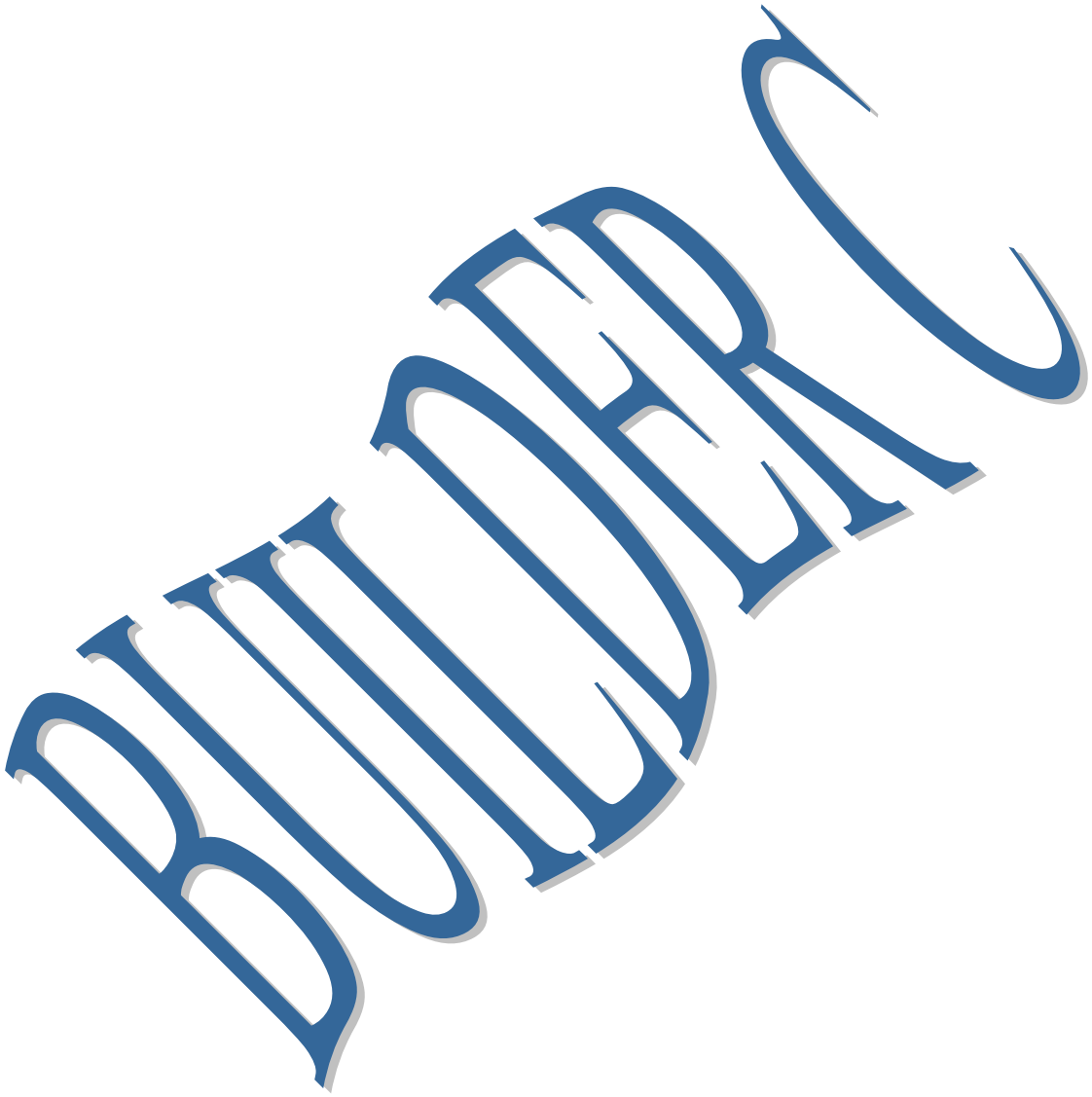


GUIA DE LABORATORIO

Prof. Belzyt González Guerra
Prof. Robustiano Gorgal Suárez

Octubre 2003

Sistema Operativo - Ambiente del Editor

1. Introducción

- En Sistema Operativo

  En esta sesión se empezará a trabajar con el Sistema Operativo Windows, el árbol de directorios y manipulación de archivos.

- En Builder C++

  En esta sesión se empezará a trabajar en el Ambiente Integrado de Desarrollo de Builder C++, en el cual escribirá las instrucciones de sus programas. Este ambiente se controla por los menús llamados tipo persiana o *pull-down*. Brindando la posibilidad de traducir el programa escrito en instrucciones de Builder C++ al código interno de la máquina.

2. El Sistema Operativo Windows

El Equipo que será utilizado no puede funcionar únicamente con sus componentes físicos (hardware). Para poder realizar alguna tarea con el computador es necesario suministrarle programas (software) que le indiquen cómo ejecutar las tareas que se requieren.

Una parte imprescindible del software es el llamado **Sistema Operativo**. El sistema operativo es un programa maestro que controla todo el trabajo del computador, y sirve de enlace entre el usuario, los componentes del equipo y el resto del sistema.

El Sistema Operativo Windows trabaja mediante un ambiente gráfico que se asienta en el uso de un dispositivo de punteo (usualmente el ratón) y manipulación de iconos.

Para ejecutar cualquier programa solo basta con colocar el puntero del ratón sobre el icono que representa el programa y pulsar dos veces de forma rápida (como se dice comúnmente *doble-clic*) sobre el botón izquierdo del ratón.

Cuando se trabaja con Windows la información, programas ó datos, se guarda en archivos (files), los cuales son guardados en dispositivos de almacenamiento permanente (almacenamiento secundario). Un archivo es un conjunto de información relacionada a la cual tenemos acceso a través de su especificador. Todos los programas y datos contenidos en un disco residen en archivos y cada archivo tiene un especificador que consta de un nombre único y una extensión. La sintaxis de un especificador de archivo es la siguiente: **NomArch.Ext**

La extensión de un nombre de archivo indica qué tipo de archivo es. Así un archivo con extensión .cpp, seguramente será un archivo de código fuente de C++.

2.1. Observar el Contenido de un Disco

Una necesidad que se tendrá al trabajar con archivos es ver cuáles son los que se encuentran en un disco en particular. Para esta operación se debe hacer doble-click en el icono Mi PC que está en la pantalla del computador y cuando aparezca la ventana correspondiente se desplaza el puntero del ratón sobre el icono de la unidad de disco y se presiona dos veces (doble-click) el botón izquierdo del ratón, en ese momento aparece una ventana donde se puede observar el nombre de los archivos que contiene el disco al lado o debajo de un icono que, por lo general, indica el programa que generó dicho archivo. Uno de estos iconos puede corresponder a una carpeta, esto no es un archivo sino un conjunto de archivos denominado directorio.

2.2. ¿Que es un Directorio?

Un directorio es un archivo que contiene otros archivos y su información es de tipo directorio. Un archivo de tipo directorio contiene la información de los archivos que pertenecen a él.

Se puede reconocer un directorio porque el icono asociado es en forma de carpeta y si se hace doble-click en él aparece otra ventana mostrando los archivos que se encuentran dentro del mismo.

2.3. Copiando y Moviendo Archivos

Si se desea copiar un archivo de una unidad a otra o de un directorio a otro, basta con marcar tal archivo y "*arrastrarlo*" hasta la unidad o directorio de destino (el término "*arrastrar*" se denomina al movimiento del ratón manteniendo presionado el botón derecho y liberándolo sobre el icono perteneciente a la unidad o carpeta).

Si se desea mover el elemento, el proceso es casi igual, su única variante es que mientras se selecciona el archivo y se desplaza el ratón se debe mantener presionada la tecla ALT.

3. Ambiente Integrado de Desarrolllo de Builder C++

Cuando usted inicia Builder C++, quizás espera ver una sola ventana para desarrollar sus aplicaciones; pero Builder C++ le presenta un grupo de ventanas dispersas alrededor de su ventana principal. Al hacer doble click en el icono de Builder C++ aparece algo similar a lo que se muestra en la Fig 1.

