bien configurado es abriendolo, sino encontramos ningún error al hacerlo, Delphi tampoco lo encontrará. Para abrirlo debemos escoger la opción *Open* del menú *Object*.

Cuando el alias está creado, se puede accesar desde algún componente Delphi que tenga la capacidad de manejar datos desde alguna base, el componente que veremos en este caso es el TQuery, pues desde este se pueden ejecutar comandos de SQL estándar, que como ya dije, es el objetivo de este anexo mostrar un poco de su uso.

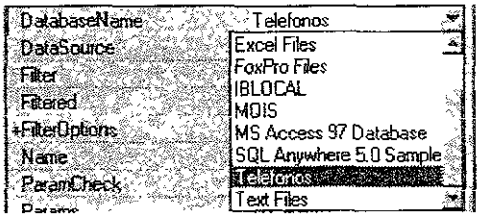
Utilizando el alias generado

Entonces, supongamos que existe otra aplicación que generamos especialmente para hacer una aplicación que lea la base de datos que acabamos de configurar en el alias *Telefonos*, de forma que desde un programa podamos verla, para esto, necesitamos pegar el componente TQuery en la forma que hagamos, el componente se ve así en la paleta.

De este componente tenemos que modificar las siguientes propiedades para poder ver las tablas que queremos, todas las modificaciones se harán desde el *Object Inspector*[22], que vimos en el capítulo 1.

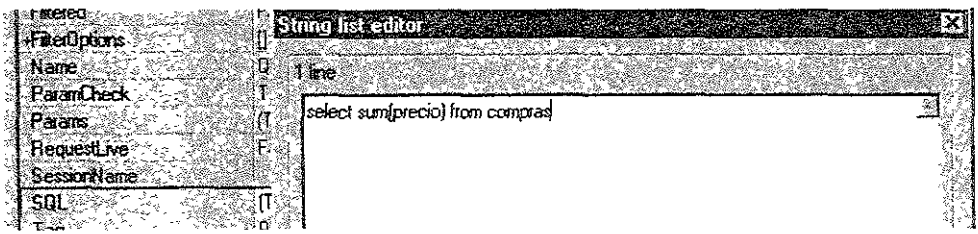
[22] Estando en Delphi, para mostrar el *Object Inspector*, se debe presionar la tecla F 11

a) **DatabaseName**: Es aquí donde se especifica el alias que el componente utilizará cuando quiera abrir una tabla, este valor puede ser modificado en *tiempo de ejecución* siempre y cuando el *query* no este activo y el alias que le solicitamos exista, sin embargo en este caso, lo especificaremos en *tiempo de desarrollo*.



Al modificar esta propiedad dejamos disponible la base de datos para que se utilice. Por otro lado, aunque para el ejemplo el contenido del query se estará modificando en tiempo de ejecución, en esta parte se mencionarán otras propiedades que se pueden modificar en caso que se quiera dar otro uso al componente. Estas propiedades son:

- **SQL** · Que nos llevará a una pantalla de captura para que se incluyan los comandos SQL que se usarán, en caso de que el query solo nos sirva para obtener algún dato bien específico en un proceso, simplemente se pondría algo como **"select sum(precio) from compras"**, para obtener la suma de precios que se encuentren en la tabla de compras. En Delphi se vería algo así:



- **Params**: En caso que el query no sea para extraer siempre el mismo dato, sino que pueda ponerse algún parámetro dentro del comando SQL para que varíe la extracción del dato dependiendo de las necesidades del proceso, entonces, se puede poner un código como a continuación

Principales comandos y su uso.

Primero debe quedar claro que el hacer peticiones a la base de datos desde un componente TQuery en Delphi, es hacer una petición general, en la que se pueden manejar en un solo "apuntador" el contenido desde una o varias tablas, hasta un dato específico, sin embargo, hay que hacer las consideraciones necesarias para que los datos puedan ser tomados de dicho "apuntador", de la manera adecuada.

