UNIVERSIDAD CENTRAL DE VENEZUELA

FACULTAD DE INGENIERIA

DEPARTAMENTO DE INVESTIGACIÓN

DE OPERACIONES Y COMPUTACIÓN

Tutorial de Borland C++ Builder 2009

Prof. Belzyt C. González Guerra

Caracas 2009

INDICE DE CONTENIDOS

CAPITULO I: PROGRAMACION VISUAL ELEMENTOS BASICOS

1.- INTRODUCCION

Los nuevos sistemas de información son costosos en tiempo y recursos, la solución moderna de sistemas de información exigen nuevas herramientas y metodologías para resolver rápida, económica y eficientemente los problemas de información planteados por las organizaciones.

Aún mas, el potencial del hardware no es aprovechado exhaustivamente y existe un considerable retraso con el software y sus aplicaciones, generando lo que se conoce como "crisis del software".

En programación tradicional, modular o estructurada un programa describe una serie de pasos a ser realizados para la solución de un problema, es decir es un algoritmo.

En **Programación Orientada a Objetos (OOP por sus siglas en inglés)** un programa es considerado como un sistema de objetos interactuando entre sí, ambientes de desarrollo visuales facilitan aún más la construcción de programas y solución de problemas, porque permiten abstraer al ingeniero de software de la interfase gráfica del problema, que constituye más del 60% del código normal de un programa.

Es decir, en programación modular o estructurada un problema sencillo de información es descompuesto en una serie de módulos (llamados procedimientos o funciones) donde cada uno de ellos realiza una tarea específica, por ejemplo uno de ellos captura los datos, otro resuelve operaciones, etc.

En OOP todo problema, aún aquellos sencillos de información, se consideran y resuelven como módulos de código gigante (clase) que contiene todo el código necesario (variables, procedimientos, funciones, interfaces, etc.) para solucionar el problema.

En programación visual (que también es heredera de OOP), la interfase con el usuario (pantallas) son generadas por el propio compilador y el ingeniero de software solo se concentra en resolver el problema planteado.

C++Builder, es un compilador que permite usar cualquiera de los tres enfoques en la solución de problemas de información que puedan y deban ser resueltos empleando el computador y el lenguaje.

Para propósitos de este libro usaremos el tercer enfoque, es decir programación en ambientes visuales y usando el lenguaje de programación *C++Builder*.

2.- MODELO DE SOLUCION

En general un problema de información es posible entenderlo, analizarlo y descomponerlo en todos sus componentes o partes que de una u otra manera intervienen tanto en su planteamiento como en su solución.

Una herramienta rápida que nos permite descomponer en partes un problema para su solución, es el llamado modelo de solución, este consiste de una pequeña caja que contiene los tres elementos más básicos en que se puede descomponer cualquier problema sencillo de información, estas tres partes son:

a. **Entrada**: son todos los datos que el computador necesita para resolver el problema, estos datos son almacenados internamente en la memoria del computador en las *variables de entrada*.

b. **Proceso**: son todas las operaciones, generalmente algebraicas, necesarias para solucionar el problema, generalmente esta parte del modelo es una fórmula (o igualdad matemática, Ej. X= y + 5).

c. **Salida**: es el resultado o solución del problema que generalmente se obtiene del proceso del modelo y dichos datos están almacenados en las *variables de salida*.

En resumen para todo problema sencillo de información es necesario plantearse las siguientes preguntas:

- ¿Que datos necesita conocer el computador para resolver el problema y en cuales variables de entrada se van a almacenar?
- ¿Que procesos u operaciones debe realizar el computador para resolver el problema planteado?
- ¿Que información o variables de salida se van a mostrar en pantalla para responder al problema planteado originalmente?

Como nota importante no confundir los términos datos, variables e información;

Datos se refiere a información en bruto, no procesada ni catalogada, por ejemplo "Caracas", "calle primera # 213","15 años", " 2.520.000Bs.", etc.

Variables es el nombre de una localidad o dirección interna en la memoria del computador donde se almacenan los datos, ejemplo de variables para los casos anteriores, CIUDAD, DIRECCION, EDAD, SUELDO, ETC.

Información son datos ya procesados que resuelven un problema planteado.

EJEMPLO DE MODELO DE SOLUCION

1. Construir un modelo de solución (Entrada-Proceso-Salida) que resuelva el problema de calcular el área de un triángulo con la formula área igual a base por altura sobre dos.

Variable(s) Entrada	**Proceso u operación**	**Variable(s) salida**
BASE ALTURA	$AREA = \frac{BASE * ALTURA}{2}$	AREA

Observar para el caso de constantes fijas o conocidas (PI) no se debe dar como dato de entrada su valor sino, colocar directamente su valor dentro de la formula, en la parte de operaciones del problema.

Se debe tener en cuenta que:

- Hay problemas sencillos donde no hay datos de entrada o no hay operaciones, pero todos los datos son de salida.
- Una formula grande o muy compleja puede ser más segura y fácil de resolver, si es descompuesta y resuelta en partes, juntando al final los parciales para obtener el resultado final.
- Un problema puede tener más de una solución correcta.
- El problema no esta suficientemente explicado o enunciado, entonces, estudiarlo, analizarlo y construirlo de manera genérica.

PROBLEMAS SUGERIDOS:

Construir los modelos de solución de los siguientes problemas:

2. Convertir la edad en años de una persona a meses.
3. Convertir Bolívares a dólares.
4. Calcular el área de un circulo con la formula: $AREA = PI * RADIO^2$
5. Evaluar la función $Y = 5X^2 - 3X + 2$ para cualquier valor de x.
6. Convertir millas a kilómetros (caso normal)
7. Convertir 125 metros a centímetros (no necesita entradas)
8. Se calcula que en promedio hay 4 nidos en cada árbol en la UCV, también se calcula que en cada nido existen un promedio de 5 pájaros, se pide calcular la cantidad total de nidos y de pájaros en los 227 árboles que existen en la UCV. (no necesita entradas)
9. La Sra. López y sus 8 hijos solo compran una vez al mes en un conocido supermercado. En dicha tienda el kilogramo de caraota cuesta 800,75 Bs., el paquete de harina pan cuesta 600,55 Bs. y el paquete de café vale 400.25 Bs., si solo compran de estos tres productos para su mercado, calcular su gasto total. (problema no claro)
10. Capturar y desplegar los cinco datos mas importantes de un automóvil (no necesita operaciones)
11. La distancia Caracas – la Guaira es de 52 kilómetros. Si un automóvil la recorre a una velocidad constante de 80 Km. por hora, cuanto tiempo tarda en llegar. (dos maneras correctas de resolverlo).
12. Evaluar la función $y = 3x^2 + 2x - 5$ para cualquier valor de x. (caso normal).
13. Evaluar la función $y = -5x^3 - 3x^2 + 8$ para cuando x vale 4. (no necesita entradas).

3.- C++ BUILDER VARIABLES

Identificadores son conjuntos de letras y/o números que se utilizan para simbolizar todos los elementos que, en un programa, son definibles por el usuario del mismo, como son las

variables: donde se almacenan los datos, funciones: pequeños módulos con código, etiquetas, clases, objetos, etc.

En *C++Builder* un identificador es una palabra compuesta de letras y/o números de hasta 32 caracteres significativos, ***empezando siempre con una letra***.

Una variable se define como un identificador que se utiliza para almacenar todos los datos generados durante la ejecución de un programa.

Existen ciertas reglas en cuanto a variables:

- Deben ser claras y con referencia directa al problema.
- No debe haber espacios en blanco, ni símbolos extraños en ellas.
- Se puede usar combinaciones de letras y números comenzando siempre con una letra.
- No deben ser palabras reservadas del lenguaje.
- Puede usar el _ para concatenar palabras.
- Las mayúsculas y las minúsculas se consideran caracteres diferentes (Edad ≠ edad).

Ejemplos de buenas variables:

Nombre, Edad, SdoDiario, Ing_Mensual, Perímetro, Calif1, etc.

Comentarios son aquellos trozos de texto que permiten explicar el funcionamiento de un pedazo de código y que son transparentes al compilador, es decir, al momento de compilar el programa, estos comentarios no son tomados en cuenta.

- Para comentarios de una sola línea, anteponga // al comentario
- Para comentarios de más de una línea comience el comentario con /* y finalícelo con */.

```
// Este es un comentario de una línea
/* Este es
   un comentario que
   ocupa más
   de una línea*/
```

4.- TIPOS DE DATOS

A toda variable que se use en un programa, se le debe asociar (generalmente al principio del programa) un tipo de dato específico.

Un tipo de dato define todo el posible rango de valores que una variable puede tomar al momento de ejecución del programa y a lo largo de toda la vida útil del propio programa.

Los tipos de datos más comunes en *C++Builder* son:

Tipo	Dato	Tamaño	Rango
Enteros	unsigned char	8 bits	0 a 255
	char	8 bits	-128 a 127
	unsigned int	16 bits	0 a 65.535
	int	16 bits	-32.768 a 32.767
	unsigned long	32 bits	0 a 4.294.967.295
	long	32 bits	-2.147.483.648 a 2.147.483.647
Definido	enum	16 bits	-2.147.483.648 a 2.147.483.647
Reales	float	32 bits	$3,4 \times 10^{-38}$ a $3,4 \times 10^{+38}$(6 dec)
	double	64 bits	$1,7 \times 10^{-308}$ a $1,7 \times 10^{+308}$(15 dec)
	long double	80 bits	$3,4 \times 10^{-4932}$ a $1,1 \times 10^{+4932}$
Alfanumérico	AnsiString	***	cadena de caracteres
Booleano	bool	8 bits	true, false

** AnsiString no es propiamente un tipo de dato, sino una **clase** que se especializa en el almacenamiento y manipulación de datos de tipo alfanumérico, es decir, como clase ya contiene toda una serie de métodos (procedimientos y funciones) que pueden usar directamente las variables (objetos) que se declaren de tipo AnsiString, como en el siguiente ejemplo:

```
// Zona de declaración
                AnsiString alfa, beta;
                int a;
/* carga normal de variables alfanuméricas*/
                alfa = "UCV";
                beta = "50";
```

```
/* procesos con las variables considerándolas como objetos de la clase AnsiString */
        alfa = alfa.LowerCase();
/*primero las convierte a minúsculas y luego se cargan a la misma variable.*/
        a = beta.ToInt();
//convierte el contenido a enteros.
```

El resto de métodos, se deben buscar en la ayuda del *C++Builder* (solo abrir la carpeta llamada "referencia del programador", pedir índex y luego escribir AnsiString).

5.- OPERADORES ARITMETICOS

Un operador es un símbolo especial que indica al compilador que debe efectuar una operación matemática o lógica.

C++Builder reconoce los siguientes operadores aritméticos:

Operador	**Operación**
+	Suma
-	Resta
*	Multiplicación
/	División
%	Residuo o Módulo
++	Incremento
- -	Decremento

Como notas importantes:

- En problemas de división entre enteros, *C++* trunca la parte residual, es decir:

```
// Zona de declaración de variables
        float a;
// Zona de operaciones
        a = 10 / 4;
// Zona de resultados
        mostrar a; //En pantalla sale (2.000000)
```

El problema no es el tipo **float**, sino que por definición de la división entre enteros *C++* siempre trunca la parte residual, mas adelante se indica como se resolverá este problema.

El operador (%) devuelve el residuo entero de una división entre enteros, ejemplo;

```
// Zona de declaración
        int alfa;
// Zona de operaciones
        alfa = 23 % 4;
// Zona de resultados
        mostrar alfa; //El resultado en pantalla es 3
```

Otro ejemplo;

```
        alfa = 108 % 10;
        mostrar alfa; //El resultado en pantalla es 8
```

Para resolver los problemas de potencias y raíces, se usan ciertas instrucciones especiales que proporciona el lenguaje, llamadas funciones de biblioteca, en *C++* existe un conjunto de librerías de instrucciones o funciones. Una lista de las funciones de biblioteca más utilizadas se muestra a continuación.

Función	Descripción	Tipo del Argumento	Tipo del Resultado	Librería Asociada
acos (x)	Arco Coseno de x	Real	Real	math.h
asin (x)	Arco Seno de x	Real	Real	math.h
atan (x)	Arco Tangente de x	Real	Real	math.h
ceil (x)	Redondea x al entero más pequeño no menor que x	Real	Entero	math.h
cos (x)	Coseno de x (en radianes)	Real	Real	math.h
exp (x)	Función Exponencial e^x	Real	Real	math.h
fabs (x)	Valor Absoluto de x	Real	Real	math.h
floor (x)	Redondea x al entero más grande no mayor que x	Real	Entero	math.h
fmod(x)	Residuo de y/x como real	Real	Real	math.h
log (x)	Logaritmo Natural de x (base e)	Real	Real	math.h
log10 (x)	Logaritmo de x (base 10)	Real	Real	math.h
pow (x,y)	X elevado a la potencia y (x^y)	Real	Real	math.h
rand ( )	Genera un número entero aleatoriamente	Vacío	Entero	stdlib.h
random (x)	Genera un número aleatorio entero entre 0 y x-1	Entero	Entero	stdlib.h
randomize(x)	Inicializa el generador de números aleatorios con la semilla tomada del reloj	Vacío	Vacío	stdlib.h time.h

Función	Descripción	Tipo del Argumento	Tipo del Resultado	Librería Asociada
sin (x)	Seno de x (en radianes)	Real	Real	math.h
sqrt (x)	Raíz Cuadrada de x	Real	Real	math.h
tan (x)	Tangente de x (en radianes)	Real	Real	math.h

Recordar que todas las funciones reciben uno o más datos o valores y pueden regresar un resultado, un ejemplo de una de estas funciones matemáticas es:

```
#include <math.h>
double pow(double base, double exp);
```

Esta función ocupa dos valores o datos (base y exp) ambos de tipo double, y regresa un resultado también de tipo double, ejemplo;

- resolver el problema de calcular 5^3

```
#include <math.h>
// Zona de declaración de variables
        double base, exponente, potencia;
// Zona de asignación o carga o inicialización de variables
        base = 5;
        exponente = 3;
// Zona de operaciones
        potencia = pow (base, exponente);
// Zona de Resultados
        mostrar potencia; //El resultado en pantalla es 125.000000000
```

Para resolver el problema de raíces, se aprovecha una de las más elementales y conocida de las leyes de exponentes que dice:

$$\sqrt[n]{a^m} = a^{m/n}$$

Es decir una raíz cualquiera se puede transformar a una potencia con un exponente fraccionario.

Ejemplo:

problema $y = 3\sqrt{x}$ esto es equivalente a $y = 3 * x^{1/2}$ entonces:

```
// Usando la función pow
        y= 3*pow(x, 0.5);
```

- En este ejemplo se esta dando por supuesto que no interesa el tipo de dato que requiere la función pow( ) para trabajar correctamente

6.- C++ BUILDER OPERADOR CAST

Se puede forzar un dato, variable o una expresión a convertirse o cambiarse a un nuevo tipo de dato.

El operador **cast** realiza este proceso, es decir convierte datos, variables o expresiones a un nuevo tipo de dato, su formato es:

Var1 = (nvotipo) dato, var, exp;

Ejemplo:

```
// declaración
        int alfa;
        float beta;
// Asignación
        alfa=20;
// Cambio de tipo
        beta = (float) alfa;
```

- Si se convierte un float a int, se trunca a la parte entera del número.
- Como nota importante este operador resuelve dos problemas:
  a. El de la división entre enteros.
  b. El tipo de dato especifico que requieren las funciones.

Ejemplos:

```
// Declaración
        float alfa;
// Operación
        alfa = (float)23/5;
// Pero en estos casos es preferible
        alfa=23/5.0;
```

En toda división recordar agregar a uno de los dos valores el (.0), solo que los dos elementos sean variables entonces usar el operador **cast** con una de ellas.

7.- C++ BUILDER JERARQUIA DE OPERACIONES

El problema de no tomar en cuenta la jerarquía de los operadores al plantear y resolver una operación casi siempre conduce a resultados muchas veces equivocados como estos:

Ejemplos:

a) 2+ 3* 4 = 20 (incorrecto) = 14 (correcto)

b) si calif1 = 60 y calif2 = 80 entonces,

   promedio = calif1 + calif2 / 2 da como resultado promedio = 100

Recordar siempre, que antes de plantear una formula en un programa se deberá evaluar contra el siguiente:

Orden de operaciones:

1. Paréntesis
2. Potencias y raíces
3. Multiplicaciones y divisiones
4. Sumas y restas
5. Dos o más de la misma jerarquía u orden, entonces resolver de izquierda a derecha

Nota: Si se quiere alterar el orden normal de operaciones, entonces usar paréntesis.

Nota: Tampoco es bueno usar paréntesis de más en una operación, esto solo indica que no se evaluó bien la formula, como en el siguiente ejemplo;

$$area = (base * altura) / 2.0;$$

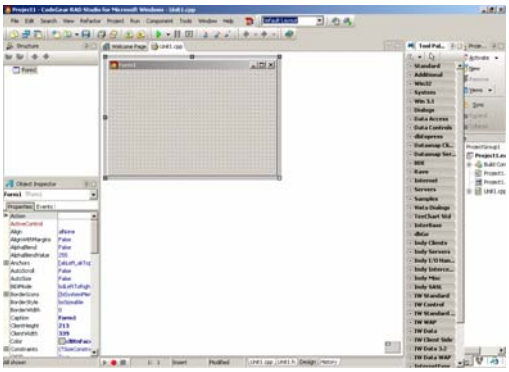
aquí los paréntesis están de más, porque por orden de operaciones, multiplicación y división tienen la misma jerarquía y entonces se resuelven de izquierda a derecha, en otras palabras ni que falten paréntesis ni que sobren paréntesis.

8.- C++ BUILDER ENTRADAS/SALIDAS EN PROGRAMACION VISUAL

Entradas o capturas de datos y salidas o despliegues de información o resultados son de los procesos más comunes en cualquier tipo de problema de información, estos procesos o instrucciones varían de acuerdo a los lenguajes y ambientes de programación a usar.

El lenguaje y ambiente de programación a utilizar, es de tipo visual, y muchos de los problemas asociados a entradas y salidas se encuentran ya resueltos por el propio compilador.

El ambiente de construcción de programas a usar, es el siguiente:



- Se cargan en pantalla ejecutando el *C++builder*, que se encuentra en la barra de inicio de windows.

Sus elementos básicos son:

1.- La barra de menús (file, edit, etc.);

2.- La barra de herramientas (icono de grabar, run, forma, etc.)

3.- La barra de componentes

4.- El Inspector de Objetos

5.- La forma activa o principal

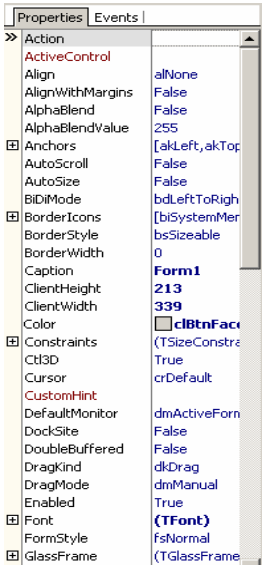
9.- LA FORMA ACTIVA O PRINCIPAL

Es sobre este formulario donde se construye el programa y este formulario se convierte en ventana al momento de ejecutarse el programa.

Es decir será la primera ventana que el usuario ve al momento de ejecutarse el programa, su nombre es ***Form1***.

Este formulario o ventana es un objeto de C++, y como todos los objetos de C++, el formulario o ventana tiene asociados propiedades y eventos.

Propiedades son todas las características particulares que diferencian un objeto de otro objeto, las propiedades o características mas comunes son forma, tamaño, color, etc., para objetos en *C++*, estas propiedades se modifican o individualizan usando el Inspector de Objetos, que es la parte del programa que las contiene.



También se pueden modificar las propiedades dentro de un programa, usando instrucciones apropiadas, que llevan el siguiente formato:

NombreObjeto->Propiedad = nuevovalor;

ej.; Form2->Color = clRed;

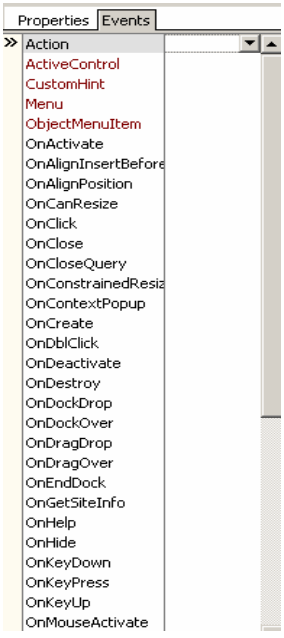
Eventos, son todos aquellos sucesos de carácter externo que afectan o llaman la atención del objeto, para estos casos el formulario o ventana:

1. Debe tener capacidad de detectar el evento
2. Aun más importante debe tener capacidad de reaccionar y emitir una respuesta, mensaje o conducta apropiada al evento detectado.