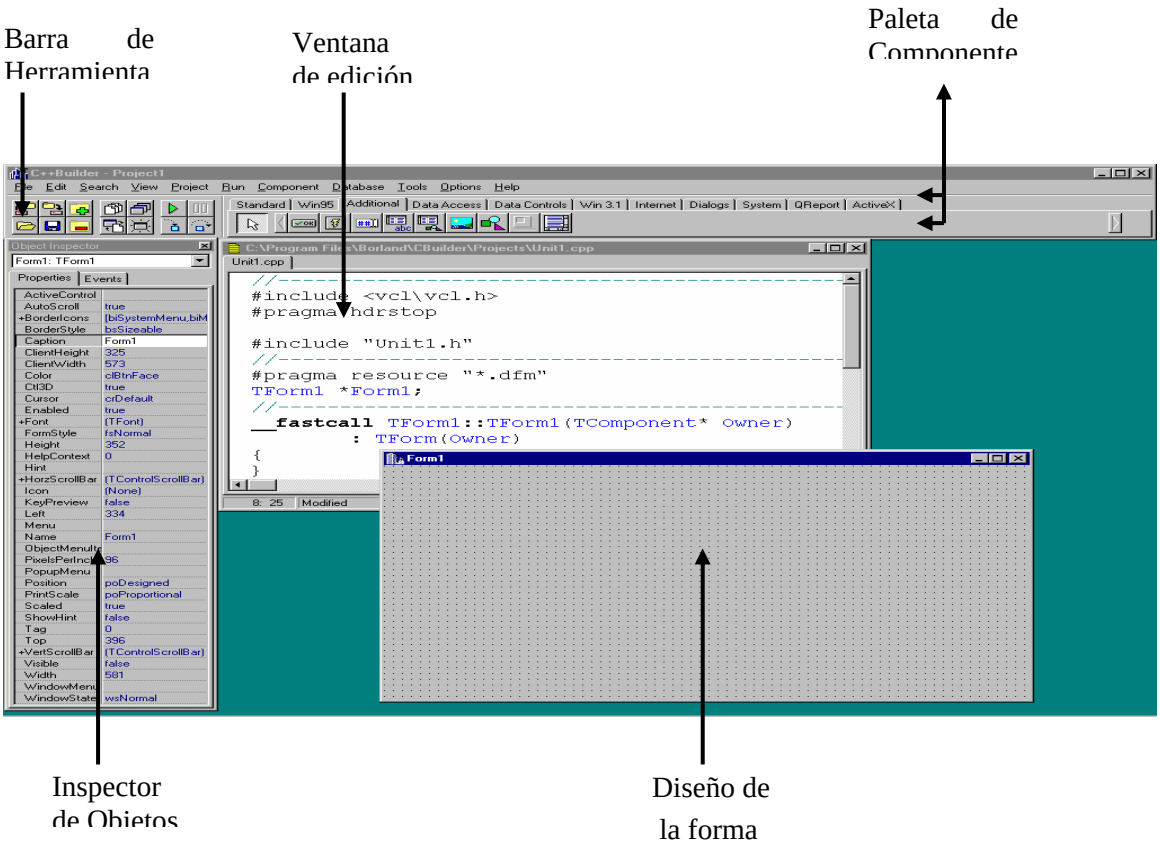


FIG.1- Ambiente de desarrollo integrado Builder C++

En la figura 1 se presenta los principales elementos de Builder C++ (Ambiente de Desarrollo Integrado, por sus siglas en Ingles IDE). Cada parte en el ambiente de desarrollo trabaja conjuntamente, diseños visuales y editor de código donde la edición es similar a otros editores; solo que con el ambiente de desarrollo integrado, usted puede observar realmente lo que esta construyendo al momento de crearlo.

3.1. Menú Principal y Barra de Herramientas

Muchas de las opciones que se pueden accesar desde el menú principal, están disponibles a través del panel de botones aceleradores en la barra de herramientas. En general la barra de herramientas provee una manera rápida de ejecutar operaciones del programa con un simple click con del ratón.

FIG. 2.1 - Menú principal

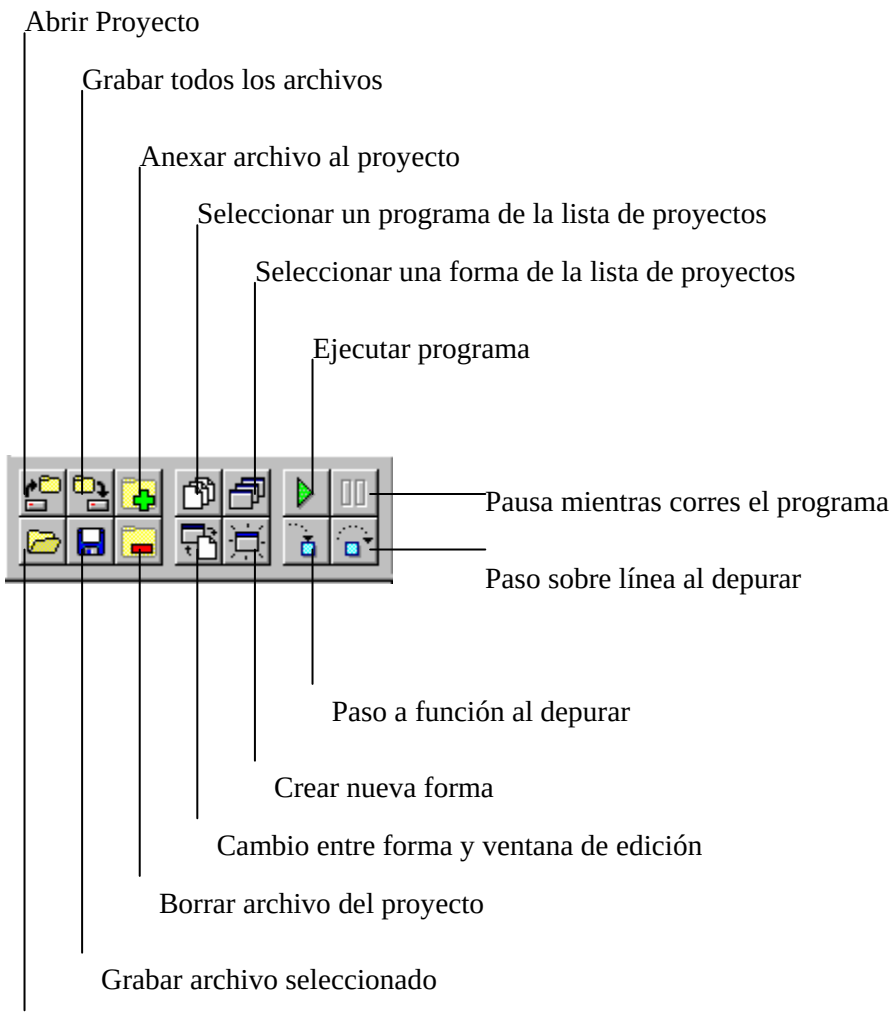


FIG. 2.2 - Barra de herramientas

Se puede configurar la barra de herramientas, seleccionándola con un click, y nuevamente haciendo click con el botón derecho del ratón, se activará un menú, y al elegir la opción de propiedades, tendrá a la disposición el editor de la barra de herramientas donde se podrá tomar las propiedades que se necesite para anexarlas a la barra de herramientas con solamente "*arrastrarla*" con el ratón al panel de la barra de herramientas.

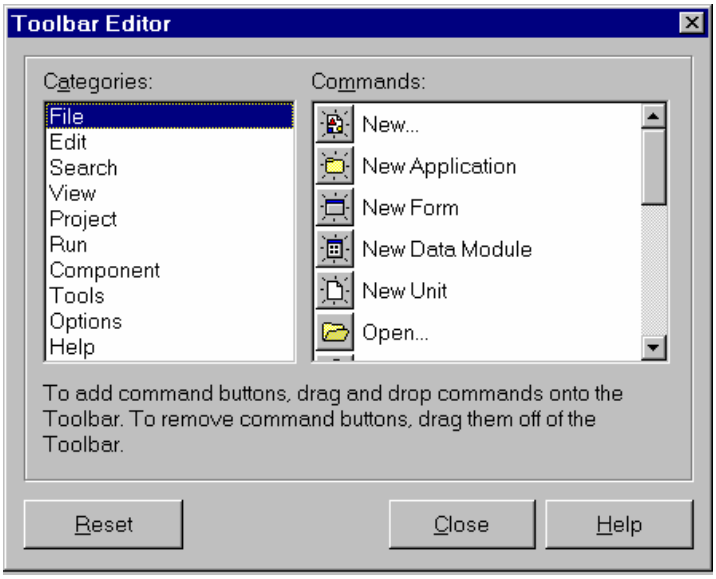


FIG. 3 - Editor de la barra de herramientas.

Si lo que se desea es remover alguna propiedad de la barra de herramientas, solamente se "arrastra" con el ratón fuera del panel de la barra de herramientas y desaparecerá.