Select {campos} from {tabla}

Este comando traerá el contenido de los campos que se especifiquen o de todos los campos, de todas las tablas que se necesiten, empecemos con el caso mas sencillo, cuando se quieren traer todos los campos de la tabla tels, el comando sería:

*Select * from tels*

Nombre	Dirección	Teléfono	Cumple
Rafael Muñoz Gómez	Rio Churubusco # 370	01-5-554-4879	30/09/74
Infinita Consultores	Irlanda # 162	01-5-549-5781	10/04/97

En este caso, estos son todos los campos (o columnas), contra todos los registros (o renglones) de la tabla. En caso de que se quiera mostrar solo algún campo, el comando sería:

Select nombre, telefono from tels

nombre	telefono
Rafael Muñoz Gómez	01-5-554-4879
Infinita Consultores	01-5-549-5781

Así se pueden especificar tantos campos como se deseen, simplemente hay que nombrarlos y separarlos por una coma. Existe una forma de condicionar el resultado del query, solo se tiene que usar la cláusula where.

Where campo {condición} valor o campo

Agregando este comando al *select*, se puede especificar mejor el resultado esperado, por ejemplo, si solo se quiere traer el registro que tiene telefono igual al 01-5-549-5781, se puede utilizar el comando:

Select nombre, telefono from tels where telefono = '01-5-549-5781'

nombre	telefono
Infinita Consultores	01-5-549-5781

Entre las condiciones se pueden utilizar los signos básicos de relación, el igual, mayor que, menor que, menor igual, etc. sin embargo, existe uno que puede funcionar para casos de búsqueda no tan definida, sino en los casos que solo se cuenta con alguna parte de la información, este puede ser el caso siguiente: necesito el nombre y la dirección del registro que tenga como número exterior el 370. Para este caso, se puede usar un comodín, el valor del comodín es "%" y funciona bajo la misma filosofía que el * en DOS, no importa cuantos ni cuales caracteres tenga en el lugar donde ponga el comodín, si al final encuentra el patrón que se está buscando, entonces se reconocerá el registro, el comando quedaría:

Select nombre, direccion from tels where direccion like '%370%'

Con este comando no importa que haya antes o después del 370, si en el registro encuentra 370, entonces lo traerá:

nombre	direccion
Rafael Muñoz Gómez	Rio Churubusco # 370

En caso de querer hacer mas de una restricción se pueden usar los valores *and* u *or* para concatenar restricciones. Además de poder hacer restricciones, se puede ordenar el conjunto de datos obtenidos utilizando el comando:

Order by {campo} {asc/desc}

Veamos otra de las tablas de ejemplo que contiene mas registros, para que el query tenga mas sentido, la tabla se llama articulos

CveArt	Articulo	Precio	CveProv
1	Plato base	15.3	1
2	Plato trinche	12.2	1
3	Plato sopa	12.2	1
4	Plato postre	10.9	1
5	Vaso largo	6.5	2
6	Vaso agua	6	2
7	Vaso tequilero	4.5	2
8	Copa vino	5.7	2
9	Copa cerveza	8	2

Aunque no se ven todos los registros, se puede ver que hay varios mas que en la tabla anterior. Los datos básicamente incluyen una clave única para cada registro (CveArt), la

descripción del registro (Articulo), el precio y la clave única en la base *proveedores* del que abastece de este artículo (CveProv).

Para ordenar el grupo de datos que un query arroja, solo hay que especificar el campo por el que se quiere ordenar y si el orden será Ascendente (asc) o Descendente (desc), si no se especifica, se ordena de manera ascendente. Por ejemplo, para ordenar alfabéticamente los artículos cuyo precio sea menor o igual a 5.40, se usa:

*Select * from articulos where precio <= 5.4*
Order by articulo

CveArt	Articulo	precio	CveProv
19	Ceniceros	4.1	4
11	Cuchara cafeter	4.75	3
18	Salero y pimente	3.4	4
16	Servilletas tela	4.8	4
13	Tenedor postre	4.6	3
7	Vaso tequilero	4.5	2

En caso de que se los datos se puedan y se necesite agruparlos, es posible usando la instrucción:

Group by {campo}

Para que este comando funcione, es necesario mostrar datos que puedan ser agrupados por algún campo y que presenten solo un valor, por ejemplo: obtener la suma del precio de los diferentes artículos que cada proveedor abastece.