Evento es que se produce cuando el usuario interactúa con los objetos de la interfaz; por ejemplo que el usuario, pulse el objeto tecla ESC, o haga click derecho con el objeto ratón en alguna parte de la ventana, etc., es en estos casos, cuando la ventana detecta un evento de estos, el programa deberá responder de manera apropiada.

Esta respuesta no es automática, es la serie de instrucciones de lenguaje (o programa) que los ingenieros de software diseñan (o programan) la que producen la respuesta, en otras palabras son los eventos quienes contendrán los programas.

Es también el Inspector de Objetos, quien contiene todos los posibles eventos asociados al formulario.



Para los primeros programas en *C++Builder*, sólo se usaran propiedades sencillas como *color*, *font*, etc. de *Form1*, y no se usan, de momento los eventos que puede detectar *Form1*.

10.- PROGRAMAS, FORMAS Y COMPONENTES

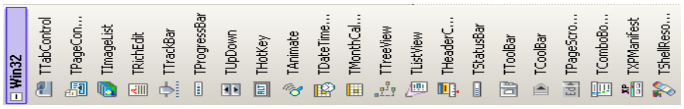
Un programa en *C++Builder*, no es mas que una o más formularios o ventanas, donde cada uno de ellos contiene elementos u objetos especiales llamados componentes, dichos componentes *C++Builder* los proporciona a través de la barra de componentes.

Es decir toda la interfase que se quiera manejar con el usuario del programa, consiste en una colección de componentes agrupados en un formulario o ventana.

La colección de componentes que pone a nuestra disposición *C++Builder* están agrupados en diversos folder o pestañas en la barra de componentes, estas categorías contienen, al menos, los siguientes componentes:

Standard: MainMenu, PopupMenu, Label, Edit, Memo, Button, CheckBox, RadioButton, ListBox, ComboBox, ScrollBar, GroupBox, RadioGroup, Panel.

Win32: TabControl, PageControl, TreeView, ListView, ImageList, HeaderControl, RichEdit, StatusBar, TrackBar, ProgressBar, UpDown, HotKey.



Additional: BitBtn, SpeedButton, MaskEdit, StringGrid, DrawGrid, Image, Shape, Bevel, ScrollBox.

Win31: DBLookupList, DBLookupCombo, TabSet, Outline, Header, TabbedNotebook, NoteBook.

Dialogs: OpenDialog, SaveDialog, FontDialog, ColorDialog, PrintDialog, PrinterSetupDialog, FindDialog, ReplaceDialog.

System: Timer, PaintBox, FileListBox, DirectoryListBox, DriveComboBox, FilterComboBox, MediaPlayer, OleContainer, Ddeclientconv, DdclientItem, Ddeserverconv, DdeserverItem.

Para incorporar un componente a un formulario sólo basta seleccionarlo con un click del botón izquierdo en su icono y luego colocar el cursor dentro de la forma en el lugar donde se quiere que aparezca y volver a hacer un click del botón izquierdo.

Los componentes son objetos de *C++Builder* y también tienen asociados propiedades y eventos, tales como los tiene el formulario principal, solo que existen pequeñas variaciones en cuanto a sus propiedades y eventos propios con respecto a *Form1*.

Recordar además, que es el Inspector de Objetos, en primera instancia, quien permite asociar o modificar propiedades específicas tanto a un formulario como a un componente.

Ya en segunda instancia, las propiedades de formularios y componentes se pueden modificar directamente en el código dentro de un programa, usando instrucciones como las ya descritas en párrafos anteriores.

Analizaremos ahora los dos primeros componentes, que también se usaran para construir o diseñar nuestro primer programa en *C++Builder*.

11.- COMPONENTE Label (Standard)

Este componente se utiliza para desplegar textos o mensajes estáticos dentro de los formularios, textos tales como encabezados, solicitud al usuario del programa para que proporcione algún dato o información (edad, dame sueldo, etc.).

También es un objeto en *C++Builder* y por tanto tiene asociados sus propias propiedades y eventos, al mismo tiempo como se está usando dentro del objeto *Form1*, muchas propiedades que se definan para el objeto *Form1*, el objeto *Label1* las va a heredar.

Si bien es cierto que el objeto se llama *Label*, cuando se ponen dentro de un formulario C++Builder los va numerando automáticamente, si se ponen tres Label en *Form1*, ellos se llaman o programan con *Label1*, *Label2*, *Label3*.

La propiedad *Caption*, es la que lleva el contenido del mensaje que se quiere desplegar en la pantalla, solo click izquierdo a un lado de la propiedad *Caption* en el Inspector de Objetos, teniendo seleccionada la caja *Label1* en la forma y escribir el texto indicado.

12.- COMPONENTE  *Button(Standard)*

Es el componente principal de la forma, contiene el código principal del programa y su activación por el usuario provoca que se realicen los principales procesos del problema planteado (aquí es donde se capturan datos, se realizan operaciones, etc.).

De este componente se maneja su propiedad *Caption* para etiquetarlo con la palabra "OK" o "ACEPTAR", y su evento *OnClick* para activarlo. Es en dicho evento donde se construye el código del programa principal.

Recordar que aunque no es un componente necesario en los programas, ya que el código se puede asociar o pegar a cualquier evento de cualquier formulario, o componente del programa, Microsoft ya acostumbro a todos los usuarios al botón OK.

Este botón también puede activar su evento *OnClick*, cuando el usuario presione la tecla <ENTER>, solo poner la propiedad *Default* en true, en este caso el botón de órdenes, se le conoce como botón de default.

Igualmente puede activar su evento *OnClick* cuando el usuario, presione la tecla <ESC>, solo poner la propiedad *Cancel* en true, a este caso se le conoce como "CANCEL BUTTON".

EJERCICIO

Construir un primer programa que consiste en un formulario que contenga los cinco datos más importantes de un automóvil, y uno de esos datos sólo deberá aparecer cuando el usuario haga click en el botón de ejecución o de orden o de OK.

Para este programa se necesita, un formulario, dos componentes *Label* para encabezados apropiados al problema, 10 componentes *Label* para textos y datos, un componente *Button*.

Para el último componente *Label* su propiedad *Caption* se dejara en blanco o vacía para que sea el usuario del programa quien lo cargue al apretar el botón de órdenes o botón de OK.

Recordar que *C++Builder* va numerando automáticamente todos los componentes en el programa en forma consecutiva, es decir al finalizar el diseño del programa se tendrán, los componentes *Form1*, *Label1*, *Label2*, *Label3*,..*Button1*.

El procedimiento completo para crear y ejecutar el programa es:

1. Cargar *C++Builder*.
2. Al cargarlo ya estará en la pantalla el primer formulario (*Form1*).
3. Antes de poner el primer componente usar la opción, File Save Project As, aparece la siguiente ventana:

Donde se deberá seleccionar primero, el icono de nuevo directorio (arriba a la derecha y tiene una carpeta con rayitos), esto es, para crear un nuevo directorio donde quedará guardado o almacenado el programa. En cuanto se crea el nuevo directorio, sobrescribir la palabra "new folder" que aparece, con el nombre que tendrá el directorio donde quedara almacenado el programa, escribir por ejemplo "programa

uno", al terminar de sobrescribir, la palabra "programa uno" apretar tecla <ENTER> y esperar un momento a que se cree el directorio.

Ya creado y renombrado el directorio, observar que en la parte inferior de la ventana el programa ya tiene el nombre de "Unit1.cpp", a un lado está una caja o botón de "Abrir", mismo que se deberá apretar y después usar en la misma parte un botón llamado "Guardar" para almacenar "Unit1.cpp" y otra vez usar otro botón "Guardar" para almacenar "Project1.mak".

4. Ahora ya que se tiene *Form1* en pantalla, recordar que se pueden modificar sus propiedades como *Color*, *Font*, etc. usando el Inspector de Objetos que está a un lado del formulario (se sugiere practicar un poco esto), los cambios que se hacen en el Inspector de Objetos se van reflejando automáticamente en la forma en pantalla y también en la ventana que el usuario verá al ejecutarse el programa.
5. Ahora se selecciona con un click el componente llamado *Label* en la barra de componentes y luego poner el cursor dentro del formulario en el lugar donde se quiera que aparezca el componente, y volver a hacer click con el ratón para que aparezca dicho componente en el formulario.

   Observar que el componente que esté seleccionado en el formulario (esto se puede hacer usando un click dentro del componente), se puede arrastrar para cambiarlo de lugar o posición o hacerlo mas pequeño o mas grande.

   Como nota también a recordar siempre, un componente o el propio formulario, está seleccionado, si el Inspector de Objetos lo está referenciando; es decir, el Inspector de Objetos contiene, en la parte de arriba del mismo, el nombre del componente que está seleccionado.

   Para cargar o para que despliegue un texto, el componente *Label1*, solo se debe escribir dicho texto en la cajita que esta a un lado de la propiedad *Caption* en el Inspector de Objetos por ejemplo, para el problema, escribir la palabra "AUTOS ECONOMICOS", y recordar que también este componente tiene propiedad *Font* que se puede usar para cambiar el tipo de letra que despliega el componente.

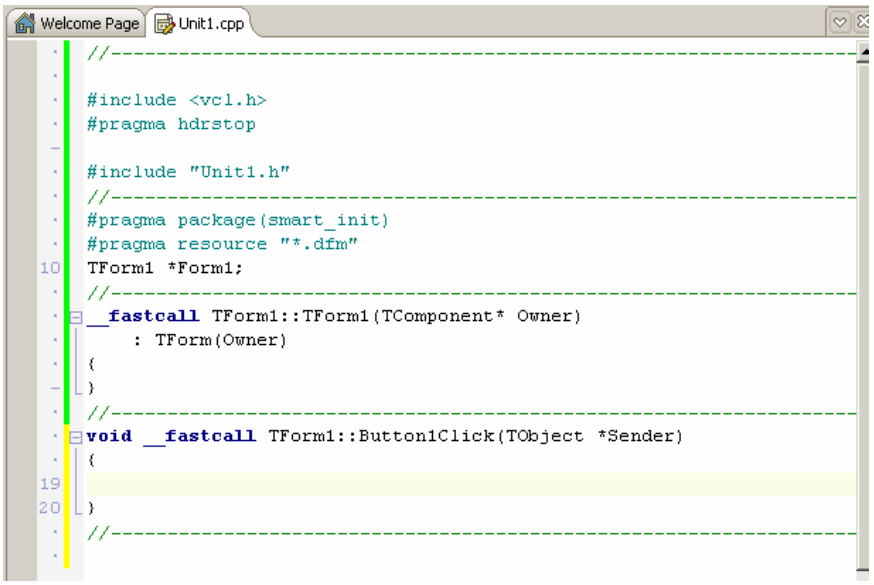
6. Repetir el procedimiento anterior hasta tener los doce componentes *Label* bien organizados en el formulario con sus textos correspondientes, solo recordar dejar el componente *Label12* con su propiedad *Caption* en blanco (usando tecla backspace).

7. Seleccionar y acomodar ahora un componente *Button* en el formulario y colocarlo en la esquina inferior derecha, en su propiedad *Caption* escribir la palabra "OK".

   Recordar que este componente, es quien contiene el código del programa y más específicamente es su evento *OnClick* quien lo contiene y quien además lo ejecuta.

   Para añadirle el código (en este ejemplo que cargue el *Caption* de *Label12* al tiempo de ejecución del programa y por decisión del propio usuario) existen dos maneras:

   En el Inspector de Objetos, hacer click en la pestaña o folder llamado *event* y ahora en lugar de propiedades el Inspector de Objetos, muestra todos los eventos que el componente puede detectar, haga click en la cajita que esta a un lado del evento llamado *OnClick* y con este paso aparece el siguiente editor de programas de *C++Builder*:



   Observe que ya trae listo el evento *Button1Onclick()*, para programarse y es entre las llaves donde se construye el programa.

   Solo escribir dentro de las llaves la instrucción:

```
Label12->Caption=" 5.000.000Bs.";
```

tenga en cuenta lo siguiente:

- Escribir dentro de las llaves
- No olvidar terminar con punto y coma (;)
- Respetar y aprender siempre la ortografía de componentes, propiedades y valores de propiedades
- Recordar que cualquier propiedad de cualquier componente se puede programar con el formato ya descrito;

```
NomComponente->NomPropiedad = NvoValor;
```

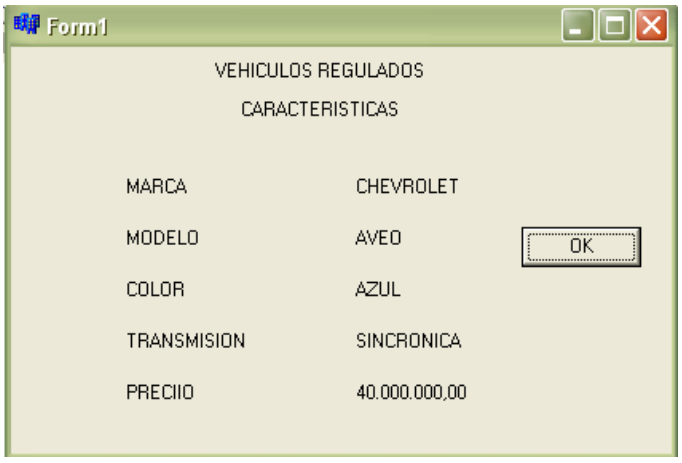
- Observar que *Caption* se carga como si fuese una variable de tipo alfanumérico, dentro de un programa.
- Observar que atrás del editor esta *Form1*, si se selecciona con un click, este procedimiento lo regresa al formulario para seguir arreglando o colocando nuevos elementos.
- Otra manera de intercambiar entre formulario y editor (ahora lo llamaremos Unit1.cpp), es decir entre *Form1* y Unit1, es usar el icono Toggle Form/Unit que se encuentra en la barra de componentes arriba y a la izquierda.

- La otra manera y mas sencilla de cargar el editor, es haciendo un dobleclick dentro del componente.

8. Ahora ya está listo el formulario o programa para ejecutarse, también existen dos maneras:
   a. Usar la opción Run que esta arriba en la barra de menús.
   b. Usar icono "run"  en barra de herramientas.

Nota: Se puede usar también la primera letra dentro del botón de comando o de ordenes (OK), para activar las instrucciones o acciones del botón, es decir se puede usar click en botón o solo pulsar la letra O, para que se ejecute este ultimo procedimiento, solo poner un símbolo & en el caption del botón antes de la letra O.

El programa ya en ejecución debe ser similar (pero mejor diseñado) al siguiente ejemplo:



Ya que se practicó, la mecánica de creación de programas, resolveremos el problema de interactividad con el usuario.

Es decir en estos casos es el usuario quien generalmente proporciona los datos para su procesamiento en el problema, el trabajo del ingeniero de software, es procesarlos y darles sentido, coherencia y sobre todo eficiencia.

Para esto se ocupan, componentes llamados de "entrada", que permitan al usuario cargar o proporcionar un dato en ellos, dichos componentes son:

13.- COMPONENTE Edit (Standard) 

Este componente es el más importante componente visual, su función principal es manejar, la gran mayoría de los procesos de entrada y salida (input/output) al programa.

Es necesario entender, que este componente permite capturar datos y también como en el caso del componente *Label* desplegar datos, textos, mensajes o resultados de operaciones de ser necesario, usando la propiedad *Text* del componente *Edit*.

Esta propiedad *Text*, así como la propiedad *Caption* en *Label*, permiten igualarse a muchos procesos básicos, es decir, es fácil igualar *Text* o *Caption* a un dato, una variable, otro *Text* o *Caption*, o una expresión algebraica normal, como en los siguientes ejemplos;

```
Edit2->Text = 5;

Label3->Caption = "PATO";

Edit4->Text = 3 * 6.2 ;
```

En principio su valor de default es la palabra *Edit1* y es en su propiedad *Text* donde se modifica, generalmente al principio de un programa se deja en blanco, y al ejecutarse el programa, el usuario lo llena con los datos solicitados o el programa lo llena con el resultado de las operaciones.

Cuando un usuario lo carga con un dato, el dato almacenado queda de tipo texto, no importa lo que haya escrito el usuario.

Para resolver el problema de usar datos numéricos dentro del *Text* de un componente Edit, en operaciones matemáticas, solo agregarle a la propiedad *Text*, las funciones .ToInt() o .ToDouble(), como se muestra en los siguientes ejemplos.

```
Edit3->Text= Edit2->Text.ToInt() * 5;

Edit5->Text= 3 * pow( Edit2->Text.ToDouble(), double (4) );
```

éste último es el problema de calcular $3x^4$

En particular, se deberá asociar los *Edit* con las variables, y solo existirán en principio cuatro tipos de *Edit*.

Formato Edit	Tipo De Dato	Ejemplo
Edit->Text	Alfanumérico	Nombre, Ciudad
Edit->Text[1]	Char	Grupo, Sexo, Letra
Edit->Text.ToInt()	Entera	Edad, Año
Edit->Text.ToDouble()	Real	Sueldo, PI

Notas:

- No usar el componente *Edit*, directamente como una variable cualquiera (con su método o función correspondiente) y no construir la operación o formula con los propios componente o cajas *Edit*.
- Como se observa este componente *Edit* permite capturar datos por parte del usuario del programa.
- Respetar la ortografía en el ejemplo, sobre todo en las instrucciones (las que terminan con ;)
- Recordar que escribir con mayúscula o minúscula se tratan como diferentes.

- Si son muchas operaciones en un problema, el procedimiento es similar, solo resolverlo por partes para evitar convertirlo en un renglón de código gigante.
- De un componente *Edit*, solo salen **int** o **double** nada más, si se necesita otra clase de dato numérico, entonces es tiempo de practicar el operador **cast**.

Conversión de datos o variables numéricas a Texto de un componente *Edit*.

Para resolver este problema, donde se tiene una variable numérica cargada, con un resultado o dato numérico y se pretende mandarla a un componente *Edit* o *Label* para su despliegue, se realiza de la siguiente manera:

- Usando la clase AnsiString, por principio de cuentas AnsiString puede recibir como argumentos o parámetros una variable de tipo entera o **double** y convertirla directamente a una **string**, que es el tipo de dato que espera el componente *Edit* o el componente *Label* para su despliegue, Ej.;

```
// Zona de declaración
        int alfa;
        double beta;
// Zona de captura
        alfa = Edit1->Text.ToInt();
        beta = Edit2->Text.ToDouble();
// Zona de operaciones
        alfa = alfa+3;
        beta = beta *5.1;
// Zona de resultados y conversión de números a string o Edit
        Label3->Caption = AnsiString(alfa);
        Edit4->Text = AnsiString(beta);
```

Como se observa se puede transferir variables numéricas tanto a *Label* como a *Edit* vía la clase AnsiString.

Las propiedades *Edit->Text* ó *Label->Caption* permiten que se igualen a una variable, un dato, o una expresión, es decir es valido;

```
int y = 20;
Edit1->Text= 50;
Edit2->Text = " gato ";
Edit3->Text= 30 * 80;
Edit4->Text= y - 3;
Edit5->Text= -5 * y + Edit3->Text.ToDouble();
Edit6->Text= 15 * Edit3->Text.ToInt() -
Edit8->Text.ToDouble();
```

Para el caso de resultados decimales, la salida incluye todo el conjunto de decimales asociados a un tipo **double** (muchos ceros), para resolver este problema se pueden usar directamente algunos de los métodos asociados a la clase AnsiString , por ejemplo uno de esos métodos es;

```
FormatFloat("string de formato", var double);
```

La string de formato contiene una serie de símbolos literales que se utilizan para darle el formato de salida deseado, esto símbolos o constantes literales son;

Const. Lit.	Significado
0(cero)	Rellena con 0(ceros) todas las posiciones indicadas, ejemplo "0000.00" para el numeró 23.4 la string de salida seria 0023.40
#	Constante muy usada para formato de valores numéricos, esta constante solo despliega el valor numérico normal.
. (punto)	Se utiliza para separar la parte entera de la parte decimal.
, (coma)	Se utiliza para separar el valor numérico en unidades de millar.
E+, E-, e+, e-	Se utilizan para convertir a notación científica exponencial el valor de salida.