3.2. Paleta de Componentes

La paleta de componentes que aparece en la Fig. 4, es algo como un catálogo de objetos que se puede usar de acuerdo a las necesidades de construcción de las aplicaciones. Está dividida en páginas o grupos de acuerdo a sus funciones. Para implantar uno de estos componentes en la aplicación, solo basta con seleccionarlo con el ratón haciendo un click en el objeto deseado y hacer click en la forma principal (Forma de edición, ventana punteada). Builder C++ soporta docenas de componentes.

FIG.4 - Paleta de componentes.

Standard: Esta tabla contiene los objetos para hacer eficaces y elegantes las aplicaciones Windows, incluye componentes para desplegar y editar texto, botones, barras de estado y menús.

FIG.5 - Paleta de componentes Standard.

Win95: Estos componentes permiten el acceso a los controles de usuario-interface de Windows95. Uno de los principales es la vista del árbol de directorio, (conocido como windows explorer), control de página, etc.

FIG.6 - Paleta de componentes Win95.

Additional: La tabla de adicionales contiene algunos de los mejores y variados de la paleta de componentes, como mapas de bits, botones aceleradores y componentes de apariencia.

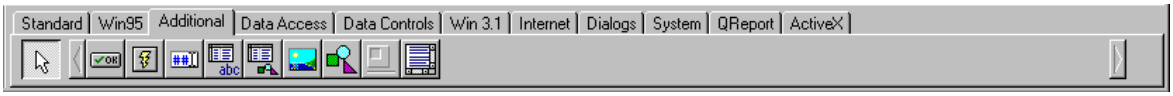


FIG.7 - Paleta de componentes Additional.

Data Access y Data Controls: Se pueden accesar bases de datos y hacer consultas dentro de las aplicaciones que se construyan con las facilidades que permiten estos 2 grupos de objetos.

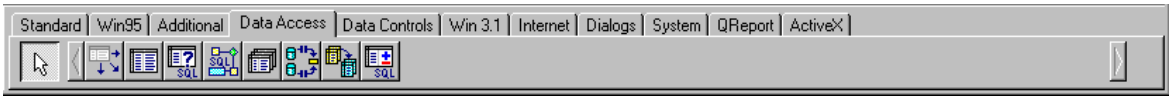


FIG.8 - Paleta de componentes Data Access.

FIG. 9 - Paleta de componentes Data Controls.

Win31: Muchos de los controles en Win31 tienen equivalentes en Win95 pero estas pueden usarse para dar sentido a aplicaciones para windows V. 3.1 además de proporcionar un block de notas.

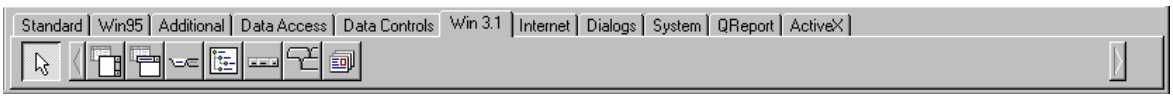


FIG.10 - Paleta de componentes Win 3.1.

Internet: Esta tabla dada por Builder C++, comprende lo referente al grupo de herramientas de internet.

FIG.11 - Paleta de componentes Internet.

Dialogs: Permite hacer cajas de dialogo que agilizan el desarrollo de las aplicaciones como abrir y grabar archivos, seleccionar tipos de letras, colores e impresoras y mucho más.

FIG. 12 - Paleta de componentes Dialogs.

System: Proporciona controles individuales para seleccionar archivos, directorios o drives.

FIG.13 - Paleta de componentes System.

Qreport: (o Quick Reports) provee de componentes para que pueda fácilmente organizar sus reportes y presenta la facilidad de una vista preliminar.

FIG. 14 - Paleta de componentes Qreport.

ActiveX: Esta tabla de componentes, contiene un revisor de ortografía así como objetos gráficos impresionantes.

FIG.15 - Paleta de componentes ActiveX.

Durante el desarrollo de este curso, solo veremos el uso de algunos de los objetos de la tabla de componentes, pero la idea es la misma para todos los objetos de esta paleta.

3.3. El Editor de la Forma

Cuando se comienza a trabajar con Builder C++, el espacio de trabajo y resultados se suple por la forma principal.

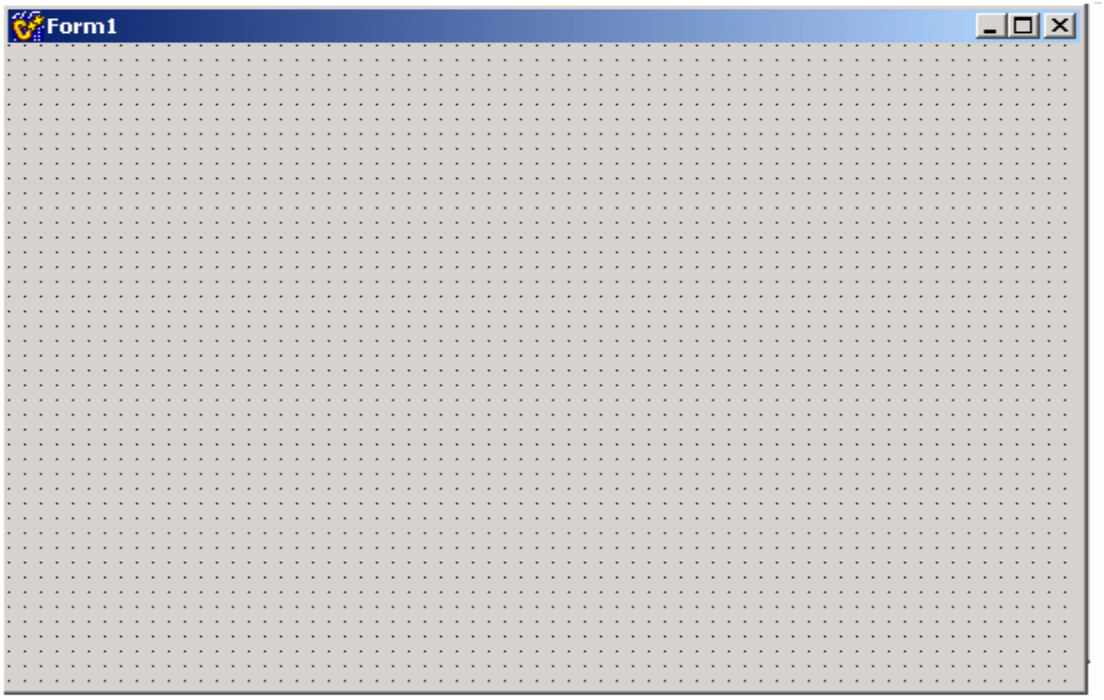


FIG. 16 - Editor de la forma.

Cada forma representa una ventana individual en la aplicación; en la forma se puede diseñar, añadir, eliminar reconfigurar los componentes según las necesidades de la aplicación.

3.4. El Inspector de Objetos

El inspector de objetos permite ver las propiedades o características de los objetos que comprendan el proyecto, por medio de él se pueden cambiar las propiedades de los objetos, también muestra los eventos asociados a los objetos de la aplicación.

Cuando se selecciona un objeto, el inspector de objetos automáticamente cambia al contenido y propiedades de este objeto. Si se oculta, o pierdes el inspector de objetos, lo puedes llamar oprimiendo la tecla de función F11.

Propiedades: Cuando se comienza un proyecto el inspector de objetos despliega las propiedades de la forma principal como son: nombre, color, altura, ancho, posición etc. Recordemos que al seleccionar otro objeto, automáticamente mostrará las propiedades de ese objeto.

Eventos: La tabla de eventos despliega para cada objeto los eventos como son: al activar el objeto, al oprimir una tecla, al oprimir el ratón, al soltar el ratón, etc. esos eventos son disparados con acciones del usuario, o del sistema operativo mismo. Por ejemplo el evento; *al hacer click en el ratón*, se dispara o hace una acción cuando el usuario hace click con el ratón para ese objeto.