Select cveprov, sum(precio) from articulos group by cveprov

cveprov	SUM OF precio
1	50.6
2	30.7
3	34.05
4	12.3

en caso de que el encabezado de la columna se quiera cambiar, se puede usar la palabra clave **as**, después del campo que se quiera modificar, por ejemplo:

Select cveprov, sum(precio) as suma from articulos group by cveprov

cveprov	suma
1	50.6
2	30.7
3	34.05
4	12.3

Con esto, simplemente se le puede dar una presentación mas formal a los datos, desde el momento en que se está corriendo la aplicación. Todos los comandos que se vieron se pueden aplicar sin ningún problema combinándolos de igual forma.

SQL tiene la facilidad de poder combinar el contenido de una o mas tablas, no importa que no haya relación entre ellas, simplemente hará un producto cartesiano de los registros de las tablas, es decir, combinará todos los registros de una tabla contra los que tenga la otra u otras, sin embargo, la relación de este tipo datos casi siempre es mas útil cuando se tiene una relación entre ellos, pues así se puede llegar a un dato específico a partir de uno en otra tabla, por ejemplo, que pasaría si además del artículo y precio de nuestros datos, también quisiéramos el nombre del proveedor para los objetos con precio mayor a 6.4. En este caso, necesitamos conectar dos tablas, lo cual puede quedar:

Select a.articulo, a.precio, b.proveedor from articulos a, proveedores b
Where a.cveprov = b.cveprov
And a.precio > 6.4

articulo	precio	proveedor
Plato base	15.3	Ernesto Martínez Jiménez
Plato trinche	12.2	Ernesto Martínez Jiménez
Plato sopa	12.2	Ernesto Martínez Jiménez
Plato postre	10.9	Ernesto Martínez Jiménez
Vaso largo	6.5	Grupo Crysa
Copa cerveza	8	Grupo Crysa
Cubierto carnes	6.5	Juán Carlos Sosa Romero

En este caso, solamente se hace referencia a las dos tablas que contienen los datos de interés, los campos tienen una letra antes del nombre para que el query sepa donde buscar el campo, los nombres de tabla tienen una letra después para hacer un alias del nombre para no tener que repetirlo, la condición inicial es para pedir que de todos los cruces posibles del producto cartesiano, solo se contemplen aquellos en los que las claves del proveedor son iguales, al final, solo se incluye la última condición pedida, que se refiere a que solo los artículos con precio mayor a 6.4 se presenten. Esta forma resulta muy útil pues en un caso de base de datos mas común (dbase, fox pro, etc), para poder hacer un ligado así,

tendríamos que leer una tabla y después del datos que se obtenga ir a revisar otra, lo cual no siempre es tan sencillo.

Hasta aquí en lo que se refiere a consultas a datos de la base, todo lo anterior puede verificarse en el ejecutable incluido en el disco que acompaña a este proyecto cuyo nombre es project2 exe, las tablas también van incluidas y se pueden utilizar para las pruebas pertinentes, en caso de que se necesite incluir alguna otra tabla, lo único que se debe hacer es que desde algún programa que pueda grabar con formato de tablas de paradox se genere el grupo de datos que interese y guardarlo donde se encuentras las demás tablas, pudiendo entonces hacer referencias al nombre de la tabla como se hace en los ejercicios de este anexo

Existen otros comandos que también son estándar en SQL pero cuya finalidad es la de hacer modificaciones a los datos que se presentan, las modificaciones que se pueden hacer en un catálogo siempre han sido, altas, bajas y cambios, así que estas son las modificaciones que se mostrarán:

Insert. En caso de que se requiera una alta a la base, se debe utilizar este comando, su sintaxis es la siguiente:

Insert into {tabla} {(campos)} values ({valores})

Este es el caso mas general para insertar algún registro a la tabla, se debe especificar el nombre de la *tabla* de datos que se desea modificar, después, en caso de que no sean todos o se vayan a cargar en un orden diferente, los *campos* que se desea afectar ya que no siempre es necesario llenar todas las columnas de un registro, finalmente, se ingresarán los *valores* que se desea incluir en los campos.