Notas:

- El valor de salida es redondeado a tantos decimales como 0s, o #s, existan en la string de formato.
- Si la string de formato no incluye un punto decimal, entonces el resultado es redondeado al valor entero más cercano.

Ejemplo:

```
button1click(---){
// ZONA de declaración
        double alfa;
// captura y conversión
        alfa = Edit1->Text.ToDouble();
// operaciones
        alfa = alfa / 3.1416 ;
// conversión y despliegue
        Edit2->Text= FormatFloat("###,###.##", alfa);
}
```

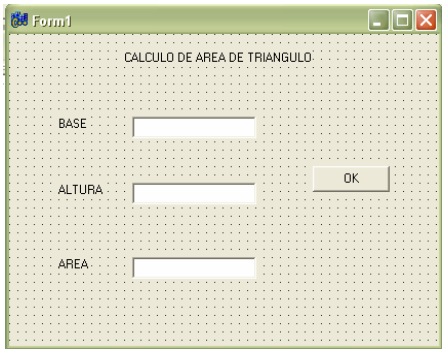
En resumen, este componente *Edit*, es el componente mas importante y elemental en todo problema que involucre el procesamiento de datos en ambientes visuales, se debe acostumbrar a considerarse como un elemento importante en cualquier problema visual, y acostumbrarse a procesarlo como si fuese una variable normal cualesquiera.

PROGRAMA EJEMPLO

Se construye y resuelve el segundo programa del modelo de solución del área del triángulo.

Para crear y diseñar el formulario y sus componentes se necesita, un formulario, cuatro *Label*, tres *Edit* y un *Button*, quedando así;

a) Pantalla de diseño



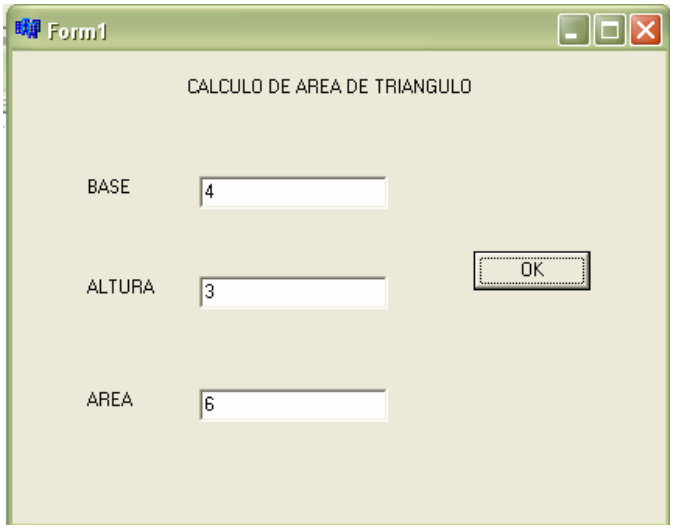
b) Programa

El código o programa principal se escribe dentro del evento *OnClick* del componente *Button*, quedando así;

```
void __fastcall TForm1::Button1Click(TObject *Sender)
        {
        // PROGRAMA QUE CALCULA AREA DE TRIANGULOS

        //ZONA DE DECLARACION
                float BASE, ALTURA, AREA;
        // ZONA DE CAPTURA
                BASE = Edit1->Text.ToDouble();
                ALTURA = Edit2->Text.ToDouble();
        //ZONA DE OPERACION
                AREA = BASE*ALTURA/2;
        //ZONA DE RESULTADOS
                Edit3->Text=FormatFloat("###.##",AREA);
        }
```

c) Pantalla de ejecución o de salida



Recordar que cuando se capturen los datos de base y altura, no dejar espacios en blanco antes del primer número o se darán problemas y errores de conversión a valores numéricos.

Para resolver mejor este problema de formatos mas adecuados para captura de datos, se usaran nuevos componentes que se analizan mas adelante.

PROBLEMAS SUGERIDOS

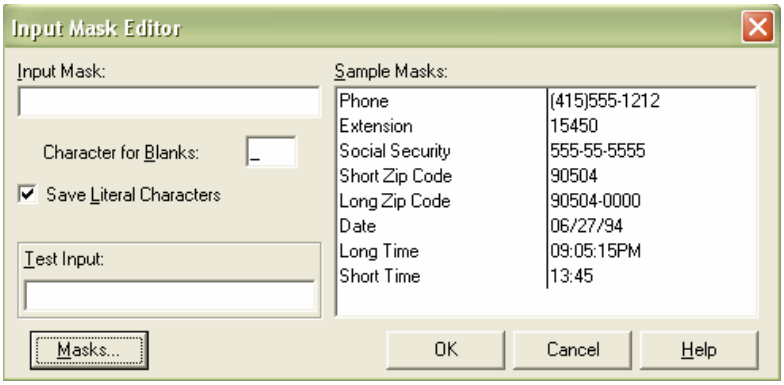
1. Convertir a programas todos los problemas vistos en el modelo de solución.

14.- COMPONENTE MaskEdit (Adicional)

Este componente es muy similar en su uso al componente *Edit*, excepto que proporciona una mascara especializada para el formato de datos, es decir se puede usar para que el usuario proporcione datos con formatos bien definidos, como son valores numéricos que incluyan puntos y comas por ejemplo 3,345.87, o que incluyan símbolos como el de $, o para el caso de fechas que lleven su propio separador como por ejemplo 02/28/97.

También se puede usar, para asegurarse que el dato proporcionado por el usuario, solo incluya números, o solo contenga letras, etc.

Para darle formato al dato que el usuario debe proporcionar solo hacer dobleclick a un lado de la propiedad *EditMask* en el Inspector de Objetos y esto nos da el siguiente mini editor de datos:



- Observar en la ventana derecha, algunos ejemplos de "mascaras de edición".
- En la ventanilla arriba a la izquierda es donde se colocan los caracteres especiales de edición (en el ejemplo se están usando, $, #, puntos y comas).

- En la ventanilla abajo a la izquierda es donde se pueden proporcionar algunos datos de prueba, para probar el formato diseñado.
- Recordar que este formato es para capturas, no para despliegues, puesto que para este caso (despliegue) se usa *FormatFloat()*.
- No olvide usar el botón OK, cuando se termine de construir la mascara de edición.
- Los principales caracteres especiales de edición son:

Caracter	**significado**
!	Caracteres opcionales se despliegan en blanco
>	Caracteres que siguen deben ser mayúsculas
<	Caracteres que siguen deben ser minúsculas
L (mayúscula)	Requiere una letra en esta posición
L (minúscula)	Permite una letra es esta posición pero no la requiere
A (mayúscula)	Requiere un alfanumérico en esta posición
a (minúscula)	Permite un alfanumérico pero no lo requiere
0	Requiere un numero en esta posición
9	Permite un número pero no lo requiere
#	Permite un número y también signos más y menos
: (dos puntos)	Separa horas:minutos:segundos
/	Separa meses/días/años
; (punto y coma)	Se utiliza para separar los tres campos o partes de una mascara
_	inserta espacios en blanco en el texto

Cualquier otro carácter que no aparezca en la tabla anterior, puede aparecer en una mascara, pero solo se tomara en cuenta como una literal cualesquiera, es decir son insertados automáticamente y el cursor los brinca.

El segundo campo o parte de una mascara es un caracter simple que indica qué caracter literal debe ser incluido como parte del texto del componente *MaskEdit*, por ejemplo (***000)_000-0000;0;*,***

Un 0 en el segundo campo indica que sólo deben capturarse los diez dígitos marcados con 0, en lugar de los 14 que tiene la mascara.

El tercer campo de la mascara, es el carácter que se quiera que aparezca en lugar de espacios en blancos.

Nota: para procesarlo usar solo *Text* no *Text.ToDouble()*

PROBLEMAS SUGERIDOS

1. Reeditar y corregir todos los problemas hechos y que contengan el componente en capturas numéricas.

15.- COMPONENTE ComboBox (Standard)

Existen muchas ocasiones en donde el usuario del programa tiene que proporcionar datos que provienen de un conjunto finito y muy pequeño de posibles respuestas, esto significa que cada vez que se ejecute el programa, el usuario estará proporcionando las mismas respuestas.

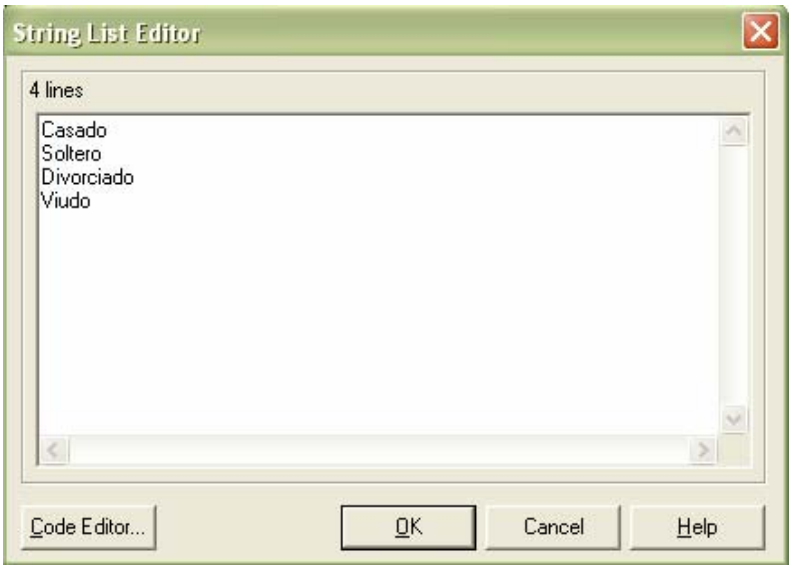
Ejemplo de esta clase de datos, son por ejemplos Estado Civil, las posibles respuestas solo son (Soltero, Casado, Divorciado, Viudo), otro ejemplo es Sexo (Masculino, Femenino), etc.

Para situaciones como esta, existen componentes que permiten programar por adelantado las posibles respuestas, y el usuario solo debe seleccionar la respuesta apropiada, en lugar de tener que escribirla.

Este componente *ComboBox* nos permite definir en primera instancia un conjunto de datos, valores ó respuestas asociados a una caja de edición cualesquiera, así ahora el usuario tendrá la oportunidad de seleccionar un dato del conjunto de datos o respuestas ya predefinido.

Este componente *ComboBox* tiene dos partes, una parte de encabezado, para poner el nombre del grupo de respuestas (por ejemplo Estado Civil, Sexo, etc.), que se carga usando la propiedad *Text* del componente.

La segunda parte es la lista de opciones o respuestas que se carga al tiempo de diseño de la ventana. En el momento de poner el componente *ComboBox1*, hacer dobleclick a un lado de la propiedad *Items* en el Inspector de Objetos y sale el siguiente editor de string:



Al momento de ejecución del programa, toda la lista de respuestas, estarán a la vista del usuario, para que este último la seleccione.

Recordar que el usuario al momento de ejecución del programa, sólo verá el encabezado, para seleccionar su respuesta deberá apretar la flechita que está a un lado del encabezado.

Para procesar este componente:

- Usar su propiedad *Text* de manera normal, es decir si la respuesta se coloca en string,
- Sólo usar *ComboBox1->Text*, o si la respuesta se quiere numérica, convertir Text aToInt() o ToDouble(), ej, *ComboBox1->Text.ToDouble()*.

PROBLEMAS SUGERIDOS

1. Reeditar los problemas ya resueltos, agregando este componente en los casos de capturas que lo pueden admitir (capturas donde ya se tienen las entradas seleccionadas o conocidas de antemano).

16.- COMPONENTES DE AGRUPAMIENTO

Como ya se empieza a notar en las aplicaciones construidas, la cantidad de datos e información empiezan a amontonarse en la ventana simple que se ha venido construyendo.

Para resolver este problema, se tienen dos métodos, el primero de ellos consiste de una serie de componentes que permiten agrupar datos o información (resultados) de una manera más lógica y estética.

El segundo método consiste de construir y trabajar con dos o más ventanas a la vez.

Se empieza por el primero método, es decir componentes de agrupamiento.

17.- COMPONENTE Panel (Standard)

Es el componente más sencillo y común de agrupamiento, se utiliza para poner un panel o un cuadro o marco dentro de una ventana.

El componente *Panel1* puede contener toda una serie lógica de otros componentes.

Solo se deberá recordar colocar primero todos los paneles en la forma y encima de ellos los componentes que contendrán.

Este componente también tiene una serie de propiedades que le dan una mejor presentación usando las propiedades *BevelInner*, *BevelOuter*, *BevelWidth*, y *BorderWidth*.

Es decir se puede dividir una sola ventana en varias partes, por ejemplo en un panel se ponen los componentes donde se capturan los datos de un problema junto con el botón de OK, y en otro panel se construye la salida, por ejemplo se crea un panel para capturar los datos de un empleado incluyendo sueldo diario y días trabajados y un segundo panel contiene su cheque semanal de pago (problema sugerido).

Para modificar programas construidos sin paneles, el procedimiento para agregarlos es:

1. Mover todos los componentes abajo en la ventana.
2. Colocar el panel en su posición.
3. Click del botón derecho en Componente a relocalizar.
4. edit, Cut
5. Click del botón derecho dentro del panel, donde se quiere el componente
6. edit, Paste

TAREAS PANEL

1. Reeditar e incluir este componente panel, en todos los programas impares hechos.

18.- COMPONENTE GroupBox (Standard) 

Este componente es otra forma standard de agrupamiento de componentes de programas en Windows, se usa para agrupar componentes relacionados dentro de una forma.

También se utiliza para separar áreas lógicas dentro de una ventana de Windows.

El texto que identifica el propósito general del grupo se escribe dentro de la propiedad *Caption* en el Inspector de Objetos, teniendo seleccionado este componente *GroupBox*.

Además de sus propiedades, métodos y eventos propios, como todos los componentes de este tipo, tiene o hereda las propiedades, métodos y eventos de todos los controles generales de tipo Windows.

Es muy similar al componente panel, excepto que incluye una pestaña que permite dejar mas claro, el propósito del grupo.

TAREAS GROUPBOX

1. Reeditar e incluir este componente groupbox para todos los problemas pares ya construidos

19.- VENTANAS EN C++Builder

El siguiente problema común, con el manejo de programas en *C++Builder*, es el de poder crear, controlar y administrar mas de dos formularios o ventanas a la vez.

Lo primero que hay que entender para poder resolver este problema es que en *C++Builder*, cada formulario o ventana tiene asociado ciertos recursos, además de los componentes que contiene, también una serie de recursos especiales, en general, el formulario, los recursos y los objetos hijos o componentes, se encuentran relacionados todos ellos, en un archivo especial, llamado "Unit1.cpp".

Es decir si se crea un segundo formulario o ventana, dicho *Form2*, junto con sus recursos, componentes, etc., se encontraría contenido en el archivo llamado "Unit2.cpp" y así sucesivamente.

Nota: Los componentes de este segundo formulario, también se simbolizarían y procesaran normalmente, es decir ellos también serán (*Edit1*, *Label5*, etc.).

Para crear un segundo formulario (*Form2*), Ir al menú *File->New->Form*.

El segundo formulario se construye normalmente, pero queda el problema de donde queda el botón de órdenes, la respuesta es, se pone en el primer formulario o ventana principal del programa.

El proceso en este botón es similar a todos los programas anteriores, es decir primero se capturan los datos (pantalla o ventana de captura), luego se resuelve las operaciones y luego traspasar los datos a los componentes del segundo formulario o ventana.

Para poder realizar este proceso, se debe usar una nueva sintaxis con todos los componentes usados, para que incluyan el formulario que los contiene, es decir se usa;

```
NombreFormulario->NombreComponente->NomPropiedad;
```

Ejemplos;

```
a) Form5->Edit3->Text = Form1->Edit2->Text;
// se esta pasando el texto de edit2 de primera
// ventana al texto de edit3 de la quinta ventana
```

```
b) int alfa = Form3->Edit4->Text.ToInt();
// se esta poniendo en una variable entera el contenido
// transformado a entero de la caja cuatro de la tercera
// ventana
```

Como se observa, procesar los elementos de dos ventanas, es sencillo pero además existen ciertas condiciones que deberán cuidarse para que estos procesos funcionen, estas condiciones son:

- Crear, armar y diseñar todas las ventanas primero, junto con sus componentes y funciones.
- Cualquier ventana que mencione o contenga una referencia dentro de su código a otra ventana, deberá incluir en su Unit respectiva, la unidad (Unit) del otro formulario o ventana.
- Para incluir la unidad (Unit) del otro formulario o ventana, solo tener seleccionada o al frente la ventana que llama, y usar la orden File Include Unit, que se encuentra arriba

en la barra de menú, junto con el Run, Compile, etc., al dar esta orden (File Include Unit) sale una lista con todas las unidades(Unit) que ya se diseñaron, seleccionar la apropiada y ya se incluirá automáticamente en el formulario o ventana actual.

- Si una ventana o formulario referencia dos o mas formularios diferentes, entonces usar la orden File, Include Unit, tantas veces como sea necesario.

Este procedimiento permite construir programas con dos o mas ventanas, pero el problema es que todas ellas estarán a la vista del usuario, para resolver este problema, el procedimiento mas sencillo es poner en *False* la propiedad *Visible* del formulario o ventana que se quiera tener oculta y poner cualquiera de las siguientes instrucciones en el código del programa para que aparezcan o desaparezcan a voluntad;

```
1. Form2->Visible = true;
2. Form2->Show(); // similar a la anterior(pero mas corta)
3. Form2->ShowModal(); /* no permite acceder la primera ventana, hasta que se
   cierra(X) la segunda ventana.*/
```

Close, en una ventana, cuando el usuario lo selecciona, cierra la ventana.

Programación Avanzada permite crear y destruir ventanas y componentes al tiempo de ejecución del programa, pero estos conocimientos, no forman parte del ámbito de este libro.

Si se tienen formularios de mas, o formularios que ya no se quieren usar o de plano mal construidas se pueden remover del proyecto usando el icono  (Remove File from Project) de la barra de herramientas, y de la ventanilla que aparece seleccionar la unidad que contiene la forma que se quiere eliminar.

TAREAS DE VENTANAS

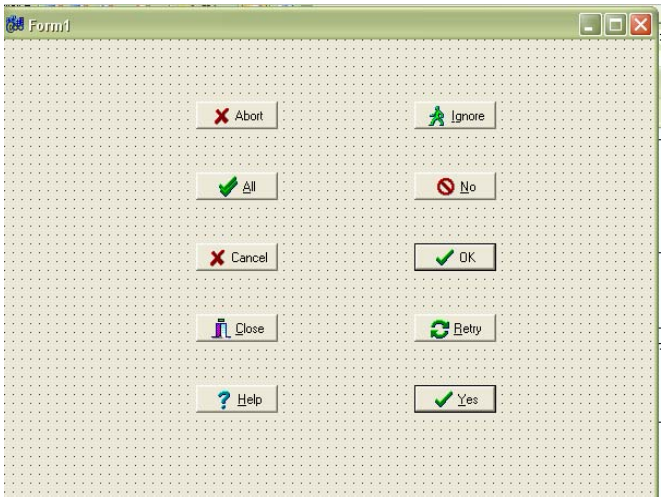
1. Reeditar tres cualesquiera de los problema ya resueltos para que incluyan cuando menos dos ventanas
2. Construir un programa donde la primera ventana capture los datos de un alumno incluyendo las calificaciones de 3 materias diferentes y una segunda ventana despliega un reporte de calificaciones del alumno incluyendo promedio final.

20.- COMPONENTE BitButton (ADITTIONAL) 

Este componente visual, permite realizar en forma fácil toda una serie de tareas comunes en Windows.

En principio es parecido al componente *Button*, pero en el ejemplo de arriba observar que incluye un gráfico o bitmap, que lo hace más agradable y visible al usuario.

Es en su propiedad *Kind*, en el inspector de objetos, donde se pueden definir cualquiera de sus diez opciones, como lo muestra la siguiente pantalla.



Todos son *BitButton*, nada más con *Kind* Seleccionado.