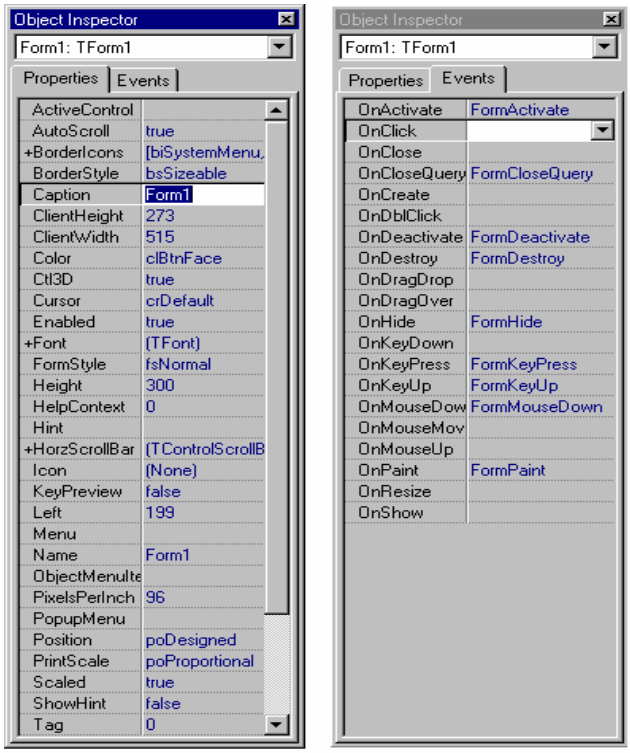


FIG.16 - Inspector de objetos.

3.5. El Editor de Código.

La ventana de edición de código muestra el código actual de la aplicación Builder C++. Al añadir objetos y hacer doble click sobre ellos, automáticamente se editará en la ventana de edición la llamada a la función que asociará al evento de ese objeto, dejando el espacio en blanco para que se codifique la acción que se desee para ese evento.

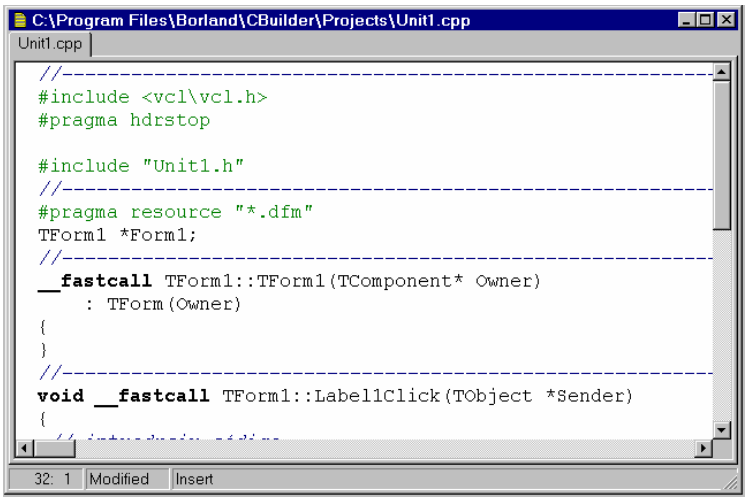


FIG. 17 - Ventana de edición de código.

3.6. El Manejador de Proyectos

Un sencillo proyecto de Builder C++ está conformado por solo una forma y su código, pero en aplicaciones muy grandes, puede conformarse un proyecto por varias formas, código y varios archivos de cabecera distintos a las librerías que por omisión ya necesita la aplicación, por tanto un proyecto puede integrar varios archivos, para saber cuales son los archivos que comprende un proyecto, usaremos el manejador del proyecto, que muestra el árbol de archivos involucrados en el orden en que fueron añadidos.

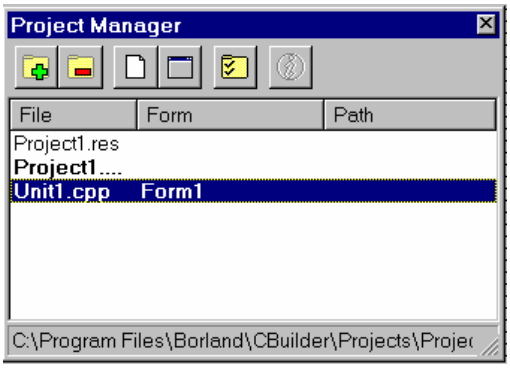


FIG. 18 - Manejador de proyectos.

3.6.1. Como Añadir Archivos a un Proyecto.

Para añadir archivos a un proyecto, se selecciona del menú principal: View ⇒ Proyect Manager ⇒ y al hacer doble click se obtiene una caja con la información de los componentes actuales del proyecto (FIG. 19), al hacer click en el botón de integración al proyecto,  , saldrá una ventana de adición “Add to proyect”, donde se puede buscar la ruta de acceso donde se encuentre el archivo a añadir, y al hacer click en el botón Ok para cerrar la caja de dialogo se obtendrá ya añadido al proyecto el archivo seleccionado.

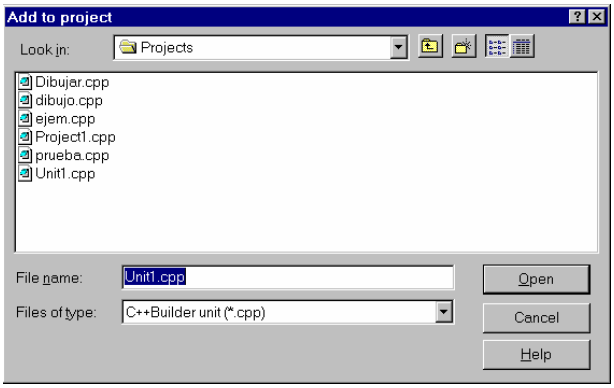


FIG. 19 - Ventana de adición al proyecto.

3.6.2. Como Eliminar Archivos de un Proyecto.

Para eliminar un archivo del proyecto, se abre el manejador de proyectos, se selecciona el archivo que se quiere eliminar y se hace click en el botón de remover archivo  y será borrado de la lista. (no de su máquina, solo del proyecto).

3.6.3. Opciones del Proyecto.

Se puede accesar a detalles del ambiente de configuración del proyecto actual eligiendo del menú principal Options ⇒ Proyect. Este comando abre la caja de dialogo de opciones del proyecto, toma el ambiente para cada proyecto y puede ser accesado en cualquier momento durante el desarrollo de la aplicación.

Puede explorar las 6 hojas para ver las opciones que están disponibles. Al iniciar un proyecto, comienza con la configuración que tiene por omisión.

3.7. Pagina de Formas.

Por omisión está incluida la auto creación de una forma, al abrir un nuevo proyecto, evitando que al crear una nueva aplicación se tenga que pedir una forma. Al crear un nuevo proyecto se crea automáticamente su forma principal y su ventana de edición de código.

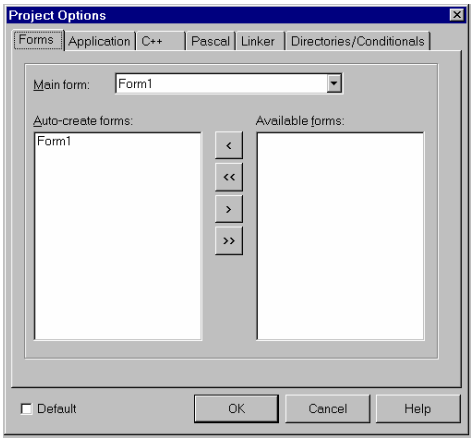


FIG. 20 - Opciones del proyecto, página de Formas.

3.8. La Pagina de Aplicaciones.

La página de aplicaciones contiene 3 mascaras de configuración:

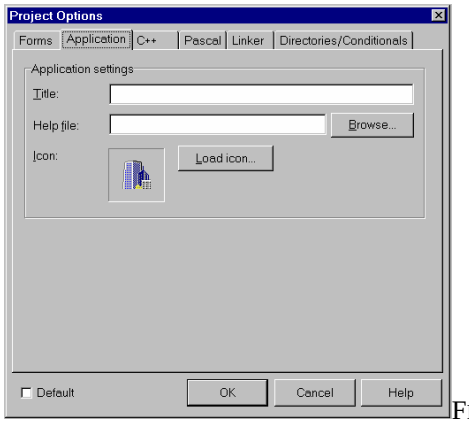


FIG. 21 - Opciones del proyecto, página de Aplicaciones.

Titulo: El texto que se introduzca será el titulo de la aplicación y será desplegado con el icono cuando se minimice la aplicación.

Help_File: Asociará un archivo de ayuda a su aplicación.

Icon: El archivo ejecutable contendrá el icono que se seleccione, por omisión mostrará el de aplicaciones Builder C++.

3.9. Archivos Fuente Generados por Builder C++.

Cada una de las siguientes extensiones son vitales para cualquier proyecto de Builder C++. Los siguientes archivos contienen detalles del diseño de los proyectos y formas, se debe tener cuidado en no perder ninguno ya que se pueden utilizar si se quisiera modificar alguna aplicación hecha.

Project1.mak: Este es el archivo principal de opciones del proyecto. Un archivo .mak se requiere en cada aplicación; es un archivo de texto que puede examinar eligiendo del menú principal: View ⇒ Project MakeFile. Este archivo contiene instrucciones de cómo Builder C++ construirá el archivo (.exe) ejecutable para el proyecto.