Cada vez que se ejecute este query se hará una inserción a la base de datos, en Delphi hay que tener cuidado con el cómo usar este tipo de query ya que estos se ejecutan, mientras que las consultas se abren. En código, un componente TQuery que tenga una modificación a la base de datos deberá utilizarse así:

```
Query1.close;
Query1.execsql;
```

Un ejemplo del *insert* puede ser:

Insert into proveedores (cveprov, telefono) values (6, '5515-8877')

Y así se tendrá cargado un nuevo proveedor, aunque sin nombre, pero con su teléfono y clave única.

Existe un formato mas para el uso del insert y es que cuando se puede hacer un select para llenar el insert, es mas conveniente que usar la instrucción uno por uno, así el formato es:

Insert into proveedores
Values select cve, nombre, tel from proveedores_viejos

Incluyendo de un solo paso todos los registros que estén en la tabla: proveedores_viejos.

Delete. Cuando se quiere eliminar algún registro o grupo de registros que estén en alguna tabla, se utiliza este comando, tiene todas las capacidades de discriminación que el *select*, solo que este borrará la información, de hecho, una forma de verificar que se va a borrar, es hacer el *select* de los datos con las mismas condiciones que el *delete*. Su sintaxis es:

Delete from {tabla} where {condicion}

Con esto solo debemos especificar la tabla a borrar y la condición bajo la cual se deben borrar los registros, en caso de que no haya condición, todos los registros se eliminarán. Un ejemplo puede ser:

Delete from articulo where precio < 4.80

Borrando con esto todos los artículos con precio menor a 4.80 pesos.

Update. Que por supuesto sirve para hacer cambios en los datos que ya están registrados en alguna tabla, sin necesidad de incluir algún registro nuevo o borrar uno existente. Su sintaxis es:

Update {tabla} set {campos = valores} where {condicion}

En este caso también es posible hacer modificaciones en grupo, siempre que se cumpla alguna condición, además es necesario especificar los valores que deberán incluirse en cada campo. Este valor puede ser traído desde alguna base de datos o puede tener alguna operación con otro campo de la misma base u otra, en la condición se debe especificar para cuales registros se hará la actualización. Por ejemplo. Si queremos modificar el total de la venta incluyéndole el I.V.A a todos los registros del proveedor 140, podemos hacer lo siguiente:

*Update ventas set monto = monto * 1.15 where proveedor = 140*

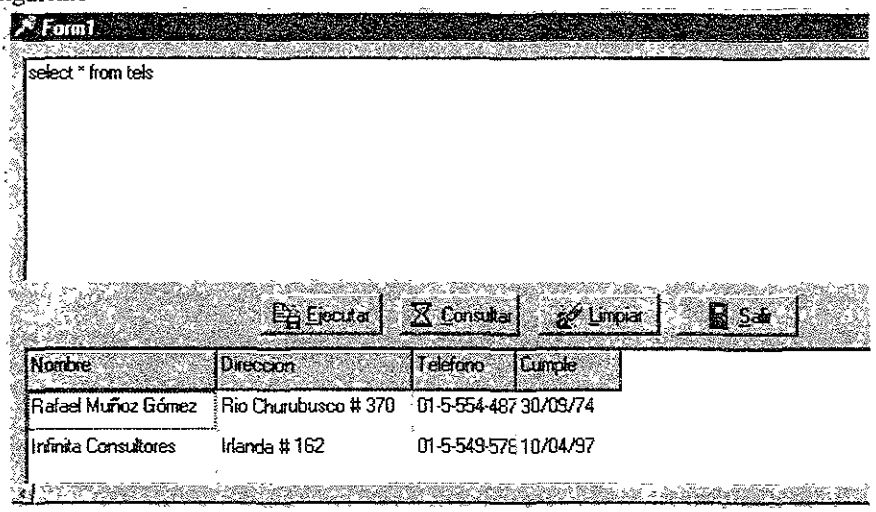
Si necesitamos modificar un dato de una tabla (ventas) dependiendo de un campo que está en otra tabla (proveedores), podríamos hacer lo siguiente.