PROBLEMAS SUGERIDOS

1. Calcular el área de un triángulo mediante la formula $Area = \left(P\left(P-a\right)*\left(P-b\right)*\left(P-c\right)\right)^{1/2}$ donde P es el semiperímetro $P = \left(a+b+c\right)/2$, siendo a, b y c los tres lados del triángulo.

2. Construir un programa para obtener la hipotenusa y los ángulos agudos de un triángulo rectángulo a partir de las longitudes de los catetos.

3. La fuerza de atracción entre dos masas, m_1 y m_2 separadas por una distancia d, está dada por $F = \frac{G * m_1 * m_2}{d^2}$, donde G es la constante de gravitación universal G = $6.673*10^{-8}$, $cm^3/g.seg^2$.

4. Escriba un programa que lea la masa de dos cuerpos y la distancia entre ellos para obtener la fuerza gravitacional entre ellas. La salida debe estar en dinas (1dina = gr.cm/seg^2).

CAPITULO II: CONTROL DE PROGRAMA

1.- INTRODUCCION

Las instrucciones de control de programa permiten alterar la secuencia normal de ejecución de un programa.

Estas instrucciones se dividen en tres grandes categorías:

1. Instrucciones Condicionales que en C++Builder se implementan con las instrucciones:
   - if
   - switch
2. Instrucciones de repetición o Ciclos como:
   - for
   - while
   - do-while
3. Instrucción de salto incondicional
   - goto

Muchas de ellas con sus correspondientes componentes visuales.

2.- INSTRUCCIONES CONDICIONALES

Una de las más poderosas características de cualquier computador es la capacidad que tiene de tomar decisiones.

Es decir al comparar dos alternativas diferentes el computador puede tomar una decisión, basándose en la evaluación que hace de alguna condición.

Ejemplo de instrucciones condicionales:

a)

si sueldo > 30000000
desplegar "rico"
sino
desplegar "pobre"
fin-condicional

b)

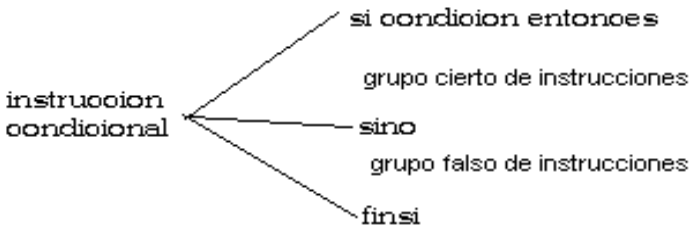
si sexo = 'F'
imprime mujer
sIno
imprime hombre
fin-condicional

De los ejemplos anteriores, observe que los caminos escogidos por el computador dependerán de la evaluación que este hace de la condición.

Para propósito de construcción visual de programas, este tipo de instrucciones condicionales se usaran en forma interna es decir son parte del código del programa que se empotra dentro de los eventos de componentes, no son formas o componentes en si.

Pero debe recordar que lenguajes visuales de igual forma tienen componentes que permiten del mismo modo al usuario tomar decisiones, incluso directamente en pantalla, es decir existen los llamados componentes de selección y decisión.

El formato general de una instrucción condicional es:



Como se observa son cuatro partes bien diferenciadas entre si;

- La propia instrucción condicional en sí
- La condición
- El grupo cierto de instrucciones

- El grupo falso de instrucciones

Cuando el computador evalúa una condición, el resultado de esa evaluación solamente tiene dos resultados posibles: o la condición es CIERTA o la condición es FALSA.

Esto dependerá del valor que tenga asignado o que se haya capturado para las variables que están en la condición, por ejemplo si se capturo 60000000 en sueldo en el ejemplo a), entonces el computador indicaría que la condición es CIERTA, pero en otro caso, si a la variable sueldo primero se le asigno un valor de 2500000 entonces el computador indicaría que la condición, es FALSA.

Ya dependiendo del resultado de la evaluación, el computador ejecuta las instrucciones contenidas en la parte CIERTA o en la parte FALSA de la condición.

3.- CONDICIONES SIMPLES

En general todas las condiciones simples se forman con:

Variables, operadores relacionales y constantes. Ejemplo

sexo = = 'm'

sueldo > 30000000

Una condición simple se define como el conjunto de variables y/o constantes unidas por los llamados operadores relacionales.

Los operadores relacionales que reconoce el lenguaje C++Builder son:

Operador	**Significado**
==	Igual que
>	Mayor que
<	Menor que
>=	Mayor o igual que.
<=	Menor o igual que.
!=	No es igual que, o es diferente que.

Observar y tener cuidado sobre todo con el operador de asignación (=), y el operador relacional de comparación por igualdad (= =), es decir;

sueldo = 500000 Se esta pidiendo cargar o asignar la variable sueldo con el valor 500000.

sueldo = = 500000 Se esta pidiendo que se compare el valor o dato que ya esta en la variable sueldo, contra el numero 500000.

Solo este último formato es valido dentro de una condición.

4.- INSTRUCCION IF

Es la instrucción condicional más usada en los diversos lenguajes de programación, su formato completo y de trabajo en C++Builder es:

asignar un valor a la variable de condición

```
if (condición)
{ grupo cierto de instrucciones;}
else
{ grupo falso de instrucciones; }
```

1. Observar donde van y donde no van los puntos y comas;
2. La condición va entre paréntesis ;
3. Si un *if* no necesita un grupo falso de instrucciones, entonces no se pone el *else*.

Ejemplos:

```
if (A = = "500000")
                { B = "Rico";}
else
                { B = "Pobre";}
```

PROBLEMAS SUGERIDOS

5. Capturar un número cualesquiera e informar si es o no es mayor de 100.

6. Capturar un numero entero cualesquiera e informar si es o no es múltiplo de 4 (recordar el operador mod(%), analizado en el tema de operadores aritméticos).
7. Capturar los cinco datos más importantes de un Empleado, incluyendo el sueldo diario y los días trabajados esto en un panel, desplegarle su cheque semanal en un segundo panel solo si ganó mas de 500.000 en la semana, en caso contrario desplegarle un bono de despensa semanal de 150.000 en un tercer panel.
8. Capturar los datos más importantes de un estudiante incluyendo tres calificaciones, todo esto en una ventana. Una segunda ventana que contiene una boleta de calificaciones es llamada si el estudiante es de la carrera de medicina, en caso contrario una tercera ventana despliega un oficio citando a los padres del estudiante a una charla amistosa con los Profesores de la escuela.
9. Capturar los datos más importantes de un producto cualquiera, incluyendo cantidad, precio, etc., desplegar una orden de compra, solo si el producto es de origen nacional, en caso contrario no hacer nada.
10. Se dice que un número es capicúa si al invertirlo queda el mismo número, por ejemplo: 12421. Realice un programa que obtenga un número entero y despliegue si es capicúa o no. Trabaje con números de hasta 5 dígitos.
11. La tarifa residencial de la C.A. La electricidad del Sur es la siguiente:

    Bs. 720 con derecho a 24 Kwh
    Bs. 81 por Kwh por los siguientes 126 Kwh
    Bs. 40 por Kwh por los siguientes 150 Kwh
    Bs. 30 por Kwh por los siguientes 300 Kwh
    Bs. 26 por Kwh por los siguientes 600 Kwh
    Bs. 21 por Kwh por los siguientes 800 Kwh
    Bs. 17 por Kwh por el resto del consumo

    Si se da el código del suscriptor, las fechas entre las cuales se mide el consumo y su consumo total el Kwh, determinar y desplegar el monto a pagar en bolívares y el costo promedio de cada día de servicio recibido.
12. Diseñar un programa que al introducir la fecha de nacimiento muestre el nombre del signo del zodíaco correspondiente

5.- CONDICIONES COMPUESTAS

En muchas ocasiones es necesario presentar más de una condición para su evaluación al computador.

Por ejemplo que el computador muestre la boleta de un alumno, si este estudia la carrera de ingeniería y su promedio de calificaciones es mayor de 12.

Una condición compuesta se define como dos o más condiciones simples unidas por los llamados operadores lógicos.

Los operadores lógicos que C++Builder reconoce son;

Operador	significado
&&	Y
||	Ó
!	NO

Para que el computador evalúe como CIERTA una condición compuesta que contiene el operador lógico "y", las dos condiciones simples deben ser ciertas.

Para que el computador evalúe como CIERTA una condición compuesta que contiene el operador lógico "o", basta con que una de las condiciones simples sea cierta.

La cantidad total de casos posibles cuando se unen dos o mas condiciones simples esta dada por la relación 2^n donde n = cantidad de condiciones, la primera mitad de ellos ciertos y la segunda mitad falsos.

Ejemplo, si formamos una condición compuesta con dos condiciones simples y el operador lógico "y", la cantidad total de casos posibles serian $2^2 = 4$, y se puede construir la siguiente tabla de verdad.

Tabla de verdad con "y"

1cs	2cs	Eval
C	C	C
C	F	F

1cs	2cs	Eval
F	C	F
F	F	F

La evaluación final, se obtiene usando la regla anteriormente descrita para una condición compuesta, que contiene el operador "y".

Esta tabla significa lo siguiente;

1. Cualquiera que sean la cantidad de datos procesados, siempre caerá en uno de estos cuatro posibles casos.

La tabla de verdad para una condición compuesta con "O" es la sig:

1cs	2cs	Eval
C	C	C
C	F	C
F	C	C
F	F	F

Como se observa, una condición compuesta con "O", es menos restrictiva, o el 75% de los casos terminarían ejecutando el grupo CIERTO de instrucciones de la instrucción condicional.

Construir una tabla de verdad para una condición compuesta de tres o mas condiciones simples, es también tarea sencilla, solo recordar que;

1. La cantidad posible de casos es $2^3 = 8$ casos posibles, la mitad empiezan con Cierto y la otra mitad empiezan con falso.
2. Para evaluar esta condición triple, primero se evalúan las dos primeras incluyendo su operador, bajo las reglas ya descritas y luego se evalúa, el resultado parcial contra la ultima condición, y ultimo operador, para obtener la evaluación final.

Ejemplo de una condición compuesta de tres condiciones simples, donde el primer operador lógico es el "y" y el segundo operador lógico es el "O", daría la siguiente tabla de verdad.

1cs	'Y' 2cs	Eva Parcial	'O' 3cs	Eva final
C	C	c	C	C
C	C	c	F	C
C	F	f	C	C
C	F	f	F	F
F	C	f	C	C
F	C	f	F	F
F	F	f	C	C
F	F	f	F	f

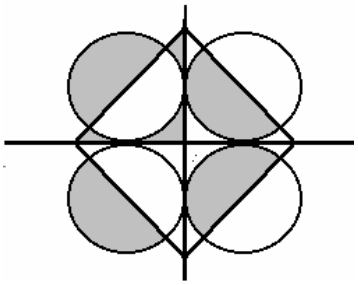
En la práctica, se recomienda que cada condición simple vaya encerrada en su propio paréntesis y las dos condiciones simples deben encerrarse entre paréntesis, como en el siguiente ejemplo;

```
if ((Sueldo > 500) && (Departamento = = "VENTAS") )
        { // aquí se construye una panel o ventana
        // por ejemplo que despliegue su cheque semanal }
else
        { // aquí se construye o despliega un panel o ventana
        // que despliegue por ejemplo un bono de despensa
        // o un oficio de motivación }
```

PROBLEMAS SUGERIDOS

1. Construir un programa que capture un número cualquiera e informe si es o no mayor de 50 y múltiplo de tres. (solo escribir el mensaje de respuesta de manera muy clara y esto resuelve el problema).
2. Construir un programa que indique si un número es un par positivo.

3. Capturar los datos de un producto incluyendo su cantidad en existencia, construir un panel que despliegue una orden de compra si la cantidad en existencia del producto es menor que el punto de reorden, o si el origen del producto es nacional.
4. Construir el programa del ejemplo del empleado, pero construirlo con tres ventanas, la del empleado, la del cheque y la del bono.
5. Determine, en una sola instrucción de decisión, si un punto de coordenadas (x,y) cae dentro del área sombreada.



6.- INSTRUCCION SWITCH

También existen ocasiones o programas donde se exige evaluar muchas condiciones a la vez, en estos casos, o se usan una condición compuesta muy grande, o se debe intentar convertir el problema a uno que se pueda resolver usando la instrucción switch();

La instrucción switch() es una instrucción de decisión múltiple, donde el compilador compara el valor contenido en una variable contra una lista de constantes int o char, cuando el computador encuentra el valor de igualdad entre variable y constante, entonces ejecuta el grupo de instrucciones asociados a dicha constante, si no encuentra el valor de igualdad entre variable y constante, entonces ejecuta un grupo de instrucciones asociados a un default, aunque este ultimo es opcional.

El formato de esta instrucción es el siguiente;

```
asignar valor a la variable de condición;
switch(variable de condición) {
        case const1: instrucción(es);
                break;
        case const2: instrucción(es);
                break;
```

```
        case const3: instrucción(es);
                break; .....
        default: instrucción(es);
}
```

Un ejemplo práctico;

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
        B = Edit1->Text;
        // modulo de switch
                switch( B ) {
                        case 'a':Animal = "aguila";
                                break;
                        case 'b': Animal = "Borrego"; break;
                        default: Animal = "No hay";
                }
}
```

Notas:

1. Solo se puede usar como variable de condición, una variable entera o variable char.
2. Las constantes que estamos buscando y comparando son de tipo char, por eso se deben encerrar entre apóstrofes (').
3. Si se quiere resolver el problema de mayúsculas o minúsculas en el teclado, observar que se usan dos case, pero con un solo break;
4. Recordar que switch() solo trabaja con constantes y variables de tipo char, int ó unsigned
5. En particular, instrucciones de tipo switch() se utilizan para construir programas de selección de RadioGroup´s donde al usuario se le plantean dos o tres problemas distintos y el propio usuario selecciona cual de ellos quiere ejecutarse tal como se explica a continuación.

7.- COMPONENTES VISUALES DE SELECCION Y DECISIÓN

Las instrucciones *if* y *switch*, nos permiten tomar decisiones o realizar selecciones dentro del código de un programa.

C++Builder proporciona una serie de componentes visuales que permiten al usuario, no al programador, tomar decisiones en pantalla, el programador solo se encarga de implantar código adecuado a la decisión tomada por el usuario.

COMPONENTE *CheckBox* (Standard)

El componente *CheckBox*, permite seleccionar una opción al usuario del programa o tomar una decisión, directamente en pantalla.

En la propiedad *Text* del componente es donde se escribe el sentido de la selección Ej.;

En los ejemplos, los componentes *CheckBox*, son las cajas donde el usuario toma una decisión (Ej. c) o realiza una selección (Ej. a,b)

Existen dos maneras de programar este componente:

a. Cuando el usuario selecciona un *CheckBox* la propiedad *Checked* refleja esta decisión quedando cargada con las constantes true o false, en estos casos solo validar con un if por cada *CheckBox* dentro de nuestro botón de órdenes, el estado de dicha propiedad.

ej.;

```
if ( CheckBox5->Checked == True){código};

if (CheckBox2->Checked = = False){código);
```

Para el ejemplo c) el botón de órdenes en el formulario o ventana respectiva, usando el método anterior, contendría 3 ifs, uno para construir boleta otro para construir citatorio y otro para construir un diploma.

b. El segundo método para programar el componente, involucra el evento *OnClick* de este componente *CheckBox*, este evento *OnClick* es activado automáticamente en cuanto el usuario realiza o marca o toma su selección, es claro que si no se programa este evento el usuario no observara ningún proceso, sino que tendrá que indicar que ya hizo su decisión, apretando el botón de OK.

   Pero si se programa el evento *OnClick* de este componente con el código adecuado, ni se tendrá que agregar un botón OK, ni se necesitará usar un if(Checked), porque el usuario ya indicó cual es su decisión o selección pero tiene la desventaja de que no se le da la oportunidad al usuario de cambiar su selección.

   Recordar que para programar este evento *OnClick*, solo hacer un dobleclick, dentro del componente.

TAREAS CheckBox

1. Evaluar la función $y = 3x^2 - 4x + 2$ para x = 2, -5, 8 (usar un *CheckBox* por cada valor de x, y programar cada evento *OnClick* de cada *CheckBox* con la operación correspondiente y el despliegue del resultado).

2. Modificar el ejemplo anterior pero colocando un botón y programar en el evento *OnClick* del botón.

3. Construir un panel con los datos de un automóvil, un segundo panel muestra un plan de financiamiento a dos años y un tercer panel muestra un plan de financiamiento a tres años. (son dos *CheckBox* en el primer panel y no hay botón de OK).

4. Construir una ventana que contenga los siguientes *CheckBox*

   i. conversión de pesos a dólares

   ii. conversión de libras a kilogramos

   iii. conversión de kilómetros a millas

Para resolver este programa, primero diseñar los cuatro formularios o ventanas que se necesitan, y en el primer formulario que contiene el *CheckBox* para el usuario, programar el evento *OnClick* del *Button* de ordenes con la instrucción switch(), los case solo contienen código para llamar o poner a la vista del usuario la ventana o formulario respectivo.

Además, poner en cada ventana de solución de un problema un botón de orden, con código de regreso a la ventana de menú, solo escribir en el *Caption* de este botón la palabra [menú] y en su evento *OnClick* ocultar la ventana de problema y poner visible la ventana o formulario del menú.

5. Construir un programa que capture un tipo de deporte y despliegue dos implementos deportivos apropiados.

COMPONENTE *RadioButton* (Standard) 

Se utilizan para presentar al usuario un conjunto de opciones mutuamente excluyentes entre si, es decir si el usuario selecciona un componente *RadioButton* todos los demás componentes *RadioButton* en el formulario, se desmarcan solos, o se deseleccionan solos, como mejor se entienda.

Su propiedad *Caption* es donde se coloca el texto que identifica el propósito del botón, en su propiedad *Checked* se refleja el cambio (True, False) además, su evento *OnClick* es activado automáticamente cada vez que es seleccionado el *RadioButton* por el usuario.

Cuando el usuario selecciona un *RadioButton*, todos los demás *RadioButton* en el objeto (formulario o ventana) son deseleccionados automáticamente, esto es por que dos *RadioButton* son mutuamente excluyentes entre si.

Esta última situación deberá resolverse por parte del programador, es decir se supone un programa donde el usuario debe seleccionar uno de entre dos sexos y uno de entre cuatro estados civiles, en este caso se necesitan seis *RadioButton*, pero como todos son mutuamente excluyentes entre sí, cuando el usuario seleccione uno de ellos, todos los demás se desmarcaran automáticamente.

Para resolver este problema se deberá usar los ya ampliamente conocidos y practicados componentes de agrupamiento, como son el componente *Panel* y el componente *GroupBox*.

Es decir se deberá encerrar en su propio *Panel* o *GroupBox* todos los *RadioButton* lógicos, es decir en un *Panel* los de sexo, en otro *Panel* los de estado civil, etc.

De esta manera C++Builder los evalúa por separado y se puede tener seleccionado un *RadioButton* en cada Panel.

TAREAS RadioButton

1. Diseñar y construir tres problemas similares a los resueltos con el componente *CheckBox*.

COMPONENTE *RadioGroup* (Standard)

Aunque es común agrupar un conjunto de *RadioButton* dentro de componentes *Panel* y *RadioGroup*, C++Builder proporciona este componente *RadioGroup* que está especializado en la agrupación de *RadioButton*.

Un componente u objeto *RadioGroup* es una caja especial que solo contiene componentes *RadioButton*, también cuando el usuario marca o selecciona uno de ellos, todos los demás se desmarcan o deseleccionan.

Para añadir los *RadioButton* al componente *RadioGroup*, solo editar la propiedad *Items* en el Inspector de Objetos, la cual nos muestra el mini editor de string ya visto y practicado en el tema de *ListBox*, solo recordar que cada renglón en el editor corresponderá a un *RadioButton* dentro del *RadioGroup*.

Para procesar o programar el *RadioButton* seleccionado por el usuario, solo usar la propiedad *ItemIndex* que queda cargada con el número de *RadioButton* seleccionado por el usuario.

Este código deberá usarse dentro del evento *OnClick* de un componente *Button* (OK).

Ejemplo:

Un ejemplo práctico;

```
void __fastcall Tform5::Button1Click(TObject *Sender)
{
        // modulo de switch
                switch (Form5->RadioGroup3->ItemIndex ) {
                        case 0: Código asociado al primer RadioButton;
```