Project1.Cpp: Este archivo contiene el código principal de la aplicación, comparte el mismo nombre del proyecto, lo crea automáticamente Builder C++ al darle nombre al proyecto; contiene el código de iniciación y terminación del programa. si se quiere examinar este archivo, solo se elije del menú principal View ⇒ Project Source.

Unit1.cpp: Este es el código que se le da a los eventos de los objetos que tendrá en su forma final, este código es el que se introduce en la ventana de edición, o en el editor de código.

Unit1.h: Para cada archivo .cpp, Builder C++ crea automáticamente un .h correspondiente. El archivo de cabecera contiene la declaración de la forma y menciona a Builder C++ la lista de componentes y los eventos que tendrá la aplicación.

Unit1.dfm: El archivo .dfm contiene la información, definición y declaración de la forma y otros detalles importantes como: tamaño, color, títulos, fondos etc. así como detalles del resto de los componentes utilizados en la forma. La extensión .dfm indica que este archivo oculta los datos de la forma en formato binario. Este archivo no se puede leer, pero se puede convertir para observar su contenido, solo se selecciona la forma, con el botón derecho del ratón, se hace click y saldrá un menú de opciones, se escoge View as text, para ver la información de su forma y de igual manera, para regresar a verlo como forma, seleccione con el botón derecho del mouse, escoja View as Form, y regresará a su forma.

Unit1.obj Cuando se compila el proyecto se crea el archivo binario con extensión .obj. Cada aplicación contendrá un archivo .obj, este archivo cambiará cada vez que se reconstruya el proyecto.

Project1.exe o Project1.dll: Este es el archivo final según se halla elegido en el proyecto, el .exe podrá ejecutarse, y los dll son librerías dinámicas que pueden utilizarse desde otros programas.

Project1.dsk: Contiene la información de la configuración que tenía el hardware al momento de crear o finalizar el proyecto, para que al momento de ejecutarse lo haga sobre la misma configuración.

Project1.il?: La extensión .il? indica que el archivo es usado al enlazar, Builder C++ usa una tecnología de compilación muy rápida, lo que hace que la compilación o recompilación de sus proyectos sea en cuestión de segundos.

Inicios - Comunicándose con el Usuario

1. Objetivos

- En Borland C++

  Conocer y emplear adecuadamente el concepto de variable conocer los tipos básicos de variables y como declararlas para poder emplearlas, la asignación y las funciones de biblioteca que ofrece Borland C++ para manipular datos alfanuméricos.

- En el Ambiente *Builder*

  Conocer los elementos básicos del ambiente de trabajo en *Builder*, para poder crear aplicaciones. Utilizar los objetos **Label**, **Edit** y **Button**, cambiando sus propiedades y empezar a asignar controladores de eventos a los objetos.

2. Introducción

Ya debe saber que cada vez que trabaja con un computador todos los datos, resultados parciales y resultados finales pasan por la memoria principal, es posible que en varias fases estos valores se ubican en algún dispositivo auxiliar de almacenamiento, pero lo que es seguro es que los mismos siempre pasan a memoria principal para su procesamiento.

Para facilitar el entendimiento de esta fase, se puede imaginar la memoria como un casillero donde cada casilla representa una localidad de memoria, considerando que en cada una de ellas sólo se puede almacenar un valor a la vez. Cada una de estas casillas se identifica con un nombre y se le dan unas características (tipo de dato a alojar) en lo que se llama declaración de variables, por ende cuando se trabaja con valores en un programa se trabaja con las variables.

Para almacenar un valor en estas casillas se realiza una operación que se llama asignación y que se define como el hecho de dar valor a una variable. La operación anterior se puede hacer de dos formas:

- Una es interna al algoritmo y se denomina sentencia de asignación.
- Otra desde el exterior (llamada lectura de datos) que consiste en el ingreso de valores desde el exterior hacia la memoria, donde la forma más común es empleando el teclado.

Eventualmente, una vez que se hayan procesado los datos calculando lo que sea necesario, será imprescindible mostrar los resultados al usuario (inicialmente sólo se trabajará con números o letras), en este caso el valor almacenado en la variable es mostrado empleando alguno de los componentes que se dispone para ello.

> ☞Es un error común creer que cuando se muestra el valor de una variable, la misma se vacía. Realmente se muestra una copia del contenido de la misma y hasta que se indique lo contrario el valor se mantendrá siempre en la variable.

A las primeras de cambio, para los alcances de este curso se trabajará con dos grupos básicos de tipos de datos: los valores numéricos y los llamados valores alfanuméricos.

Para poder emplear una variable es necesario declarar la misma, lo que se hace con esta acción es reservar una posición de la memoria principal bajo un identificador al cual se referirá en el desarrollo del programa e indicar de que tipo es, lo que implica definir el tipo de dato que puede ser alojado en tal posición de memoria.

Lo anterior nos lleva a conocer los principales tipos de datos que tendremos disponibles para declarar las variables:

Tipo de Dato	Rango de posibles valores	Comentario
CHAR	-128 a 127	Datos sólo enteros
UNSIGNED CHAR	0 a 255	Datos sólo enteros positivos
INT	-32768 a 32767	Datos solo enteros
UNSIGNED INT	0 a 65535	Datos sólo enteros positivos
LONG	-2147483648 a 2147483647	Datos sólo enteros
UNSIGNED LONG	0 a 4294967295	Datos sólo enteros positivos
FLOAT	5x10-324 a 1.7x10308	Datos de punto flotante
CHAR	Alfanumérico corto	Hasta 256 caracteres
ANSISTRING	Alfanumérico Largo	

TABLA 1

La declaración de variable tiene la siguiente sintaxis:

TIPO *IDENTIFICADOR DE LA VARIABLE* **;**

Como ejemplos válidos de declaraciones se tienen:

FLOAT X, Y, Z; **INT** A, B; **CHAR** Letra; **ANSISTRING** A;

Como se puede observar es posible realizar declaraciones consecutivas separadas por coma. En Builder también existe un tipo de variable llamada booleana la cual toma sólo dos valores: verdadero o falso.

La asignación interna denominada comúnmente sentencia de asignación se rige bajo el siguiente esquema:

IDENTIFICADOR DE LA VARIABLE = Expresión**;**

En Borland C++ el símbolo de la asignación es = . En una asignación primero se evalúa la expresión y finalmente el valor resultante se almacena en la variable y al final se escribe el carácter punto y coma.

En la formación de expresiones numéricas aparecen operandos (constantes, variables y funciones) agrupados mediante operadores, en este caso los llamados aritméticos.

Op.	Denominación	Operandos	Resultado
+	Suma	Entero, Real	Entero, Real
-	Resta	Entero, Real	Entero, Real
*	Multiplicación	Entero, Real	Entero, Real
/	División	Entero, Real	Entero, Real
%	Resto de la división entera	Entero	Entero
++	Incremento	Entero	Entero
--	Decremento	Entero	Entero

TABLA 2

PRIMEROS ELEMENTOS DE ENTRADA / SALIDA

A continuación se presentarán los primeros componentes gráficos que pueden ser empleados para solicitarle datos al usuario o presentarle resultados, se verá que los mismos poseen una

serie de características llamadas propiedades que pueden ser cambiadas según lo que se desee, esta guía sólo muestra las propiedades más relevantes y queda de parte del estudiante investigar las restantes.

Aquí es donde se comienza a notar el cambio entre los programas desarrollados en ambiente texto y los que trabajan en un entorno gráfico ya que en este último ambiente se incrementa de manera notable las formas con las que el usuario puede comunicarse con el programa, aunque en el fondo, el proceso subyacente es el mismo.

- **COMPONENTE LABEL**

  Es uno de los componentes primarios para comunicarse con el usuario al permitir presentarle etiquetas donde se escribe lo que se desea que sea leído por el usuario en la pantalla. Este componente se encuentra en la pestaña *Standard* de la barra de componentes.

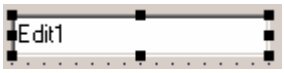
  Es un elemento muy sencillo donde su propiedad más empleada es ***Caption***, cadena alfanumérica la cual en definitiva contiene el mensaje ofrecido al usuario

  Otras propiedades interesantes de este componente son: ***Autosize*** de valor *boolean* que permite ampliar el ancho del espacio para mostrar el mensaje, ***WordWrap*** también de valor *boolean* que permite que el escrito crezca en forma vertical al permitir nuevas líneas y finalmente los valores asociados a su aspecto englobados con la propiedad **Font**.