Update ventas a, proveedores b set
*a.montocomision = a.monto * b.porcentajecomision*
where a.proveedor = b.proveedor

Con esto, para cada clave de proveedor en ventas (a), se busca el respectivo porcentaje de comisión en proveedores (b), haciendo que solo para los casos que caen en el cruce de claves de proveedor se actualice el campo deseado

Aplicación en Delphi

Existe una aplicación incluida en el proyecto que puede servir para practicar los comandos que en este anexo se vieron, la aplicación se llama project2.exe y su apariencia es la siguiente



Su funcionamiento es muy simple y en caso de no contar con una herramienta especializada para la interpretación de querys, manejo de bases de datos o algo mas complicado, puede servir para practicar los comandos y familiarizarse con la forma de obtener datos vía SQL. Lo que hacen los botones es lo siguiente:

- Ejecutar: Se encarga de llevar a cabo todas las operaciones de querys que tengan que ver con modificaciones a los datos, desde aquí no podrá correrse un *select*, pues el comando que asocia este botón al query es: execsql.
- Consultar: En caso de que el comando sea de consulta de datos (select), entonces este es el botón que deberá presionarse.
- Limpiar: Con esto se limpiará el área de resultados, que es la que se encuentra en la parte de abajo, separada como una matriz.
- Salir: Termina la aplicación.

El cuadro de texto que se tiene en la parte de arriba es para poder capturar los comandos que se desea ejecutar y después solo hay que decidir el botón que deberá presionarse.

Los fuentes, así como el ejecutable de esta pantalla de muestra se encuentran incluidos en el disco que acompaña a este proyecto

Conclusión.

Al final del proceso de elaboración de este proyecto, aún quedan algunas cosas que no pueden ser redactadas y puestas en un trabajo escrito, sin embargo, existe una muestra de lo que la gente egresada de la UNAM puede conseguir y aportar al país en una época cada vez mas difícil para estos, pues, el desprestigio y algunas diferencias de opinión entre los dirigentes de algunas compañías importantes a nivel mundial y algunos representantes estudiantiles han cerrado muchas puertas, aún antes de que se intenten tocar por algún miembro de esta comunidad.

El desarrollo del proyecto a algunos les servirá y algunos otros, tal vez nisiquiera lo consideren interesante, sin embargo el objetivo del proyecto es dejar un punto de apoyo para que las siguientes generaciones tengan al menos un rumbo por donde buscar, un punto donde empezar y un ejemplo de alguien que encontró un lugar entre los desarrolladores de software, a pesar de no haber tenido el conocimiento de las herramientas necesarias desde el principio, principal factor para ser rechazado en las compañías desarrolladoras mas importantes, pues la curva de aprendizaje de estas no siempre es tan corta como todo el mundo quisiera, esa es la parte fundamental que este trabajo atacó y que al final me parece haber cubierto satisfactoriamente, al menos, en el sentido de haber dejado en papel la experiencia de aprendizaje y utilización de las herramientas.

La búsqueda de cada uno, debe llevar un rumbo bien específico para obtener los resultados esperados y debe además, aportar soluciones, perspectivas, procedimientos y metas nuevas y diferentes, debe tender hacia nuevos horizontes y sobre todo a mejorar las oportunidades, que la gente egresada de la Universidad pueda tener. Los egresados tenemos el derecho y la obligación de mostrar que la Universidad no es solo grande en población, sino también en capacidad. Con este proyecto pretendo aportar una herramienta que los universitarios puedan utilizar muy bien, pues creo que no necesitamos de grandes clases, sino mas bien de un objetivo bien claro, ya que el camino para llegar, seguramente, puede ser creado por nosotros mismos.

Del objetivo especificado en el inicio del trabajo me parece que queda cubierto en este punto, hasta ahora se mostró como utilizar la teoría de probabilidad y estadística, que puede ser aprendida en la carrera de MAC, y cómo manipularla e incluirla en un programa hecho en Delphi.