```
                    break;
                case 1: Código asociado al segundo RadioButton; break;
                case 2: Código asociado al tercer RadioButton
        } }
```

También se pueden desplegar los botones en una o más columnas, usando la propiedad *Columns* en el Inspector de Objetos, para indicarle cuantas columnas de *RadioButton* se quieren manejar.

PROBLEMAS SUGERIDOS

1. Capturar los cinco datos mas importantes de un cliente, usando *RadioButton* para las respuestas, un segundo panel abajo en el formulario despliega en componentes Label las selecciones del usuario.
2. Construir un cuestionario de 6 preguntas sobre los hábitos de estudio de un estudiante.
3. Construir un cuestionario de 5 preguntas con las preferencias políticas de una persona, un panel abajo despliega un concentrado con totales y porcentajes acumulados por cada pregunta.

   Es decir un usuario responde el cuestionario aprieta el botón de OK, el panel de abajo se actualiza para mostrar totales y porcentajes de cada pregunta, y se limpian las respuestas, un segundo usuario responde el cuestionario.... Y así sucesivamente.

COMPONENTE *MainMenu*(Standard)

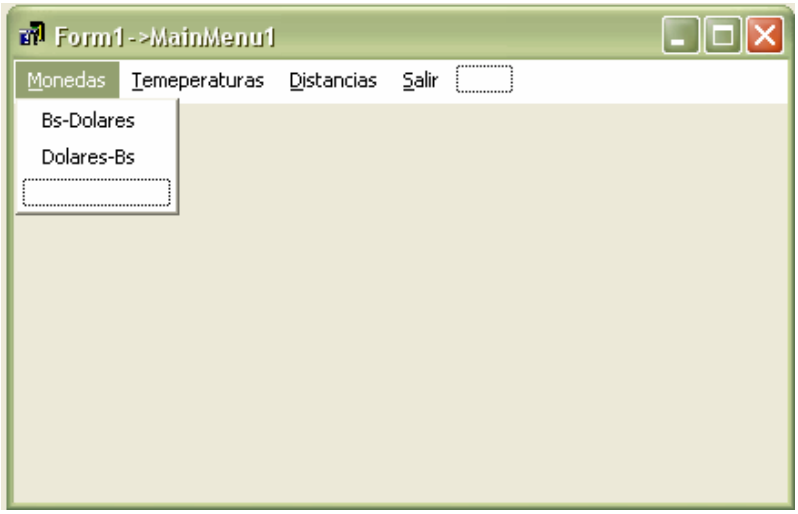
Con este componente *MainMenu* se forman las barras de menú en cualquier programa de Windows (la barra que contiene File, Edit, etc.) junto con sus opciones correspondientes.

Cuando se coloca este componente en un formulario, se deberá entender que *Form1* ahora mostrará o desplegará una barra de menú, tal como si fuese una ventana de Windows.

Para construir un programa de selección de menú;

1. Poner un componente *MainMenu* en una parte de la forma donde no estorbe (esto es porque este componente queda flotando dentro de *Form1* y solo activa o se convierte en barra de menú al momento de ejecución del programa).

2. Para cargarle las opciones, no del código que ejecutara cada opción, haga dobleclick dentro de este componente para que aparezca el siguiente diseñador de menús.



3. No es *Form1*, es un formulario especial que se parece mucho, pero solo se activa cuando se hace un dobleclick en un componente *MainMenu*.
4. La barra superior es el menú, cada opción del menú (Moneda, Temperatura, Distancias, etc.) tienen sus propias subopciones o submenús.
5. Para escribir las opciones (Moneda, Temperatura, etc.) primero hacer un click en la parte correspondiente (en el lugar que ocupan o donde se desplegaran) y escribir en la propiedad *Caption* del Inspector de Objetos la palabra correspondiente.
6. Para escribir las subopciones, primero hacer click en la opción correspondiente (por ejemplo en la parte donde está la palabra Distancias) y aparece debajo de ella la primera subopción marcada y seleccionada (es un cuadrito que se pone debajo de Distancias) y escribir en el *Caption* del Inspector de Objetos la palabra adecuada a la subopción correspondiente (ejemplo Millas-Kms), al escribir y usar tecla <ENTER> en el caption ya aparece el cuadrito de la segunda subopción.
7. Para marcar una letra de una opción o subopción como HotKey (Tecla de acceso rápido), solo debe poner el signo **&** antes de la letra correspondiente.
8. Con los pasos anteriores ya queda construido el menú, pero ahora falta cargarles el código o programa a ejecutar por cada opción o subopción del menú, recordar que

estos eventos también se activan al hacer un click en cada opción o al seleccionar la primera letra (si se uso el &).

9. Para cargar el código haga dobleclick en la opción o subopción correspondiente y aparece el mini editor de programas, con el evento OnClick de la opción o subopción correspondiente.
10. El código a escribir en la mayoría de los casos, consiste solamente en ocultar la ventana principal y llamar la ventana que contendrá el programa correspondiente a la opción deseada.
11. Para terminar y regresar al formulario principal haga click en la "X" del diseñador de menús.

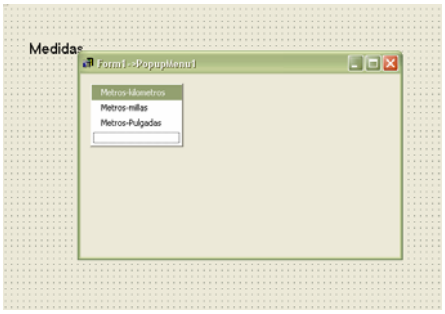
PROBLEMAS SUGERIDOS

1. Realizar el ejemplo de dos conversiones monetarias, dos opciones de temperaturas y dos de distancias.
   - Primero diseñar y construir todos los formularios que va a necesitar
   - No olvide poner algunos *Label*'s de encabezado en la ventana principal
2. No olvidar agregar código para ocultar la ventana de trabajo y regresar a la ventana con el menú principal en el botón ok o en otro botón similar de dicha ventana.

COMPONENTE *PopupMenu* (Standard)

Este componente encapsula propiedades, métodos y eventos de un menú popup, este mini menú se activa cuando el usuario hace un click derecho en muchos componentes que lo pueden contener.

Un ejemplo;



Este componente *PopupMenu* se tiene que “pegar” a otro componente, por ejemplo un *Label*, un *Edit*, etc.

Para crear un *PopupMenu* debe:

3. Seleccionar el componente *PopupMenu* en la barra de componentes y ponerlo en una parte del formulario donde no estorbe (también queda flotando).
4. Para cargarlo de opciones haga dobleclick en dicho componente e igual que en componente *MainMenu*, sale el mini editor, haga click en la parte donde va la opción escrita y escribir la palabra en la propiedad *Caption* del Inspector de Objetos.
5. Para cargarle el código a cada opción, haga dobleclick en la opción correspondiente.
6. Para salirse del mini editor, haga click en la [X] de arriba.
7. Por ultimo y muy importante, recuerde que se tiene que pegar a un componente cualquiera, para esto seleccione el componente y haga click en la propiedad popup del componente, sale la lista con todos los componentes *PopupMenu* que ya se haya construidos.

PROBLEMAS SUGERIDOS:

1. Construir un menú con las siguientes opciones:
    - [autos] $\Rightarrow$ financiamiento a 2,3,4 años
    - [lavadoras] $\Rightarrow$ financiamiento a 2,3 años

8.- Ciclo FOR

Estos ciclos, resuelven el problema de repetir todo el programa, o cierta parte del programa más de una vez.

Este ciclo es uno de los más usados para repetir una secuencia de instrucciones, sobre todo cuando se conoce la cantidad exacta de veces que se quiere que se ejecute una instrucción simple o compuesta.

Su formato general es:

```
for (inicialización; condición; incremento)
{ instrucción(es); }
```

Ejemplo:

```
For (x = 1; x <= 10; x = x + 1)
{Label4->Text = "pato";}
```

En su forma simple la inicialización es una instrucción de asignación que carga la variable de control de ciclo con un valor inicial.

La condición es una expresión relacional que evalúa la variable de control de ciclo contra un valor final o de parada que determina cuándo debe acabar el ciclo.

El incremento define la manera en que la variable de control de ciclo debe cambiar cada vez que el computador repite un ciclo.

Se deben separar esos 3 argumentos con punto y coma (;)

Casos Particulares;

1. El ciclo comienza en uno y se incrementa de uno en uno, este es el caso más general.
2. Pero el valor inicial puede se diferente de uno, ejemplo;

   ```
   for (x = 5; x <= 15; x = x + 1){ Instrucciones;}
   ```

3. Incluso el valor inicial puede ser negativo, ejemplo;

   ```
   for (x = -3 ;x <= 8; x = x + 1) { Instrucciones;}
   ```

4. Los incrementos también pueden ser diferentes al de uno en uno, Ej.;

   ```
   for (x=1; x <= 20; x = x + 3) { Instrucciones;}
   ```

5. Incluso pueden ser decrementos, solo que en este caso, recordar;
   a. el valor inicial de la variable debe ser mayor que el valor final.
   b. cambiar el sentido de la condición.

   Ejemplo:

   ```
   for (x = 50 ; x >= 10; x = x - 4 ) { Instrucciones;}
   ```

6. Solo para los casos de incrementos y decrementos de una en una unidad, se puede sustituir en el for
   a. el x = x + 1 por x++

b. el x = x - 1 por x- -

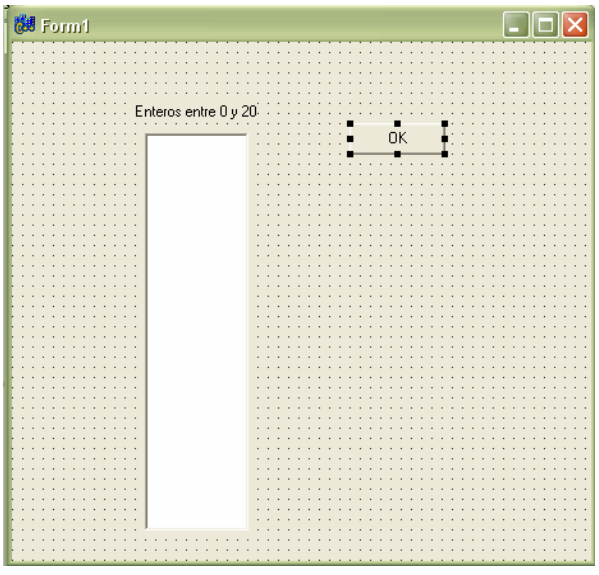
Ejemplo: desplegar los números enteros, comprendidos entre el 10 y el 20.

Necesitamos ahora un componente que pueda almacenar y desplegar un conjunto de los 10 resultados, el único componente visto hasta ahora con esta capacidad es el componente *ComboBox*, sin embargo existe otro componente llamado *ListBox* (Standard), muy similar a *ComboBox*, excepto que no tiene encabezado y todos sus elementos los mantiene a la vista del usuario, no ocultos como el *ComboBox*, dicho componente *ListBox* se analiza a fondo en el siguiente capítulo, pero es el que de momento permite resolver el problema del for (desplegar un conjunto de resultados a la vez).

Tanto *ComboBox* como *ListBox* permiten cargar todos sus elementos o valores, dentro de un programa, usando un método llamado *Add* (valor) en su propiedad *Items*, como se ve en el siguiente programa ejemplo.

Para este problema se necesita poner en *Form1*, un componente *Button* OK que en su evento *OnClick* contiene el for y la carga del componente *ListBox*, como sigue:

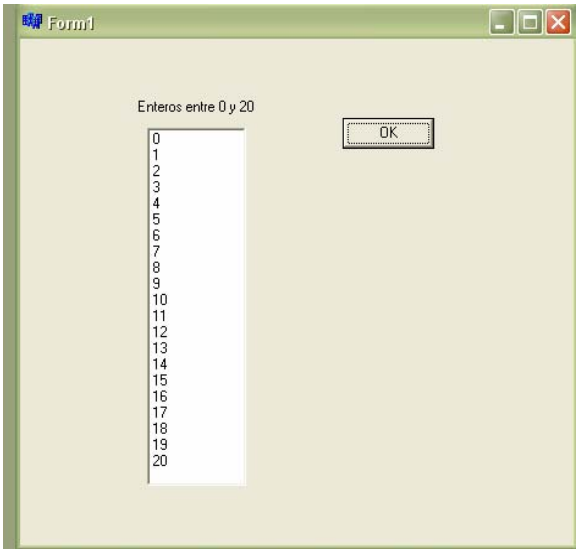
a) Pantalla de diseño



b) Programa

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
    //declaración
    int x;
    // ciclo for
    for (x = 10; x <= 20; x++)
            ListBox1->Items->Add(x);
}
```

d) La pantalla de salida debe ser igual o parecida a la siguiente;



Practicar hasta conseguir tener esta pantalla de salida o de ejecución, cuando se consiga entonces ya está listo para lo siguiente.

TAREAS for

1. Construir un programa que despliegue los números del 20 al 30.
2. Desplegar los enteros entre 50 y 30 acompañados de su potencia cuadrada y raíz cúbica respectiva (necesita tres *ListBox*).
3. Desplegar la tabla de multiplicar que el usuario indique.

9.- CICLO WHILE

En este ciclo el cuerpo de instrucciones se ejecuta mientras una condición permanezca como verdadera, en el momento en que la condición se convierte en falsa el ciclo termina.

Su formato general es:

```
While (condición)    // cargar o inicializar variable de condición
{
        grupo cierto de instrucciones;
        instrucción(es) para salir del ciclo;
}
```

Ejemplo #1 :

```
        x=1;
        while (x <= 10)
        {
                ListBox1->Items->Add("pato");
                x++;
}
```

1. While puede llevar dos condiciones, en este caso inicializar 2 variables de condición y cuidar que existan 2 de rompimiento de ciclo.
2. El grupo cierto de instrucciones puede ser una sola instrucción o todo un grupo de instrucciones.
3. La condición puede ser simple o compuesta.
4. A este ciclo también se le conoce como ciclo de condición de entrada prueba por arriba, porque este ciclo evalúa primero la condición y posteriormente ejecuta las instrucciones.

PROBLEMAS SUGERIDOS:

1. Desplegar enteros entre 50 y 80.
2. Desplegar múltiplos de 4 entre 60 y 20 acompañados de su logaritmos de base 10 y base *e* respectivos (revisar la ayuda y buscar las funciones matemáticas que están la librería <math.h>

3. Construir la tabla de dividir que el usuario indique.
4. Realizar un programa que calcule el factorial del número que el usuario indique.
5. Un número perfecto es aquel que es igual a la suma de todos sus divisores excepto él mismo. El primer número perfecto es seis (6) ya que 1+2+3=6. Escriba un programa que muestre todos los números perfectos menores que mil.
6. Un número palindrómico es aquel que también es primo cuando se invierten sus dígitos. Así tenemos que 17, 31, 37 y 113 son ejemplos de primos palindrómicos pues 71, 13, 73, 311 son también primos. Diseñe un programa para generar y desplegar los primeros 100 primos palindrómicos.
7. Calcule y muestre el número de términos necesarios para que el valor de la siguiente sumatoria se aproxime lo más cercanamente a 610000 sin que lo exceda.

$$\sum_{k=1}^{k=?} \frac{k^2+1}{k}$$

10.- CICLO DO WHILE

Su diferencia básica con el ciclo while es que la prueba de condición es hecha al finalizar el ciclo, es decir las instrucciones se ejecutan cuando menos una vez, porque primero ejecuta las instrucciones y al final evalúa la condición. Se le conoce por esta razón, como ciclo de condición de salida.

Su formato general es:

```
//cargar o inicializar la variable de condición;
    do {
        grupo cierto de instrucción(es);
        instrucción(es) de rompimiento de ciclo;
    } while (condición);
```

ejemplo:

```
x=1;
do {
        ListBox1->Items-Add("pato");
        x++;
}while (x <= 10);
```

Otra diferencia básica con el ciclo while es que, aunque la condición sea falsa desde un principio, el cuerpo de instrucciones se ejecutara por lo menos una vez.

PROBLEMAS SUGERIDOS:

1. Todos los de for
2. Todos los de while

11.- CONCLUSIONES ACERCA DE CICLOS

El problema de decidir cuál ciclo se debe usar, dado un problema cualquiera, se resuelve con:

- Si se conoce la cantidad exacta de veces que se quiere que se ejecute el ciclo o si el programa de alguna manera puede calcularla usar for.
- Si se desconoce la cantidad de veces a repetir el ciclo o se quiere mayor control sobre la salida o terminación del mismo entonces usar while.
- Si se quiere que al menos una vez se ejecute el ciclo, por ejemplo, el valor de la variable condición se conoce, sólo, dentro del ciclo, entonces usar do-while.

CAPITULO III: FUNCIONES

1.- INTRODUCCION

En secciones anteriores se mencionó la conveniencia de, para programas muy largos, dividir un programa en partes ó módulos de solución más sencilla para resolver luego cada uno de estos y así al terminar todos los módulos, queda casi listo el programa. Estos módulos se denominan más comúnmente como subprogramas o subrutinas, los cuales dividen el programa en trozos de forma tal que cada subprograma pueda ser utilizado por este y otros programas tantas veces como se necesite, evitando así tener que reescribir los mismos. Al separar un programa en módulos se tiene la ventaja que se pueden probar los módulos de manera independiente, antes de incluirlos en el programa y así hacer más sencilla la tarea de depuración de errores, además clarifican la lectura y análisis de un programa y por ende facilitan el mantenimiento y actualización de los programas.

Las funciones son subprogramas con secciones de código separadas del programa principal que se ejecutan cuando se necesitan realizar acciones específicas en el programa.

En conjunto pueden denominarse subrutinas, siendo posible llamarlas (usarlas) varias veces y en cualquier momento de la ejecución del programa e incluso desde otras subrutinas.

Se pueden clasificar en:

- ✓ Funciones invocadas por el usuario (Eventos): Se les debe colocar la identificación ***_fastcall*** en el encabezado de la función. Se caracteriza porque se ejecuta cada vez que ocurre el evento a cual ella controla.
- ✓ Funciones invocadas por código. Son funciones a las cuales sólo las puede invocar el programador mediante un llamado a través del código del programa. Este tipo de funciones las detallaremos en adelante.

Asociado a las funciones se tiene el término parámetro: que es un valor que se pasa a una función y se usa para alterar su operación o indicar el grado de alcance de su operación.

Una función posee el siguiente esquema:

```
Void(u otro tipo)[_fastcall] Identificador (Lista de Tipo y Parámetros)
{       Zona de Declaraciones Locales


Zona de Instrucciones Asociadas a la función
}
```

2.- FUNCIONES QUE DEVUELVEN VACIO

Una Función que Devuelve Vacío se utiliza para identificar un subprograma dentro de un programa. El identificador de la función define un conjunto de instrucciones que van a ser ejecutadas como un programa cada vez que se invoca el nombre de la función.

Cada función posee una cabecera seguida por un bloque similar al de un programa principal.

Una función que devuelve vacío es activada por una simple instrucción de ejecución de función desde el programa principal o desde otra función. Ésta puede requerir información de entrada y también puede producir información de salida. El intercambio de información con una función que devuelve vacío lo hace el programa principal o la función desde donde se le está invocando. Este intercambio se hace a través de parámetros los cuales tienen tipo definido y están representados por constantes, variables o expresiones seguidos por coma y colocados entre paréntesis a continuación del nombre y tipo de la función. La función de biblioteca **randomize ( )** es un ejemplo de función que devuelve vacío.

Una función que devuelve vacío, posee el siguiente esquema:

```
void Identificador (Lista de Parámetros)
{
Zona de Declaraciones Locales;
Zona de Instrucciones Asociadas a la Función;
}
```

3.- ALCANCE DE LOS OBJETOS

- **Locales**

  Son aquellos que sólo tienen vigencia en una parte del programa, así que las modificaciones que sufra el objeto durante la función, sólo se reflejarán en esa función, fuera del ámbito de la función, son inexistentes.

- **Globales**

  Son aquellos que tienen vigencia en cualquier parte; así que las modificaciones que sufra el objeto durante la función, se reflejarán en el resto del programa.

4.- FUNCIONES QUE DEVUELVEN VALORES

En programación, una función que devuelve valor trabaja igual que en matemáticas: un nombre de función con una lista opcional de parámetros y que representa un valor dependiente tanto de los parámetros como de la naturaleza de la función. Las funciones que devuelven valor sólo pueden ser invocadas desde expresiones y al igual que las

variables, llevan asociado un tipo de dato en el valor que devuelven. Las funciones de biblioteca como cos(x), sqrt(x) son ejemplos de funciones que devuelven valores.