  Estas propiedades pueden ser modificadas en tiempo de diseño o en tiempo de ejecución al igual que las otras de este componente y de otros componentes.

  Un hecho importante con este componente es que no sólo sirve para presentar mensajes, sino que también es posible ofrecer resultados numéricos, lo único que se debe tomar en cuenta que el número debe ser previamente convertido a su equivalente alfanumérico.

- **COMPONENTE EDIT**

  Este componente se encuentra en la pestaña *Standard* de la barra de componentes.

  Con este elemento se le da la oportunidad al usuario de ingresar datos al programa mediante una caja de edición en la cual puede escribir. Al igual que con el componente **Label**, este elemento trabaja con una cadena alfanumérica, por lo que si es ingresada una cadena que representa valor numérico, ésta debe ser convertida a su equivalente numérico. Aunque no es común, también se le puede emplear para mostrar resultados.

  Su propiedad más empleada es ***Text***, cadena alfanumérica la cual, en definitiva, contiene el valor escrito por el usuario. Al igual que en el componente **Label**, se puede modificar las características de la letra empleada con los aspectos englobados en la propiedad ***Font***.

- **CAJAS DE DIÁLOGO**

  El *Builder* ofrece un grupo de 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 son en el estilo como la mostrada en la figura y sirven tanto para dar un simple aviso como para solicitar confirmación o no de una acción.

  La más sencilla de ellas se invoca empleando una simple sentencia de llamado de procedimiento:

  Es **ShowMessage**: produce una caja con un mensaje y un único botón de aceptación.

  ShowMessage('Mensaje que aparece');

COMPONENTE BUTTON

Este componente se encuentra en la pestaña *Standard* de la barra de componentes.

Es el botón rectangular de amplia difusión en las aplicaciones gráficas, permite que se le escriba un mensaje en su superficie mediante la propiedad ***Caption***, y es una forma de solicitar al usuario que indique el comienzo de un proceso.

Modificación de propiedades de componentes

Como puede observar existen varias propiedades en los diferentes componentes que significan lo mismo. Estas propiedades pueden ser modificadas en lo que se denomina *Tiempo de Diseño*, cuando mediante toque del cursor sobre el elemento modifica una propiedad en particular en el recuadro (que usualmente se ubica a la izquierda de la pantalla) llamado *Object Inspector*.

La otra posibilidad es que mientras está funcionando el programa se modifiquen estas propiedades, esto se llama *Tiempo de Ejecución* y de hecho existen propiedades en los elementos que sólo pueden ser modificadas en estos momentos.

Para modificar una propiedad en tiempo de ejecución se realiza una sentencia similar a la sentencia de asignación, de acuerdo con el siguiente esquema:

COMPONENTE->PROPIEDAD = nuevo valor*;*

Por ejemplo: **Button1->Caption= "Concluir";**

Lo anterior colocaría la palabra *Concluir* sobre la superficie de un botón.

Es importante resaltar que el nombre del componente debe estar separado de su propiedad mediante un guión seguido del símbolo mayor.

Como puede observar las propiedades que se emplean en los componentes explicados son valores alfanuméricos y eventualmente existirá la necesidad de presentar valores numéricos, esta asignación no pede ser realizada directamente, sino que hay que "traducir" un valor al otro o viceversa, para lo anterior dispone de una serie de funciones de conversión, algunas son:

FUNCIÓN	PARÁMETRO	RESULTADO
ATOI(VALOR)	Alfanumérico Corto	Entero
GCVT, ECVT(VALOR)	Real	Alfanumérico Corto
ATOF(VALOR)	Alfanumérico Corto	Real
VALOR.C_STR()	Alfanumérico Largo	Alfanumérico Corto
VALOR.TODOUBLE()	Alfanumérico Largo	REAL
VALOR.TOINT()	Alfanumérico Largo	REAL
ANSISTRING(VALOR)	**N**umérico	Alfanumérico Largo

TABLA 3

Por ejemplo si se quiere convertir un valor escrito por el usuario en una caja de texto (que aunque parezca un número es una cadena alfanumérica) a una variable numérica del tipo real, la sentencia a ser escrita será:

IDENTIFICADOR DE LA VARIABLE = EDIT1->TEXT.**TODOUBLE()**;

Algo que hay que resaltar que si el usuario escribe mal el número (por ej. Coloca una letra) la ejecución de esta sentencia dará un error y se detendrá la ejecución del programa (este es un ejemplo de los llamados errores de ejecución):

Asignación de Controladores de Evento

De los componentes que el programador decida colocar al alcance del usuario, algunos interactuarán con el mismo, por ejemplo: cuando haga *click* sobre este, cuando presione una tecla, etc. Ante esa acción o evento el programa realizará "algo", ese "algo" son instrucciones que son conocidas como control del evento asociado a un componente.

Para poder escribir las sentencias asociadas al evento sobre un objeto lo que debe hacer es posicionar el cursor sobre el objeto, en tiempo de diseño, y hacer *doble click* sobre el mismo; verá como la pantalla cambia a la zona de escritura de código que ya posee elementos escritos previamente por el ambiente de trabajo y que el cursor estará colocado entre unas líneas en particular que corresponden al procedimiento que será ejecutado cuando suceda un evento (si bien es cierto que pueden suceder varios eventos sobre un objeto, por ahora no se preocupe, empleará el evento que por defecto se le asigna a ese objeto, usualmente el más común).

3. Actividades de Laboratorio

Realizará un programa que dadas las coordenadas cartesianas (x, y) de un punto en el plano, determine sus coordenadas polares. Para esto debe:

- Crear una forma en la cual insertará, al menos, los siguientes componentes:
- Cuatro componentes **Label** que serán rotulados para pedir los datos y presentar los resultados, dos componentes **Edit** para solicitar los valores de x y y, un componente **Botón**.
- El componente **Botón** se etiquetará como “Calcular” y asociado al evento *click* programar un proceso que calcule lo solicitado.

Este es un problema muy simple pero permitirá que conozca los elementos fundamentales de desarrollo de programa en el ambiente *Builder*, preste atención a la explicación del profesor preste atención a los detalles operativos de trabajar con un elemento visual.

4. Problemas Propuestos

1) Diseñar un programa que dadas las coordenadas cartesianas (x, y, z) de un punto en el espacio, determine sus coordenadas cilíndricas y sus coordenadas esféricas.
2) Dados dos puntos en el plano por sus coordenadas (x_1, y_1) y (x_2, y_2), diseñe un programa para calcular la ordenada correspondiente a un abscisa x cualquiera empleando interpolación lineal..
3) Dado un triángulo cualquiera por las longitudes de sus tres lados, escriba un programa que calcule el área del triángulo.
4) Escriba un programa que dada la hora en un reloj analógico (con agujas), indique que ángulo se forma entre el minutero y el segundero. Asuma que este reloj no posee la manecilla de los segundos.

Tomando Decisiones

1. Objetivos

- En Borland C

  Conocer y emplear adecuadamente las sentencias de toma de decisiones **if – then** y **switch**. Los operadores relacionales y lógicos y las variables del tipo Lógicos.

- En el Ambiente Builder

  Utilizar los objetos **CheckBox**, **RadioButton** y **GroupBox.**

2. Introducción

Como en la mayoría de las actividades del ingeniero, cuando se realiza un programa es muy común encontrarse con la necesidad de tomar decisiones entre realizar un grupo de instrucciones u otro, dependiendo del cumplimiento o no de una o varias condiciones.

El Borland C ofrece dos sentencias compuestas para tomar decisiones: una que representa los pasos de decisión de una y/o doble alternativa que se logra con la sentencia **if** y que es de un uso muy común. La otra es la sentencia **switch**, la cual nos permite discriminar entre diferentes posibilidades de una vez pudiendo de esta forma sustituir un conjunto de sentencias **if** anidadas.

Para emplear estas sentencias debemos igualmente manejar con fluidez los conceptos asociados a las expresiones lógicas (aquellas que resultan en los valores **True** y **False** ó **Uno** y **Cero** y que pertenecen a un tipo de datos denominado LÓGICOS ó BOOLEAN). Las expresiones lógicas son formadas con los siguientes elementos: Operadores relacionales, constantes, variables, funciones y operadores lógicos.

3. Sentencia if –else

La sintaxis de una sentencia **if** es la siguiente:

```
:
if (Expresión)
    Sentencia 1
else
    Sentencia 2
:
Resto de las sentencias
```

La parte **else** puede ser obviada de considerarse pertinente.