Durante el capítulo 1 se mencionó la teoría indispensable para familiarizar al programador en Pascal con el ambiente de Delphi, así mismo, se indica el uso y funcionamiento de algunos componentes escenciales, estos componentes son los que generalmente se utilizan en un desarrollo común, estos permanecen a lo largo de las versiones mas recientes del lenguaje, por lo que pueden ser utilizados en la generación de una aplicación sencilla, pero completa sin temor a quedar "incompatible". En las nuevas versiones de Delphi existen muchos componentes que están pensados para desarrollar en un ambiente de Internet, se recomienda ampliamente revisarlos y probarlos, pues seguramente

si se está pensando incluir tecnología de Internet en algún desarrollo, Delphi ahora puede ser una solución bastante robusta y sencilla de implementar

Durante los capítulos 2 y 3, el contenido muestra como mezclar la teoría, que para este caso es matemática, con Delphi. En general se muestra al principio de cada sección, la teoría que se puede encontrar en cualquier libro de probabilidad y estadística y posteriormente, se muestra la *codificación en delphi*[23], de manera que pensando en una aplicación propia o en hacer modificaciones a la que se desarrolló en este proyecto, esta sección de codificación puede ser muy útil, pues dice que hace la función en Delphi, para conseguir lo que la teoría dice.

Los anexos solo tienen la intención de mostrar todos los detalles de código, pues en estos se expresan las definiciones de las variables, funciones, interfaces, etc. que hace Delphi para que el *dll* pueda correr, de modo que para poder tener una visión global de toda la aplicación es necesario tener todos los detalles del código. Los listados de los programas que se presentan son los originales y no tienen comentarios, salvo los que se pusieron en Delphi, de modo que si se capturan estos, se debe tener una aplicación funcional, igual a la que se entrega con el resto del trabajo.

Los anexos 3 y 4, pretenden ampliar el apoyo que se puede encontrar en el presente trabajo, pues se muestra la teoría del lenguaje SQL para hacer peticiones a una base de datos que lo soporte. Teniendo con esto una herramienta que se puede utilizar para consultar datos, agregarlos, modificarlos o eliminarlos, que son básicamente las operaciones que se pueden realizar en una base de datos común. Además se pretende familiarizar al lector con la terminología para programación en Windows y algunos conceptos básicos de Programación Orientada a Objetos, de modo que si no se está acostumbrado a los eventos y propiedades de Windows, estos anexos puedan ser de utilidad para tener un punto de partida en la exploración de estos lenguajes.

Espero que el tiempo y esfuerzo dedicado a este trabajo sea aprovechado por alguien de alguna manera, pues de él, se pueden obtener varios provechos. También quiero que el que no esté seguro de lo que pasará o de lo que está haciendo, tome muy en cuenta este texto, porque el vivir día a día en la experiencia laboral, me ha mostrado el valor que puede generar la UNAM en competencia y colaboración con egresados de otras universidades, este valor en lo que hacemos, por supuesto es inmenso.

Rafael Muñoz Gómez

[23] De hecho en cada rutina que se codificó se puede buscar la sección con este nombre, que ahí se encontrará el código además de una explicación de cómo se resolvió el planteamiento teórico al momento de codificarlo en Delphi.

Bibliografía

Modelos y Simulación
MariCarmen González Videgaray
Ediciones Acatlán
Universidad Nacional Autónoma de México
Escuela de Estudios Profesionales "Acatlán".

Probabilidad y Aplicaciones Estadísticas
Paul L. Meyer
Addison – Wesley Iberoamericana
Wilmington, Delaware, E.U.A., 1992

Estadística Matemática con Aplicaciones
William Mendenhall
Dennis D. Wackerly
Richard L. Scheaffer
Grupo Editorial Iberoamérica
México, D.F., Mayo de 1995

Probabilidad y Estadística Aplicaciones y Métodos
George C. Canavos
Mc.Graw Hill
México, D.F. 1988

Introducción a la Investigación de Operaciones
Frederick S. Hillier
Gerald J. Lieberman
Mc.Graw Hill
México, D.F. 1991

Using Delphi 3
Todd Miller & David Powell y otros
QUE Corporation.

Métodos Numéricos
Rafael Iriarte V. Balderrama
Trillas : UNAM
Facultad de Ingeniería, 1990.

http://www.xmission.com/~renates/delphi.html
Renate Schaaf
1252N 400E #3
Logan, UT 84321
CompuServe: 71031,2774
Internet: renates@xmission.com schaaf@math.usu.edu