Tanto las funciones que devuelven valores como las que devuelven vacío poseen una estructura similar ya que constan de un encabezado, una zona declaraciones y una zona de instrucciones.

Ambas, usualmente, comunican información con el programa principal o con la función desde donde se hace el llamado a ejecución, mediante los parámetros.

La diferencia radica en que la función que devuelve valores al ser invocada devuelve siempre un único resultado, mientras que las que devuelven vacío no necesariamente da un valor como resultado, pudiendo dar varios o simplemente generar una acción.

Una función que devuelve valores, posee el siguiente esquema:

```
Tipo de Resultado Identificador (Lista de Parámetros)
{
Zona de Declaraciones Locales;
Zona de Instrucciones Asociadas a la Función;
return valor a devolver;
}
```

Todo lo dicho para las funciones que devuelven vacío es válido para las funciones que devuelven valor, pero además:

- Las Funciones que devuelven valor tienen un Tipo de resultado.
- Las Funciones que devuelven valor especifican siempre un valor.
- Dentro del cuerpo de la Función que devuelve valor debe existir la instrucción **return** seguida del valor que se le quiere dar a la función.
- Las Funciones que devuelven valor deben ser invocadas siempre desde una expresión de su mismo tipo.

Ejemplo:

```
float Eleva (float X , float Y)
 {
        float Potencia;

        Potencia = Exp (Y*Log (X)) ;

        return Potencia;
 }
```

Las Funciones creadas por el usuario pueden ser declaradas en la zona de declaraciones de las funciones asociadas a los eventos de un objeto (luego de la sub-sección de declaración de variables). Estas rutinas así declaradas son de alcance local y sólo pueden ser empleadas en el evento que las contiene.

Pero si la función se declara en la sección previa a la forma permite que la misma sea invocada desde cualquier sitio en la unidad o desde cualquier programa que declare en uso esta unidad.

Nota: Toda función que devuelve valores de *Borland C* asigna el resultado al llamado de la función si se realiza la instrucción *return* con la variable que debe contener el resultado de la misma; en cualquier caso se debe colocar, al menos una vez, esta variable a la izquierda del signo de asignación en la ejecución de la función y debe ser declarada del mismo tipo de la función.

5.- PARAMETROS

Son aquellas variables que tienen vigencia solamente dentro de la función a quien pertenecen, pero son los encargados de transmitir y recibir valores hacia y desde el programa principal u otras funciones.

Parámetros Formales son aquellos que aparecen en el encabezado de la función y se utilizan cuando se quiere que la función se realice sobre objetos diferentes.

Parámetros Actuales son los valores y/ó variables que se colocan en el llamado a la función para resolver el problema específico. A cada parámetro formal debe corresponder uno actual del mismo tipo y en la misma posición.

Como analogía puede tratar de visualizar la función como un cuarto de calculo especializado para una oficina donde los datos de entrada y los resultados se pasan a través de una taquilla, en esta taquilla se colocan recipientes con los datos entrantes y en otros los salientes; los parámetros vienen a ser estos recipientes y pueden ser de varios tipos (que no tiene nada que ver con el tipo de dato contenido), entre ellos se tienen:

- Por Valor
- Por Referencia ó Dirección

6.- PARAMETROS POR VALOR

Reciben el valor que es enviado a la función desde donde se hace el llamado y dentro del cuerpo de la función actúan como una variable local. Lo anterior implica que esta variable puede ser modificada dentro de la función y la variable original permanecerá sin cambios.

De hecho todo parámetro es una variable completamente distinta a la original en el llamado, solo que su ámbito de acción se ciñe estrictamente a la función o lo que se llama es de *alcance local*.

Ejemplo de su declaración:

```
int Nombre (int Número, int  A, int Z)
```

Ejemplo de Parámetros por Valor:

```
int Nombre(int Numero, int  A, int Z)
{       float P;
        A = A+(Z++) ;
        P = Numero*A/Z ;
        return P;
}
```

7.- PARAMETROS POR REFERENCIA O DIRECCION

Son parámetros que pueden servir como portadores o transmisores de resultado o como portadores de valor inicial y valor final, ya que cualquier modificación será tomada en cuenta en todo el evento o programa desde donde es llamada la función.

En este caso la variable original del parámetro actual en la llamada, sufrirá toda modificación que se realice sobre su parámetro formal dentro de la función. Se denomina por Dirección porque el parámetro formal indicará la dirección de memoria donde se encuentra la original.

Ejemplo de su declaración:

```
int Nombre( int * Numero, int * A, int * Z)
```

Ejemplo de Parámetros por Referencia ó Dirección:

```
int Nombre( int * Numero, int * A, int * Z)
 {      float P;
        *A = *A+(*Z++) ;
        P =*Numero*(*A)/*Z ;
        return P;
 }
```

8.- PARÁMETROS POR SU USO

Los podemos clasificar en tres tipos:

- **Entrada**: Son portadores de información o datos a la función. A partir de ellos la función aplica el procedimiento para obtener el resultado. Se colocan por Valor.
- **Entrada/Salida**: Son portadores de información o datos a la función. A partir de ellos la función aplica el procedimiento para obtener la solución y ésta resulta en la modificación de este parámetro. Se colocan por Referencia o Dirección.

- **Salida**: Su valor inicial no tiene la menor la importancia pero es portador del resultado del problema que resuelve la función. Se colocan por Referencia o Dirección.

9.- CAJAS DE DIALOGOS

BUILDER provee la posibilidad de comunicarse con el usuario mediante tres funciones que despliegan cajas de diálogo de una línea de texto, que son de uso muy común en las aplicaciones en Windows y que se emplean en particular para atraer la atención del usuario en un momento determinado.

Estas cajas sirven para dar un simple aviso como para solicitar confirmación o no de una acción.

Los tipos de cajas de diálogos la tenemos a continuación:

ShowMessage

Se invoca empleando una simple instrucción de llamado de función que devuelve vacío. Produce una caja con un mensaje y un único botón de aceptación. Su sintaxis es:

ShowMessage("Mensaje que aparece");

MessageDlgPos

En aspecto es similar al anterior, sólo que uno puede indicar la posición de aparición de la caja de diálogo al indicar las coordenadas de la esquina superior izquierda y personalizar la caja.

Respuesta = MessageDlgPos("Mensaje que aparece", Tipo de Caja de Diálogo, [lista de botones], Ayuda, X, Y, botón por defecto, imagen);

Ayuda indica si este elemento posee Ayuda (Help) o no, X y Y son la posición en aparecerá la caja de diálogo, botón por defecto indica cual de la lista de botones aparecerá marcado de antemano e imagen indica cual imagen en mapa de bits aparecerá en la caja de dialogo. En la variable Respuesta, de tipo entera, la función *MessageDlgPos* devuelve el valor del botón que el usuario seleccionó.

Lista de Botones

Existen en BUILDER unas constantes predefinidas para indicar el tipo de botón que aparecerá en la caja de diálogo, estos son:

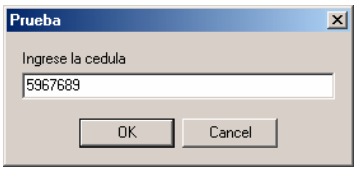
Constante	Tipo de Botón
mbYes	Un botón con 'Yes'.
mbNo	Un botón con 'No'.
mbOK	Un botón con 'OK'.
mbCancel	Un botón con 'Cancel'.
mbAbort	Un botón con 'Abort'.
mbRetry	Un botón con 'Retry'.
mbIgnore	Un botón con 'Ignore' .
mbAll	Un botón con 'All' .
mbYesNoCancel	Un botón de cada uno

Tipos de Cajas de Diálogo

Existen en BUILDER unas constantes predefinidas para indicar el tipo de etiqueta que aparecerá en la banda superior de la caja de diálogo o un símbolo que ya es estándar en Windows:

Constante	Etiqueta
mtWarning	Presenta un símbolo de una exclamación amarilla.
mtError	Presenta el símbolo de STOP.
mtInformation	Símbolo con una caja azul y una "i"
mtConfirmation	Con un signo de interrogación verde.
mtCustom	No se presenta un símbolo, pero la etiqueta de la caja es el nombre de la aplicación.

InputBox e InputQuery



Las Cajas de Edición permiten mostrar información y solicitar que el usuario presione un botón como respuesta, pero si se quiere, utilizar para solicitud de datos por parte del

usuario, se pueden emplear las funciones ***InputBox*** (que retorna una cadena alfanumérica) junto con ***InputQuery*** (que retorna un valor **Booleano**).

InputBox

Se emplea como función que devuelve valor y su sintaxis es:

```
Variable alfanumérica = InputBox( "Cadena de identificación" ,
"Mensaje que se da", "valor por defecto");
```

Lo que escriba el usuario se almacenará en la variable alfanumérica, siempre y cuando se presione el botón OK, en caso contrario la cadena es nula.

InputQuery

Retorna *True* o *False* si el usuario presionó el botón OK (aunque se debe verificar si no cambió el contenido de la caja). La idea de utilizarlas juntas es principalmente para saber si el usuario presionó en botón OK ó el botón CANCEL, ya que al presionar el botón CANCEL la cadena es nula, pero esto sucede igualmente si presiona el botón OK dejando la caja de edición en blanco.

PROBLEMAS SUGERIDOS

1. Dada la función: $F(x) = e^x - Sen(x)$, verificar si ésta cambia de signo en un intervalo [X1, X2], utilizando puntos separados por una cantidad DELTA. En caso de ser así, determinar por el Método de Bisección, el valor de x para el cual $F(x) = 0$. Despliegue el resultado en pantalla mediante una caja de dialogo.

2. Hacer un programa en Builder C que realice lo siguiente:

   a. Calcule y despliegue el valor del seno para los ángulos de 1º a 90º creando las funciones: seno, factorial y potencia.

$$Sen(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots$$

   b. Calcule y despliegue el valor del coseno para los ángulos de 1º a 90º utilizando, para el cálculo del coseno, una función diseñada por usted que use la serie de Taylor para dicho cálculo. La mencionada serie para el coseno se expone a continuación:

$$Cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \cdots$$

La función diseñada para calcular el coseno debe utilizar las funciones factorial y potencia creadas por usted.

c. Calcule y despliegue el valor de la tangente para los ángulos de 1º a 90º utilizando las funciones: seno y coseno creadas anteriormente.

Nota 1: Las series de Taylor para el cálculo del seno y del coseno utiliza el valor de X en radianes.

Nota 2: Utilice funciones con paso de parámetros.

CAPITULO IV: ARREGLOS

1.- INTRODUCCION

Uno de los problemas más comunes en los diversos sistemas de información, es el tratamiento o procesamiento de un gran volumen de datos o de información. Las variables o componentes visuales manejados hasta ahora, no pueden ayudar a resolver este problema.

Las variables usadas hasta ahora reciben propiamente el nombre de variables escalares, porque solo permiten almacenar o procesar un dato a la vez.

Por ejemplo si se quiere almacenar nombre y edad de 15 personas, con el método tradicional se ocuparan 30 variables o 30 componentes visuales, y solo es nombre y edad de 15 personas, agreguen mas datos y mas personas y tendremos problemas para manejarlos. Por eso es tiempo de empezar a analizar otro tipo de variables y de componentes.

En problemas que exigen manejar mucha información o datos a la vez, variables escalares o componentes visuales de manipulación de datos normales (*Edit*, *Label*, etc.), no son suficientes, ya que su principal problema es que solo permiten almacenar un dato a la vez.

Se necesita entonces, variables y sus correspondientes componentes visuales que sean capaces de almacenar y manipular conjuntos de datos a la vez. Es conveniente usar un

arreglo cuando, en el problema, existe una serie de datos del mismo tipo que reciben tratamientos similares, ya que nos permite repetir instrucciones mediante ciclos, lo que nos simplifica enormemente la estructura del programa.

Un arreglo es una estructura homogénea de datos, de tamaño constante y que puede accederse a cada uno de los valores por él representados mediante un índice (de tipo entero) que representa la posición relativa del elemento con respecto a los demás elementos del arreglo. Es bueno recordar que en C++Builder la primera posición, elemento o renglón es la 0 (cero), Ej.

NOMBRES

Valor	Posición
Juan	0
Pedro	1
José	2
Ana	3
Carmen	4

EDAD

Valor	Posición
18	0
20	1
25	2
30	3

Sin embargo sus problemas son similares a los de las variables normales, es decir hay que declararlos, capturarlos, hacer operaciones con ellos, desplegarlos, compararlos, etc., individualmente para cada valor o posición. La cantidad de valores que contiene el arreglo es lo que se conoce como dimensión.

Para propósitos del aprendizaje se analiza o clasifican en tres grupos diferentes los arreglos que ofrece C++Builder, ellos son;

- Arreglos tradicionales en C++ (internos dentro del programa)
- Componentes Visuales de tipo Arreglo

2.- ARREGLOS TRADICIONALES EN C++

En programación tradicional siempre se manejan dos tipos de arreglos, los arreglos unidimensionales, tipo listas ó vectores y los arreglos bidimensionales, tipo tablas, cuadros ó matrices, en ambos casos son variables que permiten

almacenar un conjunto de datos del mismo tipo a la vez, su diferencia es en la cantidad de columnas que cada uno de estos tipos contiene, como en los siguientes ejemplos;

A. Arreglos Unidimensional

Nombre

Juan	Pedro	José	Ana	Carmen

Edad

18	20	25	30

B. Arreglos Bidimensional

COMPAÑIA ACME

ING MENS VTAS (MILES DE $)

SUCURSAL	ENE	FEB	MAR	ABR	MAY
A	10	12	15	10	9
B	8	7	5	9	6
C	11	18	20	14	17

PRUEBA DE ADMISION

CALIFICACIONES

NOMBRE	MAT	FIS	VERBAL	BASICA
JUAN	10	10	10	10
JOSE	8	8	8	8
PEDRO	5	5	5	5
ANA	18	18	18	18

Como se observa, la diferencia principal entre un arreglo unidimensional, y un arreglo bidimensional, son las cantidades de columnas ó filas que contienen.

ARREGLOS UNIDIMENSIONAL

Un arreglo unidimensional se define como una variable que permite almacenar un conjunto de datos del mismo tipo organizados en una sola columna o fila. En álgebra y física se les conoce con el nombre de vectores.

Los procesos normales con un arreglo o con sus elementos, incluyen declarar todo el arreglo, capturar sus elementos, desplegarlos, realizar operaciones con ellos, desplegarlos, etc.

Para declarar un arreglo unidimensional se usa el siguiente formato;

```
TipoDato nombre [cant máxima de elementos];
```

Ejemplos:

```
int edades[12];
float sueldos[10];
AnsiString municipios[5];
```

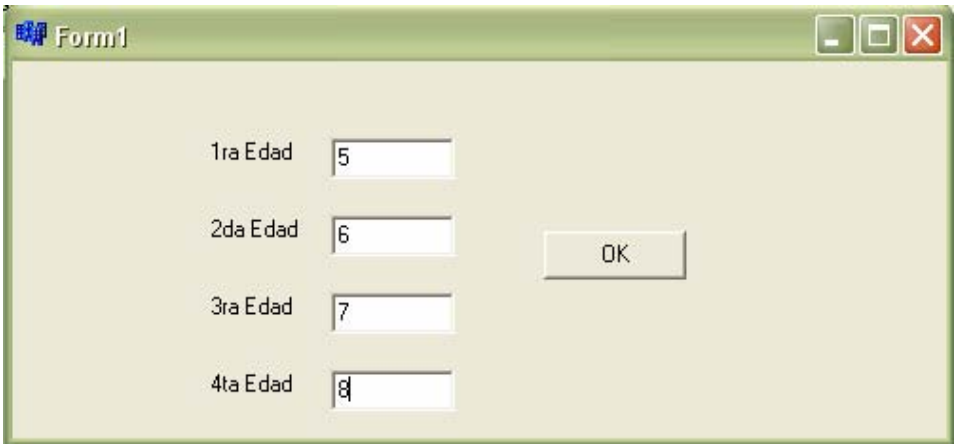
notas:

- La declaración de un arreglo se puede hacer en dos lugares diferentes, dependiendo de si sólo se usa un botón de órdenes en la pantalla, o si dos o más botones de órdenes.
- Si un solo botón, en toda la ventana, va a realizar todos los procesos (declaración, captura, operaciones, comparaciones, despliegue), con el arreglo, basta con hacer la declaración del arreglo, al principio del evento onclick, como lo muestra el programa ejemplo.
- Recordar también que la primera posición o elemento en un arreglo es la posición o elemento 0 (cero).
- El dato capturado, proviene, de momento, de un componente visual escalar y por tanto para capturas se deben de usar tantos componentes visuales como elementos tenga el arreglo, por ejemplo, para capturar una lista de 4 edades (el código y pantalla se da un poco mas adelante):

a) Programa

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{ // declaración
int edad[4];
// carga o captura del arreglo
        edad[0]=Edit1->Text.ToInt();
        edad[1]=Edit2->Text.ToInt();
        edad[2]=Edit3->Text.ToInt();
        edad[3]=Edit4->Text.ToInt();
}
```

b) Pantalla de corrida



Para el caso de operaciones y comparaciones con todos los elementos del arreglo a la vez, se deberá usar un ciclo for con una variable entera que varíe el índice del arreglo, para el ejemplo anterior, suponer que se quieren convertir todas las edades a meses, el código seria;

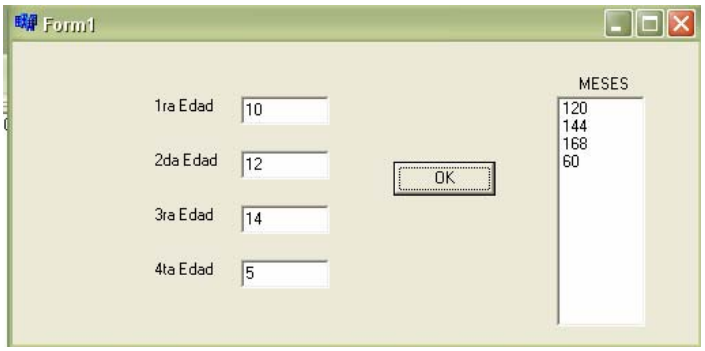
```
void __fastcall TForm1::Button1Click(TObject *Sender)
{ // declaración
        int edad[4];
        int Indice;
// carga o captura del arreglo
        edad[0]=Edit1->Text.ToInt();
```

```
        edad[1]=Edit2->Text.ToInt();
        edad[2]=Edit3->Text.ToInt();
        edad[3]=Edit4->Text.ToInt();
// conversión a meses
        for(Indice = 0; Indice <= 3; Indice++)
                edad[Indice] = edad[Indice]*12;
}
```

Recordar que todos los datos internos del arreglo estarán almacenados en la memoria ram del computador, para despliegues se puede usar un componente visual que permite manipular un conjunto de datos a la vez, el *ListBox*, pero se tiene que usar un ciclo for para ir añadiendo o agregando elemento por elemento, como se observa en el ejemplo que se ha venido desarrollando, en este caso se quiere desplegar las cuatro edades convertidas a meses:

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{// declaración
        int edad[4];
        int Indice;
// carga o captura del arreglo
                edad[0]=Edit1->Text.ToInt();
                edad[1]=Edit2->Text.ToInt();
                edad[2]=Edit3->Text.ToInt();
                edad[3]=Edit4->Text.ToInt();
// conversión a meses
        for(Indice = 0; Indice <= 3; Indice++)
                edad[Indice] = edad[Indice]*12;
// CONVERSION A ALFANUMERICO
//Y ALMACENANDO EN ListBox
        for(Indice = 0; Indice <= 3; Indice++)
                ListBox1->Items->Add(edad[Indice]);
}
```

La pantalla de salida es;



Solo recordar que para capturar un arreglo de alfanuméricos, primero se deberá usar un componente visual por cada elemento del arreglo y segundo el traspaso de dato es directo, y para desplegarlo en un componente *ListBox* de igual forma es directo, como en el siguiente ejemplo:

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{ // declaración
        AnsiString Ciudad[3];
        int I;
//Captura de elementos del arreglo
        Ciudad[0]=Edit1->Text;
        Ciudad[1]=Edit2->Text;
        Ciudad[2]=Edit3->Text;
// Pasando arreglo a ListBox
        for (I = 0; I <= 2; I++)
                ListBox1->Items->Add(Ciudad[I]);
}
```

Pantalla de salida

PROBLEMAS SUGERIDOS

1. Capturar y desplegar 5 precios de productos cualesquiera, usando dos panel, uno para capturar y uno para desplegar.
2. Capturar 4 sueldos en un panel, desplegarlos aumentados en un 25% en otro panel.
3. Capturar los datos de 5 productos comprados en una tienda, incluyendo nombre, precio y cantidad en sus 3 arreglos respectivos, después calcular un cuarto arreglo con el gasto total por cada producto desplegarlo todo en un segundo panel e incluir también el gran total.
4. Capturar en un arreglo solamente 6 números múltiplos de 5. Se debe de capturar los números uno a uno hasta que se completen los 6.

ARREGLOS BIDIMENSIONALES

Un arreglo bidimensional se define como un conjunto de datos del mismo tipo organizados en dos o más columnas como una tabla

Para procesar internamente todos los elementos de la tabla se necesitan dos ciclos for, uno externo para controlar la fila y uno interno para controlar la columna.

Para declarar un arreglo bidimensional se usa el siguiente formato;

```
TipoDato nombre [cant máxima de filas] [cant máxima de columnas];
```

Ejemplos:

```
int Calificaciones[12][4];
float IngresosMensuales[20][12];
```

Para las capturas, se deberán usar tantos componentes *Edit* como celdas tenga la tabla y en despliegue usar tantos componentes *ListBox* como columnas tenga la tabla, estos métodos son provisionales mientras se analizan los componentes visuales apropiados y respectivos.

Por ejemplo, capturar un arreglo bidimensional que nos muestre el peso en Lb de los tres jugadores claves de 4 equipos de fútbol, desplegarlos en otro arreglo pero convertidos a kg. (una libra = .454 kg.), el programa y la pantalla de salida son;

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{ // declaración
        double lbrs[3][4];
        int fil, col;
//Captura de elementos de la tabla
                lbrs[0][0]=Edit1->Text.ToDouble();
                lbrs[0][1]=Edit2->Text.ToDouble();
                lbrs[0][2]=Edit3->Text.ToDouble();
                lbrs[0][3]=Edit4->Text.ToDouble();
                lbrs[1][0]=Edit5->Text.ToDouble();
                lbrs[1][1]=Edit6->Text.ToDouble();
                lbrs[1][2]=Edit7->Text.ToDouble();
                lbrs[1][3]=Edit8->Text.ToDouble();
                lbrs[2][0]=Edit9->Text.ToDouble();
                lbrs[2][1]=Edit10->Text.ToDouble();
                lbrs[2][2]=Edit11->Text.ToDouble();
                lbrs[2][3]=Edit12->Text.ToDouble();
//conversión tabla lbrs a kgrs
        for(fil = 0; fil <= 2; fil++)
                for(col=0;col<=3;col++)
                        lbrs[fil][col] = lbrs[fil][col]*.454;
//pasando la tabla a los ListBox de resultados;
        for(fil = 0; fil <= 2; fil++)
                {ListBox1->Items->Add(FormatFloat("###.##",lbrs[fil][0]) );
                 ListBox2->Items->Add(FormatFloat("###.##",lbrs[fil][1]) );
                 ListBox3->Items->Add(FormatFloat("###.##",lbrs[fil][2]) );
                ListBox4->Items->Add(FormatFloat("###.##",lbrs[fil][3]) );
// observe, hay solo un for por fila, y los cuatro ListBox con
// las columnas constantes
}
```

Debe tomar en cuenta que:

a. La dimensión máxima del arreglo, con la que éste es declarado, debe ser constante, es decir, no debe depender de valores almacenados en variables.

b. El índice debe ser de tipo entero positivo, mientras que los valores almacenados en el arreglo pueden ser de cualquier tipo, pero todos del mismo tipo.

c. El índice siempre comienza en cero (0).

d. Un arreglo puede tener tantos o menos elementos de los indicados en la declaración.

e. C++Builder no indica cuando se ha sobrepasado el límite de un arreglo (dimensión) por lo que hay que tener cuidado para no ocasionar errores graves y difíciles de detectar.

f. En un arreglo bidimensional, el primer índice referencia la fila y el segundo referencia la columna de la matriz o tabla.

Problemas sugeridos

1. Construir un arreglo bidimensional que contenga los costos fijos de cuatro productos cualesquiera, que se producen en tres plantas diferentes de una empresa maquiladora.
2. Construir un arreglo bidimensional que contenga los ingresos mensuales por ventas durante los tres primeros meses del año de cuatro sucursales de una cadena de auto refacciones, agregar al final un arreglo unidimensional que muestre los ingresos mensuales totales por meses y un segundo arreglo unidimensional que muestre los ingresos mensuales totales por sucursal.
3. Construir un arreglo bidimensional que contenga las comisiones ganadas por tres vendedores diferentes de 5 tipos de muebles de línea blanca, agregar además arreglos unidimensionales de comisiones totales por vendedores y por tipo de mueble y otras de comisiones promedio por vendedores y por tipo de muebles.

   Sugerencia; primero hacer un bosquejo en papel de como quedaran los arreglos en pantalla.

4. El Sr. Pedro Pérez heredó una fortuna en Dólares que le dejó su abuelo antes de éste morir. El Sr. Pérez desea decidir qué hacer con el dinero y se plantea las siguientes opciones:

    I. Guardarlo debajo de la cama y esperar a que liberen el dólar

    II. Comprar bonos en dólares

    III. Viajar a Miami y boncharse todo el dinero

    IV. Quedarse en Venezuela e invertir en bienes raíces, autos, etc. antes del cambio de moneda.

    Independientemente de la opción que Pedro escoja, el beneficio que puede obtener de ese dinero, dependerá del futuro del país. Pedro visitó un brujo y este le vaticinó que podía suceder una de las siguientes posibilidades:

    I. Que el dólar baje vertiginosamente

    II. Que el petróleo baje y Venezuela quiebre

    III. Que Venezuela se convierta en un país próspero

    IV. Que expulsen a todos los Venezolanos de USA

    Pedro Pérez ha calculado una matriz que contiene los beneficios que ha de ganar con cada opción y para cada una de las posibilidades futuras, donde Bkj corresponde al beneficio que ganaría si escoge la opción k y sucede en el futuro la posibilidad j. Pedro Pérez desea escoger la mejor opción (que maximice su beneficio) y sabe que existe un método que escoge, de la matriz de beneficios, el menor beneficio para cada opción. De este vector se extrae la opción con mayor beneficio y esa es la solución que necesita Pedro Pérez.

    Diseñe un programa en Builder C que dada la matriz de beneficios, implemente el algoritmo antes descrito para ayudar al Sr. Pedro Pérez a invertir bien su dinero, desplegando el resultado en una caja de diálogo.

3.- COMPONENTE StringGrid (Adicional)

Este componente es de los más importantes para el procesamiento de muchos datos. Permite concentrar y mostrar gran cantidad de información para la vista del usuario.

Este componente presenta y procesa conjuntos de datos de tipo alfanumérico en forma tabular, es decir en forma de tablas ó matrices, por ejemplo:

COMPAÑIA ACME ING MENS VTAS (MILES DE $)

SUCURSAL	ENE	FEB	MAR	ABR	MAY
A	10	12	15	10	9
B	8	7	5	9	6
C	11	18	20	14	17

Recordar que son los datos numéricos internos los que se procesan (es decir, se capturan, se realizan operaciones con ellos, se despliegan, etc.), es la información externa quien le da sentido.

Algunas de sus propiedades y métodos más interesantes, son:

ColCount.- Determina la cantidad de columnas que contendrá la tabla.

RowCount.- Determina la cantidad de filas que contendrá la tabla.

Fixedcol , Fixedrow.- Determinan la cantidad de columnas y filas fijas o de encabezado, estas propiedades comienzan en 0.

+Options, goediting = true; Para que permita editar o capturar datos al usuario.

+Options, gotab = true; Para que el usuario pueda navegar entre celdas usando la tecla del tabulador.

Cells[columna][fila], Es la propiedad mas importante, porque es la que permite el acceso a cualquier celda de la tabla, Ej.

```
StringGrid1->Cells[1][1] = "PATO";
Edit1->Text = StringGrid1->Cells[0][0];
Edit2->Text = StringGrid->Cells[0][1] * 3.1416;
```

Nota: observe que a pesar de ser de tipo alfanumérico, se pueden efectuar algunas operaciones matemáticas directamente con las celdas. Aunque es mucho más seguro sobre todo en el caso particular de la suma, convertirlos a Cells[columna][fila].ToInt( ) o ToDouble(), Ej.;

```
Edit2->Text = StringGrid1->Cells[1][0].ToInt()+5;
```

Para procesar todos los elementos del StringGrid, se deben usar dos ciclos for, uno externo para controlar las columnas, y uno interno para controlar las filas (observar que es lo inverso de los arreglos normales).

Ejemplo: capturar un arreglo de 3 * 4 enteros, y restarles 4 después;

Pantalla de Diseño:

Programa;

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{ int col, fil ;
        for(col = 0; col <= 2; col++)
                for(fil = 0;fil <= 3; fil++)
                        {StringGrid1->Cells[col][fil] =
                        StringGrid1->Cells[col][fil]. ToInt()- 4 ;};
}
```

Pantalla de Corrida:

Un proceso muy común con tablas o arreglos bidimensionales, es agregarles totales y promedios ya sea por columna o por fila o ambas, por ejemplo:

COMPAÑIA ACME

INGRESOS MENSUALES

(MILES DE PESOS)

	ENE	FEB	MARZO	TOTALSUC	PROMSUC
SUC A	1	2	3	6	2
SUC B	4	5	6	15	5
SUC C	7	8	9	24	8
SUC D	10	11	12	33	11
TOTMES	22	26	30		
PROMMES	5.5	6.5	7.8		

En este ejemplo aparte de la tabla inicial se necesitan 4 vectores, dos para totales y dos para promedios.

El código es sencillo:

```
//declaración
        float tabla[4][3], totsuc[4], promsuc[4];
        float totmes[3], promes[3];
/*observar que hay tamaños de arreglos que referencian fila
```

```
 y otros que referencian columnas, se supone que la tabla
ya esta capturada.*/
/*código para operaciones para totales y promedios
renglones por sucursal: */
        for(fil = 0; fil <= 3; fil++)
                for(col = 0; col <= 2; col++)
                        totsuc[fil]=totsuc[fil]+tabla[fil][col];
        for(fil = 0; fil <= 3; fil++) promsuc[fil] = totsuc[fil] / 3.0;
//operaciones para totales y promedios por mes
        for(fil = 0; fil <= 3; fil++)
                for(col = 0; col <= 2; col++)
                        totmes[col]=totmes[col]+tabla[fil][col];
        for (col = 0; col <= 2; col++) prommes[col] = totmes[col] / 4.0;
```

PASANDO ARREGLOS A FUNCIONES COMO PARÁMETROS

Los arreglos solamente se envían como parámetros a una función por referencia ó dirección, ya que el nombre del mismo contiene la dirección al primer elemento del arreglo.

Ejemplo:

```
{       //declaración
         int A[10][10];
        /* llamado a la función. Observar que solo se coloca el
           nombre del arreglo sin subíndices */
        funcion (A);
}
// diseño de la función
// note que sólo el primer corchete puede ir vacío
void funcion (int A[ ][10])
{       Instrucciones;
}
```

PROBLEMAS SUGERIDOS

1. Construir un programa que despliegue los costos fijos de tres diversos productos que se fabrican en cuatro sucursales de una empresa MAQUILADORA.
2. Construir un programa que contenga los ingresos por ventas mensuales de los 4 primeros meses del año de tres sucursales de una cadena refaccionaría, agregar listas de ingresos totales por mes e ingresos promedios por sucursal.
3. Construir un arreglo que contenga las calificaciones de 5 materias de cuatro alumnos cualesquiera, incluir promedios de calificaciones por materia y por alumno.
4. Se tienen N empleados en una compañía y se ha ideado llenar un arreglo unidimensional A con los sueldos de los empleados, un arreglo B con las asignaciones totales de cada empleado y un arreglo C con las deducciones de cada uno. Crear un arreglo T que contenga el neto a pagar a cada empleado.

$$\text{NETO} = \text{SUELDO} + \text{ASIGNACIONES} - \text{DEDUCCIONES}$$

5. Realice un programa que a partir de una matriz de NxM, genere dos vectores: el primero de orden N donde cada componente corresponde a la suma de cada fila y otro vector de orden M, formado por la suma de los elemento de cada columna de la matriz.

   Por ejemplo, para una matriz de 3x4:

1	3	7	2	13
5	6	9	1	21
4	1	7	2	14
1	4	6	5	16

11	14	29	10

6. Diseñe una función en Borland C que verifique si un vector A de tamaño N es simétrico y que devuelva el valor uno (1) en caso afirmativo y el valor cero (0) en caso negativo. Un vector es simétrico si se cumple, en todos los casos, que

el primer elemento es igual al último, el segundo al penúltimo, el tercero al antepenúltimo, y así sucesivamente. La función debe tener el siguiente encabezado: unsigned char Simétrico(int N, int [A])

7. Dada una matriz numérica, se denomina “Elemento Punto de Silla” aquel que es simultáneamente máximo de su fila y mínimo de su columna. Determinar mediante un programa en Builder C todos los puntos de silla de un matriz de orden NxM que se encuentra en un archivo tipo texto llamado ‘MATRIZ.DAT’. Añada el valor, la fila y la columna de esos puntos de silla al mismo archivo.

Otra forma de desplegar los valores de un arreglo es utilizando los componentes *ListBox*.

5.- COMPONENTE ListBox(Standard)

Este componente permite procesar visualmente un conjunto de elementos de tipo alfanumérico.

- Hereda muchas de las propiedades y métodos de *TStringList* y *TStrings*, más algunas propiedades y funciones propios.
- Se puede añadir, eliminar e insertar ítems en la lista usando los métodos Add, Delete, Insert con la propiedad Items, que también es de tipo *TStrings*.
- Si se quiere que *ListBox* presente varias columnas, solo cambiar el valor de la propiedad *Columns.*
- Para ordenar o clasificar los ítems, usar la propiedad *Sorted*.
- Se puede permitir que un usuario realice una selección múltiple, poniendo la propiedad *MultiSelect* en true, la propiedad *ExtendedSelect* determina como se realiza la selección múltiple.
- Para determinar cual ítem en particular esta seleccionado solo validar la propiedad *Selected.*
- Para conocer cuantos ítems se han seleccionado revisar los valores de la propiedad *SelCount*.

1. Propiedades:

BorderStyle.- Despliega la lista con un marco sencillo o sin marco.

Canvas.- Se utiliza para asociar un área de dibujo o de imagen a un ítem de la lista (consultar ayuda de esta propiedad en *ListBox* puesto que es interesante).

Columns.- Define una determinada cantidad de columnas para su despliegue dentro del *ListBox*.

ItemIndex.- Se utiliza para seleccionar la posición o índice de un ítem o elemento en la lista.

Items.- Es la propiedad que más se ha venido usando, se utiliza para acceder las diversas propiedades y funciones de las strings en la lista.

Sorted.- Se usa para ordenar alfabéticamente los elementos de la lista (ListBox1->Sorted = true;)

Style.- Define diversos o varios estilos o formas de *ListBox*.

Funciones:

Clear.- Elimina todos los elementos a la vez.

Add.-Para cargar o capturar sus elementos. No se permite cargar directamente sus datos o elementos por el usuario del programa, solo usando un código como el siguiente:

```
ListBox1->Items->Add (Edit1->Text);
```

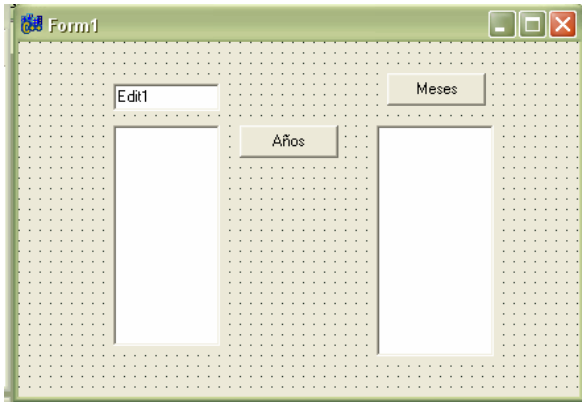
- Se pueden usar todas las funciones, para insertar, eliminar, contar, etc.
- Si los valores son de tipo "numérico", se pueden efectuar pasando los valores de *ListBox* a una variable numérica de tipo arreglo y procesando esta ultima variable.
- Si es necesario pasar el contenido de *ListBox1* a *ListBox2*, solo usar la propiedad *Text* en ambas, Ej.;

```
ListBox2->Text=ListBox1->Text;
```

Ejemplo:

Para este caso se pide capturar 5 edades en un *ListBox*, luego sumarle 3 años a cada una usando el mismo *ListBox*, y desplegarlas en un segundo *ListBox* convertidas a meses.

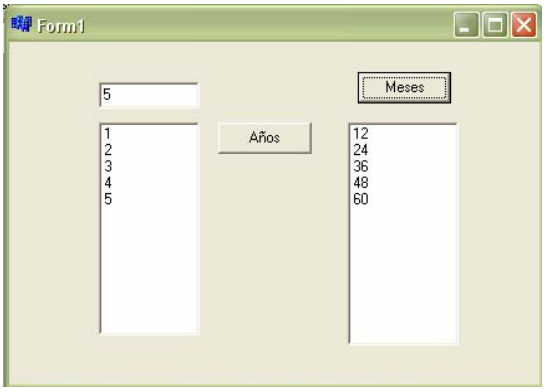
Pantalla de diseño:



Código:

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{ ListBox1->Items->Add(Edit1->Text); }
//---------------------------------------------------
void __fastcall TForm1::Button2Click(TObject *Sender)
{ int Indice;
        for (Indice = 0; Indice <= 4; Indice++)
                {ListBox2->Items->Add (ListBox1->Items->
                                        Strings[Indice].ToInt() * 12);}
}
```

Pantalla de salida:



PROBLEMAS SUGERIDOS

1. Capturar y desplegar 5 muebles de hogar
2. Capturar en una lista 5 estaturas en centímetros, desplegarlas convertidas en pulgadas (1 pulgada = 2.54 cm.)
3. Capturar en una lista 6 pesos en kg., convertirlos posteriormente a libras y solo desplegar en el listbox respectivo pesos mayores de 120 libras.
4. Capturar en sus cuatro listas respectivas, matricula, nombre, calificación de matemáticas, calificación de física, de 5 alumnos, calcular posteriormente una lista de promedios, y en los listbox de salida solo desplegar los datos de alumnos aprobados.

CAPITULO V: ESTRUCTURAS Y UNIONES

1.- INTRODUCCION

Bajo el concepto de **Estructura** se tiene la reunión de un conjunto de datos heterogéneos en su tipo, pero relacionados entre ellos.

Bajo el concepto de **Unión** se tiene la reunión de un conjunto de datos heterogéneos en su tipo que comparten la misma localidad de memoria.

2.- DATOS DE TIPO ESTRUCTURA Ó STRUCT

Tal como se indicó previamente, este concepto abarca una estructura heterogénea de datos, denominados campos, a los cuales se les accede mediante un nombre o identificador. Esta es otra diferencia con respecto a los arreglos, en los cuales tal acceso se realiza por un índice.

El formato para la declaración del tipo es:

```
struct Identificador {
        Tipol Campal;
        Tipo2 Campo2;
        TipoN CampoN;
} Variable;
```

Donde: *Tipo1, Tipo2 ... TipoN* es cualquier tipo predefinido u otro tipo definido por el usuario

Ejemplo:

```
struct Alumno {
        char Cedula[8];
        char Seccion[2];
        unsigned int Nota;
};
struct Alumno Persona;
```

3.- TRABAJO CON ESTRUCTURAS

Para acceder a una variable estructura se realiza mediante su identificador. Para acceder a uno de los campos que lo componen se efectúa empleando un identificador de campo, compuesto por el nombre de la variable estructura y el nombre del campo separados por un punto. Las operaciones sobre un campo serán las mismas que correspondan a su tipo.

La única operación que se puede hacer con una variable tipo estructura como tal, es la asignación, es decir, pueden ser copiados todos los campos de una variable estructura a otra variable estructura del mismo tipo.