Si el resultado de *Expresión* es **TRUE** ó distinto de **CERO**, la sentencia que viene inmediatamente después será ejecutada (*Sentencia1)* y luego el flujo del programa irá hacia el resto de las sentencias saltando la *Sentencia2* asociada al **else**.

Pero si el resultado de *Expresión* es **FALSE** ó igual a **CERO** y la parte **else** está presente, será ejecutada *Sentencia2*. En caso de no encontrarse presente la parte **else** no será ejecutada la *Sentencia1*. En ambos casos el flujo del programa continuará con el resto de las sentencias.

En general, un **else** es asociado con el **if** más cercano que ya no se encuentre asociado con otro **else**. No puede existir un punto y coma inmediatamente después de la (*Expresión)*, ya que este punto y coma sería interpretado por el compilador como final de la estructura **if**.

En esta sentencia empieza a ser relevante el uso de la sentencia compuesta por medio de las llaves **{ }**.

El uso de la sentencia **if - else** lleva a la necesidad de saber escribir correctamente *Expresión* cuyo resultado se indicó como es del tipo lógico.

Una expresión es la forma en que pueden agruparse operadores y operandos para obtener un resultado. En el caso de la sentencia **if**, se emplean operadores relacionales que son operadores binarios, es decir, actúan sobre dos valores, y se introducen para efectuar comparaciones entre dos valores del mismo tipo y dan como resultado un valor lógico, es decir, cierto (**TRUE**) o distinto de cero ó falso (**FALSE**) o igual a cero.

Los *operadores relacionales* son:

==	Igual a	>	Mayor que
! =	Distinto a	<=	Menor o igual que
<	Menor que	>=	Mayor o igual que

TABLA 1

Los *Operadores lógicos* sólo trabajan con valores lógicos y estos son:

!: usado para indicar la negación lógica. Aplicado a **TRUE** produce **FALSE** y aplicado a **FALSE** produce **TRUE**.

&&: es el operador lógico **Y**, y actúa sobre dos valores lógicos. El resultado de la evaluación es cierto si los dos valores toman simultáneamente el valor cierto, en otro caso el resultado es el valor falso.

||: es el operador lógico **O** y también requiere dos operadores lógicos. Proporciona el valor verdadero si al menos uno de los dos valores es cierto.

Tanto los operadores lógicos como los relacionales también pueden ser utilizados en una expresión de una instrucción de asignación, la variable recibirá el valor **CERO** si el resultado es **FALSE** ó **UNO** si el resultado es **TRUE**, por ejemplo:

```
A = (5 < Z) || ( X > Y);
```

4. Sentencia switch

El mecanismo de selección múltiple de Borland C está asentado en la sentencia **switch**, que se limita a verificar la ocurrencia de un valor (ordinal) en una expresión de control y a transferir la secuencia de sentencias según ésta ocurrencia, teniendo la siguiente sintaxis:

```
switch (Expresión Ordinal) {
    case Valor 1 : Bloque de Sentencias 1; break;
    case Valor 2 : Bloque de Sentencias 2; break;
    :
    case Valor N : Bloque de Sentencias N; break;
    default      : Bloque de Sentencias N+1;
}
```

Aquí las listas de valores pueden ser valores ó incluso expresiones constantes, compatibles en tipo con la expresión de control – conocido como selector – sin que importe el orden de aparición. Similar al **else** la cláusula **default** es opcional y se emplea al existir una alternativa diferente a todas las listas de valores.

La sentencia **break** se debe colocar al final de cada bloque e indica el final del Bloque de sentencias. No es obligatorio colocarla al final del último bloque de sentencias.

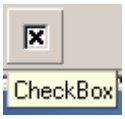
Un ejemplo de esta sentencia es el siguiente:

```
switch (Opcion){
    case 1 : Bloque de Sentencias 1; break;
    case 3 : Bloque de Sentencias 2; break;
    case 4 : Bloque de Sentencias 3; break;
}
```

Es de resaltar que cada sentencia finaliza en punto y coma.

5. Componentes visuales de Builder C

- **Componente CheckBox**

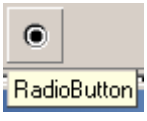


Este componente se encuentra en la pestaña *Standard* de la barra de componentes.

Permite colocar cajas de marcaje, las mismas tienen asociadas una etiqueta que permite dar una breve explicación del elemento seleccionado. Se pueden colocar todos los que se consideren necesarios y es muy útil cuando se desea que el usuario elija uno o más elementos de una lista de opciones.

Sus principales propiedades son: ***Caption***: que permite escribir el mensaje deseado en la etiqueta ubicada al lado de cada caja y ***State*** que indica el estado de la caja de selección y ***Checked*** que indica si la caja está marcada o no.

- **Componente RadioButton**



Este componente se encuentra en la pestaña *Standard* de la barra de componentes.

Es similar al anterior elemento, físicamente es un círculo en vez de un cuadrado, pero la principal diferencia radica en que de un grupo sólo un elemento puede estar seleccionado, mientras que con **CheckBox** pueden estar varios seleccionados de forma simultánea, su uso, por lo tanto, es cuando se quiere que el usuario elija una opción única de entre varias propuestas.

También posee la propiedad ***Caption*** que permite escribir un mensaje al lado del círculo y la propiedad ***Checked*** que indica si la caja está marcada o no, este último elemento posee valor **BOOLEAN** (**TRUE** si el círculo está marcado y **FALSE** en caso contrario).

- **Componente GroupBox**

Este componente se encuentra en la pestaña *Standard* de la barra de componentes.

A parte de poder usar este componente para presentar en la forma los diferentes componentes en una manera "más ordenada", su principal utilidad reside en poder separar grupos de componentes **RadioButton**. Para entender un poco mejor debe recordar que sólo un **RadioButton** puede estar marcado en un grupo de ellos, pero si necesita dos o más grupos de estos elementos (digamos: para diferenciar el estado civil y laboral de una persona) empleará componentes **GroupBox**.

6. Funciones Numéricas

Función	Tipo	Nombre	Comentario
SQRT(x)	Float	Raíz Cuadrada	Saca la raíz cuadra, x puede ser entero
POW(x,y)	Real	Potencia	Eleva x a una potencia y de tipo real

TABLA 1

7. Actividades de Laboratorio

- **Primera Actividad**

Va a elaborar un programa que calcule las raíces de una ecuación de segundo grado del tipo $Ax^2 + Bx + C = 0$ empleando la fórmula general y cumpliendo con la exigencias que se indican a continuación:

- Cree una forma en la cual desplegará los componentes necesarios:
- Coloque sobre el formulario las etiquetas y cajas de edición necesarias para solicitar los datos A, B y C.
- Coloque también un componente **CheckBox** para que el usuario indique si desea que sean calculadas raíces imaginarias o no.
- Inserte un botón el cual etiquetará como "Calcular" y asociado al evento *click* programe un proceso que calcule las raíces de la ecuación tomando en cuenta lo indicado en el **CheckBox**.

  Es obligación de un programador que no sucedan errores de ejecución, así que debe evitar la ocurrencia de divisiones por cero o intentar sacar raíces cuadradas de valores negativos.

El proceso básico de solución de este problema debe ser discutido con el profesor del laboratorio.

- **Segunda Actividad**

Aquí diseñará un programa que pida al usuario una distancia en metros y dependiendo de la selección del usuario calculará su equivalente en una de varias medidas hispanoamericanas poco comunes, que son: Almud (0,27 m), Cana (1,541 m), Jarocha (4,19 m), Palmo (0,212 m) o Estadal (3,391 m), para esto:

- Cree una forma en la cual desplegará los componentes necesarios:
- Coloque un **RadioButton** por cada opción.

- Coloque una caja de edición con una respectiva etiqueta pidiendo al usuario la distancia en metros y una segunda etiqueta en donde colocará el resultado
- Inserte un botón el cual etiquetará como "Calcular" y asociado al evento *click* programe un proceso basado en la sentencia **if-then** para presentar el resultado.

El proceso básico de solución de este problema debe ser discutido con el profesor del laboratorio, recordando que no debe permitir que sucedan errores de ejecución.

8. Problemas Asociados

1) Una forma de determinar si un año es bisiesto es que el mismo sea divisible de manera exacta por cuatro, pero si el mismo es fin de siglo (secular) debe ser divisible por cuatrocientos. Realice un programa que basado en el criterio anterior determine si un año dado como dato es o no bisiesto, generando un mensaje adecuado.
2) Ork el planeta natal de Mork celebra fiestas planetarias cada ocho años, pero cada 48 años la celebración se suspende debido al penoso recuerdo que dejó la derrota con su planeta enemigo en las Guerras Impúdicas, de manera similar cada 72 años se celebra el cumpleaños de Orson y para hacerlo en grande se festeja también al año siguiente. Realice un programa para saber si un año determinado es o no festivo.
3) Realice un programa que dados cuatro valores A, B, C, D los presente ordenados de menor a mayor.
4) Diseñe un programa que indique si tres puntos dados por sus coordenadas forman triángulo y en caso de formarlo debe indicarse que tipo de triángulo forma.