Una estructura puede ser pasada como parámetro a una función, como parámetro actual siempre y cuando el parámetro formal sea del mismo tipo. Para acceder a una variable estructura dentro de una función, se realiza mediante su identificador y el nombre del campo separados por el operador flecha (->).

4.- EJEMPLO

Escriba y codifique el siguiente programa para observar y analizar como funcionan algunas las instrucciones antes mencionadas. En particular, esta codificación corresponde a un programa que lee dos valores complejos (de la forma a+bi) y los suma:

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
        //Declaraciones
                struct Complejo {
                        float R;
                        float I;
                };
                  struct Complejo Comp1, Comp2, Resultado;
                  void Lee_Complejo (struct Complejo *Numero);
          // Ejecución
                Lee_Complejo (&Comp1);
                Resultado.R := Compl.R+Comp2.R;
                Resultado.I := Comp1.I+Comp2.I;
                Label1->Caption = Resultado.R;
                if (Resultado.I > 0)
                        Label1->Caption = Label1->Caption + '+' +
                                          AnsiString(Resultado.I);
                else
                        Label1->Caption = Label1->Caption + '+' +
                                          AnsiString(fabs(Resultado.I));
}
//Función para buscar datos
void Lee Complejo (struct Complejo *Numero)
        {float Re, In;
        Label2->Caption = "Introduzca parte real";
        Re = Edit1->Text.ToDouble();
        Label3->Caption = "Introduzca parte imaginaria";
        In = Edit2->Text.ToDouble();
        *Numero->R = Re;
        *Numero->I = In;
}
```

5.- UNIONES O UNIÓN

Los tipos uniones se definen como posiciones de memoria que son compartidas por varias variables de diferentes tipos.

Un ejemplo de la declaración de un tipo unión es:

```
unión unido {int AB; float C; char Z};
unión unido A;
```

6.- EJEMPLOS

Escriba y codifique el siguiente programa para observar y analizar como es el trabajo con uniones:

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
//Declaraciones
        unión Enteros {
                unsigned char Baja_Alta[2];
                unsigned int Numero;
        }
        unión Enteros PP;
        void Lee_Datos (union Enteros *PP);
//Ejecución
        Lee_Datos(&PP);
        Label1->Caption = AnsiString (PP.Baja_Alta[0]) + ‘ ‘ + AnsiString
                                (PP.Baja_Alta[l]);
}
//Función para buscar datos
        void LeeJDatos (unión Enteros *PP)
        {
                *PP->Numero = Edit1->Text.ToInt();
        }
```

7.- PROBLEMAS PROPUESTOS

1. Amplíe el programa ofrecido como ejemplo para suma de números complejos a un programa que permita al usuario realizar y calcular la resta, multiplicación, el conjugado y valores absolutos de números complejos.

CAPITULO VI: MANEJO DE ARCHIVOS

1.- INTRODUCCION

En este capitulo se analiza en general el problema de la permanencia de los datos. Hasta ahora todos los datos capturados, calculados, creados, etc. al terminar o cerrarse el programa se pierden y es necesario volver a capturarlos, etc., en la siguiente ejecución o corrida del programa. Este problema se puede resolver usando el concepto de archivos, que son medios permanentes de almacenamiento de datos en los dispositivos o periféricos apropiados, generalmente disco.

2.- ARCHIVOS

Un archivo se puede definir como un conjunto de informaciones almacenado sobre un periférico de entrada/salida e identificado por un nombre. Un archivo permite que este conjunto de informaciones pueda ser almacenado en un dispositivo de almacenamiento secundario y recuperado posteriormente.

C++Builder permite el trabajo y la manipulación de datos con diversos tipos de archivos, pero en este texto sólo trataremos con archivos de tipo Texto ya que presentan algunas ventajas con respecto a otros:

- Un archivo texto puede ser manipulado por otros lenguajes de programación, por otras aplicaciones Windows o de cualquier otra plataforma de computación.
- El intercambio de información con un archivo texto se lleva a cabo de manera similar a como se lleva con el teclado y la pantalla.

3.- ARCHIVOS TIPO TEXTO

Debido a que los archivos de caracteres se emplean frecuentemente, existe un tipo de archivo estándar denominado Archivo de Texto. Estos archivos están compuestos por líneas separadas por marcas de fin de línea (EOLN) y los datos colocados son almacenados uno después del otro.

Un archivo externo es típicamente un nombre de archivo de disco (tal como aparecen en Windows).

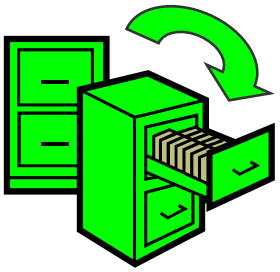
Las operaciones básicas de manejo y comunicación con archivos tipo texto se reducen a:

- Leer datos de un archivo existente
- Escribir resultados sobre un archivo nuevo
- Agregar información a un archivo ya existente

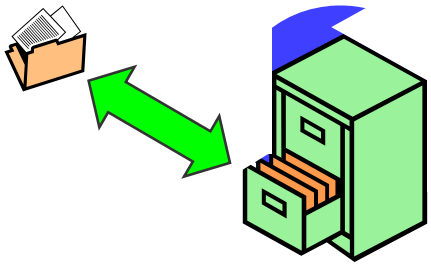
Antes de realizar cualquiera de estas tres operaciones, por un concepto impuesto por los sistemas operativos, éste debe ser abierto, lo cual significa solicitar al sistema que nos permita acceso al mismo.

De la misma manera, al terminar de realizar cualquiera de las tres operaciones antes mencionadas sobre un archivo texto, éste debe ser cerrado. Cerrar un archivo es una operación donde el sistema operativo protege al archivo de otras operaciones. Si un archivo no es cerrado se corre el riesgo de perder parte o la totalidad de su contenido. Para manipular información con archivos texto, es necesario recordar entonces:

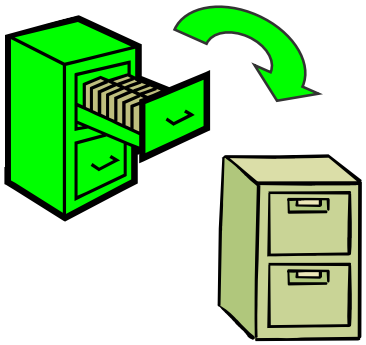
1. Abrir el archivo



2. Leer o escribir datos en el archivo



Cerrar el archivo



C++Builder obliga a que la comunicación entre el programa y un archivo contenido en un dispositivo de almacenamiento secundario se realice a través de un variable de tipo archivo. Por lo cual si el programa va a hacer uso de una variable llamada Fp para manipular un archivo texto debe declararla antes de utilizar el archivo mediante la siguiente instrucción:

```
FILE *Fp;
```

4.- Funciones para Manejo de Archivos

Abrir un Archivo texto: *fopen( )*

Esta función abre un archivo tipo texto, es decir, le solicita al sistema acceso al archivo de texto contenido en la unidad de almacenamiento secundario, llamado con el nombre especificado y lo asocia a la variable tipo archivo declarada con anterioridad

```
Fp = fopen( “Nombre”, “Modo”);
```

De tal manera que todas las operaciones siguientes que utilicen *Fp*, trabajarán sobre el archivo externo identificado en *Nombre*, donde:

- ***Fp*** es una variable tipo FILE que se ha de asociar al archivo externo señalado en “Nombre”.

- ***Nombre*** es una cadena alfanumérica que contiene: el nombre del archivo físico asociado, indicando unidad, directorios, nombre y extensión. Si ya existe un archivo con el mismo nombre, la información existente se pierde.
- ***Modo*** indica de que forma se abrirá el archivo, es decir, para qué va a ser utilizado dicho archivo. A continuación se muestran los *"Modos"* asociados al uso de archivos tipo Texto:

  rt*:* Abre un archivo existente para leer información. Se comienzan a leer desde el primer dato y en forma secuencial. La apertura del archivo es solo para lectura por lo que no será posible realizar modificaciones.

```
Fp  =  fopen ("Nombre", "rt");
```

  wt*::* Abre un archivo por primera vez para escribir datos en el, en caso de existir un archivo con el mismo nombre y en la misma localización, destruye toda la información allí contenida. No es posible leer datos a través de este procedimiento.

```
Fp  =  fopen ("Nombre", "wt");
```

  at*:* Abre un archivo existente para anexarle información al final del mismo. La información se "pega" a partir del último dato existente en el archivo.

```
Fp  =  fopen ("Nombre", "at");
```

  La información se "pega" a partir del último dato existente en el archivo.

Una cualquiera de las funciones anteriores debe ser llamada antes de emplear las funciones de Entrada/Salida para archivos: *fscanf( )* y *fprintf( ).* Que en su forma más general se invocan como:

```
fscanf ( Fp , "Formato", &V1, &V2)

fprintf ( Fp , "Formato", V1, V2)
```

Cerrar un archivo texto: *fclose( )*

Cierra un archivo existente, necesario en el C++Builder porque de lo contrario se corre el riesgo de perder información.

```
fclose ( Fp );
```

donde *Fp* es una variable tipo FILE que se ha asociado al nombre de un archivo externo en la apertura. Cierra el archivo, y coloca la marca de finalización de archivo si este se abrió para escritura.

5.- ESCRIBIR DATOS EN UN ARCHIVO Ó CREAR UN ARCHIVO:

Los datos se agregan al principio del archivo. En el ejemplo la variable *fe* fue declarada tipo FILE. Las variables a, b, y c fueron declaradas tipo int.

Ejemplo:

```
fe = fopen ("nombre.dat", "wt");
fprintf(fe, "%d %d %d", a, b, c);
fclose (fe);
```

6.- AGREGAR DATOS A UN ARCHIVO:

Los datos se agregan al final del archivo. En el ejemplo la variable *fe* fue declarada tipo FILE. Las variables a, b, y c fueron declaradas tipo int.

Ejemplo:

```
fe = fopen("a:nombre.dat", "at");
fprintf (fe, "%d %d %d", a, b, c);
fclose (fe);
```

7.- LEER DATOS DE UN ARCHIVO:

Los datos se comienzan a leer desde el principio del archivo. La variable *fe* fue declarada tipo FILE. Las variables a, b, y c fueron declaradas tipo float.

Ejemplo:

```
fe = fopen("a:\temp\nombre.dat", "rt");
fscanf(fe, "%f %f %f", &a, &b, &c);
fclose (fe);
```

8.- FIN DE ARCHIVO: feof ( )

En muchos casos nos encontramos en la necesidad de leer una gran cantidad de datos de un archivo sin saber con exactitud cuantos datos contiene dicho archivo. La forma de resolver este problema es buscar la marca de fin de archivo que coloca el sistema cuando el mismo es creado. La función *feof()* devuelve un valor distinto de cero si el apuntador de datos del archivo se encuentra al final del archivo. Su sintaxis es:

```
F = feof ( fe );
```

donde *fe* es una variable tipo *FILE* que se ha de asociar a un archivo externo, y *F* es una variable booleana que obtendrá el valor true si se llegó al final del archivo y false si no se llegó.

LEER DATOS DE UN ARCHIVO UTILIZANDO LA MARCA DE FIN DE ARCHIVO:

Ejemplo:

```
{
        char J = 1;
        int A[ll];
        FILE *Archivo;
        Archivo = fopen("A:\Datos.dat", "rt"),
        do{
                fscanf(Archivo, "%d\n", &A[J]);
                StringGrid1->Cells[J][0] = A[J];
                J++;

        }while (!(feof(Archivo)));
        fclose(Archivo);
}
```

9.- FORMATO DEL printf Y scanf

$$scanf\ (\text{"\%}[\,flags\,][Tam\,][.\,\text{Pr}\,ec\,][MOD\,]Tipo\text{"},\&Var1,\&Var2,\cdots,\&VarN\,)$$

$$pr\,\text{int}\ f\,(\text{"\%}[\,flags\,][Tam\,][.\,\text{Pr}\,ec\,][MOD\,]Tipo\text{"},Var1,Var2,\cdots,VarN\,)$$

- **%** indica el sitio donde se escribirán los datos
- **flags** indica cómo se presentará la data
- **Tam** especifica el número de caracteres que contendrá el dato a escribir ó leer
- **.Prec** nos da la precisión
- **MOD** son operadores que modifican el tamaño de la variable
- **Tipo** indica el tipo de dato de la variable
- **Variable (1, 2, ..., N)** es el nombre de la variable que se desea escribir

TIPO DEL ARGUMENTO

TIPO	SIGNIFICADO
d, I	Número decimal tipo int
o	Número octal sin signo tipo int
x, X	Número hexadecimal sin signo tipo int usando un cero inicial
u	Número decimal sin signo tipo unsigned int
c	Carácter sencillo tipo char o int. En scanf no salta los espacios en blanco, en este caso utilice %ls.
s	Cadena de caracteres tipo char*. En scanf agrega \0 al final
f	Decimal punto flotante tipo float
e, E	Decimal punto flotante tipo float en notación científica en minúscula ó mayúscula respectivamente
g, G	Igual a la anterior, pero utiliza la notación científica si el exponente es menor que -4 ó mayor a la precisión, en otro caso utiliza %f
%	No es convertido a ningún argumento, imprime un porcentaje. En scanf no hace asignación alguna.

TAMAÑO DEL ARGUMENTO

TAM	SIGNIFICADO
n	Escribe, al menos, n dígitos del argumento. Si el argumento tiene menos d dígitos, se llenan con espacios los restantes hasta completar n.
0n	Igual al anterior pero rellena con ceros.

MODIFICADOR DE TAMAÑO

MOD	SIGNIFICADO
h	Entero de un byte tipo Short int
l	Entero largo (long int) ó punto flotante de doble precisión (double flo Solamente tiene efecto en el procedimiento scanf.
L	Punto flotante de precisión extendida (long double)

CÓDIGOS DE BARRA INVERTIDA

CODIGO	SIGNIFICADO
\b	Mueve un espacio atrás.
\f	Salto de página
\n	Salto de línea
\r	Retorno de carro
\t	Tabulación horizontal
\”	Comillas dobles
\’	Comilla simple
\0	Carácter nulo
\\	Barra invertida
\v	Tabulación vertical
\o	Constante octal
\x	constante hexadecimal

10.- CAJAS DE DIÁLOGO DE DIRECTORIOS

BUILDER presenta las herramientas necesarias para que uno pueda crear una caja de diálogo que permita visualizar unidades, estructura de directorios y archivos para que el usuario pueda realizar operaciones del tipo Abrir o Guardar archivos.

Para lo anterior se dispone de los componentes:

- DirectoryListBox
- DriveComboBox
- FileListBox

De todas maneras debe conocer que también es posible emplear cajas estándar de Windows.

DriveComboBox

Este componente se encuentra en la pestaña Win 3.1 de la barra de componentes. Permite presentar las unidades y conexiones de red disponibles en el equipo que se ejecuta la aplicación.

DirectoryListBox

Este componente se encuentra en la pestaña Win 3.1 de la barra de componentes. Permite presentar el árbol de directorios que posee una unidad de disco en el formato estándar en Windows. Lo anterior implica que si el usuario hace doble click se expande (o contrae) la rama de un subdirectorio que contenga subdirectorios.

FileListBox 

Este componente se encuentra en la pestaña Win 3.1 de la barra de componentes. Presenta los archivos de un subdirectorio.

Estos tres elementos se concatenan en su accionar para que cuando un elemento cambie los otros también cambien de manera adecuada e instantánea a los ojos del usuario.

Una vez colocados los elementos anteriores sobre la forma se debe escribir las siguientes líneas de código:

```
void __fastcall TForm1::DriveComboBox1Change(TObject *Sender)
 {
   DirectoryListBox1->Drive=DriveCombobox1->Drive;
 }
// ---------------------------------------------------------------------------
void __fastcall TForm1::DirectoryListBox1Change(TObject *Sender)
 {
    FileListBox1->Directory=DirectoryListBox1->Directory;
 }
```

11.- FILE OPEN

Se emplea cuando se desea permitir al usuario abrir un archivo en su aplicación. Está encapsulado en un componente *OpenDialog*.

Es importante aclarar que este elemento simplemente recupera del usuario el nombre de un archivo. Depende del programador el escribir el código que efectivamente haga algo con ese nombre.

PROBLEMAS SUGERIDOS

2. Realice un programa que ofrezca la posibilidad de anexar datos al archivo creado anteriormente, y otra opción para ordenar de menor a mayor los datos del mismo archivo.

3. Posee N datos enteros en un archivo tipo texto ordenados de menor a mayor, diseñe un programa que coloque un valor, leído como dato, en el archivo sin que este se desordene.

4. Considere que el valor N se encuentra como primer dato del archivo.

5. Dado un archivo tipo texto con N datos del tipo real, crear un programa que calcule la media y la desviación estándar de tales valores, estos resultados deben ser guardados en el mismo archivo.

REFERENCIAS

- **La Cara Oculta de C++ Builder**. Ian Marteens. Madrid, Junio de 1999
- **Borland C++Builder™ 5 for Windows 2000 / 98 / 95 / NT Interprise Corporation**. 100 Enterprise Way, Scotts Valley, CA 95066-3249
- **Borland - C++ Win32 API**.pdf. Alex (yuekin80@hotmail.com) 20/11/2001. Descripción: Resumen-tutor de C++
- **PAUTAS DE DISEÑO DE INTERFASES GRÁFICAS BASADAS EN MODELO DE APRENDIZAJE S.O.I**., PLATAFORMAS: MICROSOFT®, LINUX Gregoria Romero E. Instituto Universitario de Tecnología Valencia. Departamento de Informática, Unidad de Investigación. Coordinación de Lenguajes de Programación. gregoria@cantv.net
- **Creación de una ventana sencilla en C++** LSCA. Israel E. García Sokabatou@hotmail.com
- **APERTURA DE TABLAS DE DATOS ACCESS CON BCB** (VERSION 3.0). By Juan Mª CASTILLO
- **Getting started with Borland C++Builder compiler . 2000** Deitel & associates. Inc and Prentice Hall
- **CURSO BASICO DE C** Por Andrés Giovvani Lara Manzano
- **"Teach Yourself C++ in 21 Days"** de Sams Publishing.
- http://www.planet-source-code.com
- http://www.lawebdelprogramador.com
- **Aprenda lenguaje así C como si estuviera de primero**. Javier García de Jalón de la Fuente, Alfonso Brazales Guerra, José Ignacio Rodríguez Garriod, Patxi Funes Martínez, Rufino Goñi Laceras, Ruben Rodríguez Tamayo. Escuela superior de ingenieros industriales. Universidad de Navarra
- **TUTORIAL SOBRE APUNTADORES Y ARREGLOS EN C** por Ted Jensen. Versión 1.2. Febrero de 2000

- **Thinking in C++, Volume 1, 2nd Edition**. Completed January 13, 2000. Bruce Eckel, President,
- **MindView, Inc.** http://www.planetpdf.com/
- **ANSI/ISO C++ Professional**  http://kickme.to/tiger/ © Copyright 1999, Macmillan Computer Publishing. All rights reserved.
- **C++ for dummies. 5th edition.** Stephen Randy Davis. Wiley publishing, Inc
- ***Manual de Programación C* Para principiantes y avanzados** by Federico Rena http://www.casarramona.com/mt/programador/
- ***Thinking in C++ 2nd edition Volume 2***: Standard Libraries & Advanced Topics 2nd Edition, Volume 2 Bruce Eckel President, MindView Inc. © 1999 by Bruce Eckel, MindView, Inc.
- ***Guía de Laboratorio de Builder C***. Prof. Belzyt González Guerra y Prof. Robustiano Gorgal Suárez. Departamento de Investigación de Operaciones y Computación. Facultad de Ingeniería. UCV.
- ***Programación Builder C. Resumen de Clases***. Prof. Belzyt González Guerra y Prof. Robustiano Gorgal Suárez. Departamento de Investigación de Operaciones y Computación. Facultad de Ingeniería. UCV.
- ***Programación en C++. Algoritmos, Estructuras de Datos y Objetos.*** Luís Joyanes Aguilar. McGraw Hill
- ***Como Programar en C/C++.*** H. M. Deitel & P. J. Deitel. Prentice Hall.
- ***Problemario de Programación***. Departamento de Investigación de Operaciones y Computación. Facultad de Ingeniería. Universidad Central de Venezuela. 2005