Ciclos

1. Objetivos

- En Borland C

  Conocer y emplear adecuadamente las sentencias de repetición **do - while**, **while** y **for**
- En el Ambiente Builder

  Utilizar los objetos **ProgressBar, ScrollBar**, **TrackBar** y **UpDown.**

2. Introducción

En función del tipo de problema planteado y por ende su método de solución mediante el computador a veces es necesario repetir un grupo de instrucciones hasta que se cumpla una o varias condiciones. Este tipo de sentencia siempre presenta un encabezado, que indica el inicio del ciclo, y siempre se evaluará una condición mediante la cual se controlará la cantidad de repeticiones que dará el ciclo.

En ésta sesión se trabajará con las tres instrucciones compuestas de control que Borland C ofrece para la repetición de sentencias, también conocidas como ciclos iterativos, ellas son las sentencias **DO - WHILE**, **WHILE** y **FOR**.

En el uso de estas sentencias se debe prestar particular atención a su elaboración en conjunto con las sentencias que repetirá ya que es un error bastante común crear ciclos que nunca culminarán su ejecución quedando en lo que se llama un ciclo infinito o un *Loop* como también se le conoce, esto es un error de ejecución y su ocurrencia usualmente está asociada a no evaluar adecuadamente la condición de finalización.

2.1. Instrucción de repetición do - while

La sentencia compuesta **DO – WHILE** o bucle *do* ejecuta las sentencias comprendidas entre las palabras reservadas **DO** y **WHILE** hasta que la expresión de control sea verdadera (**TRUE**) ó distinta de cero. La sintaxis de esta sentencia es la siguiente:

do{

< *sentencia* > ;

< *sentencia* > ;

....

} while (*Expresión)*;

< *sentencia* > ;

El lazo posee, al lado de la palabra **WHILE** una *Expresión* que controla la repetición de la misma. *Expresión* debe producir un resultado del tipo lógico. Las sentencias escritas entre las palabras reservadas **DO** y **WHILE** serán ejecutadas en secuencia hasta el final del proceso mientras la evaluación de *Expresión* produzca el resultado **TRUE** Ó DISTINTO DE CERO. En el momento que al evaluar *Expresión* se obtenga el resultado **FALSE** Ó IGUAL A CERO, el flujo del programa continuará con la sentencia siguiente a **WHILE**.

La secuencia de sentencias ubicadas entre las palabras reservadas **DO** y **WHILE** será ejecutada al menos una vez, esto debido a que *Expresión* es evaluada al final de la ejecución de cada secuencia.

Un elemento que debe ser tomado en cuenta es que, aunque la condición pueda dejar de cumplirse en alguna sentencia dentro del ciclo, éste no se detiene hasta el final del mismo, donde es evaluado efectivamente en *Expresión*.

2.2. Sentencia de repetición WHILE

La sentencia **WHILE** indica a la computadora que se ejecute una sentencia, que puede ser compuesta, mientras se cumpla una determinada condición.

La condición viene determinada por una *Expresión* cuyo resultado debe ser del tipo lógico. El formato general que adopta la estructura es el siguiente:

WHILE (*Expresión)*

< *sentencia* > ;

La sentencia **WHILE** comprueba inicialmente si la *Expresión* es verdadera. Si *Expresión* es **TRUE** Ó DISTINTO DE CERO se ejecuta la sentencia, y lo seguirá haciendo mientras la condición sea verdadera; el ciclo finaliza en el momento que al evaluar *Expresión* resulte el valor **FALSE** Ó IGUAL A CERO, esto hace que el flujo del programa salte la *Sentencia* del bucle, continuando la ejecución del programa en la siguiente sentencia. Dado que *Expresión* puede llegar a ser falsa inicialmente, es decir, antes de comenzar el bucle; habrá casos en que el ciclo no se ejecute.

También debe tener muy en cuenta que aunque en el desarrollo de *Sentencia* se cambien los valores que hagan que *Expresión* sea **FALSE** Ó IGUAL A CERO, la ejecución no se interrumpirá inmediatamente, sino que el efecto sólo se hará presente en el momento de evaluar a *Expresión*.

Debe comprender cabalmente el significado de lo que es una Sentencia, la cual puede llegar a ser una Simple Sentencia, de Asignación o llamado a función con una sola línea, como también puede ser una Sentencia Compuesta que abarque varias líneas de instrucciones.

2.3. Sentencia de repetición FOR

La sentencia o estructura repetitiva **FOR**, repite la ejecución de una o varias sentencias una cantidad de veces, que puede ser previamente establecida. Al igual que *while* y *do*, necesita de una forma para controlar el lazo y en este caso se hace con una variable de control, la cual necesariamente debe ser discreta (del tipo *Ordinal*), ya que el lazo se ejecuta mientras la variable de control toma una serie consecutiva de valores, comenzando por el valor de su inicialización.

La sentencia **FOR** tiene el siguiente formato:

FOR (*Variable de Control = Valor Inicial ; Expresión ; paso)*

<*Sentencia*>;

Donde *Sentencia* puede ser una simple sentencia o una sentencia compuesta a semejanza del ciclo *while*.

En caso de que el paso sea un incremento tenemos un **FOR** ascendente en donde los valores de la variable de control irán aumentando; mientras que al emplear un decremento tenemos un **FOR** descendente, en donde los valores de la variable de control van disminuyendo.

La variable de control de bucle y su valor inicial deben ser ordinales y del mismo tipo y no deben ser alterados por ninguna de las sentencias que componen el lazo.

El ciclo finaliza en el momento que al evaluar *Expresión* resulte el valor **FALSE** Ó IGUAL A CERO, esto hace que el flujo del programa salte la *Sentencia* del bucle, continuando la ejecución del programa en la siguiente sentencia. Dado que *Expresión* puede llegar a ser falsa inicialmente, es decir, antes de comenzar el bucle; habrá casos en que el ciclo no se ejecute.

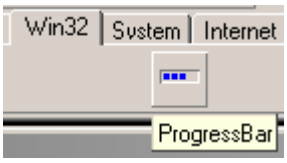
También debe tener muy en cuenta que aunque en el desarrollo de *Sentencia* se cambien los valores que hagan que *Expresión* sea **FALSE** Ó IGUAL A CERO, la ejecución no se interrumpirá inmediatamente, sino que el efecto sólo se hará presente en el momento de evaluar a *Expresión*. *Expresión* deberá ser de tipo lógica y no necesariamente contendrá comprobación del valor final de la variable de control.

Dado que los valores que toma la variable de control constituyen un conjunto de valores ordinales que se incrementa ó decrementa según el *paso*, el valor final de la variable de control será un paso mayor ó menor, según el caso, al que tenía en la última vuelta que realizó el ciclo.

Es de hacer notar que en la sección de inicialización y del paso (las que se encuentran al inicio y al final de la sentencia del **FOR**), se puede realizar más de una operación separándolas por coma. En el bloque central o *Expresión* se podrán realizar varias decisiones concatenándolas con operadores lógicos.

2.4. Componente ProgressBar

Este componente se encuentra en la pestaña *Win32* de la barra de componentes.



Es un elemento de apoyo visual para el usuario ya que representa una barra horizontal que se va "llenando" de izquierda a derecha, lo que permite representar el avance de un proceso.

Sus propiedades más relevantes son: ***Min***, ***Max*** que permiten definir el mayor y el menor valor de este componente (lo que viene a ser el tope izquierdo y el tope derecho); ***Step*** que define los saltos en la representación del rango ya indicado y ***Position*** que define el extremo derecho del llenado.

2.5. Componente TrackBar

Este componente se encuentra en la pestaña *Win32* de la barra de componentes.

Es representado por una barra (se puede colocar horizontal o vertical) que posee un marcador que puede ser desplazado por el usuario. Es un componente que permite al usuario ingresar valores enteros sin que tenga que escribirlos (lo cual elimina la posibilidad de una mala escritura).

Sus propiedades más relevantes son: ***Min***, ***Max***, ***Position*** y ***Frecuency***.

2.6. Componente ScrollBar

Este componente se encuentra en la pestaña *Standard* de la barra de componentes.

Es básicamente el mismo componente anterior sólo que es representado de una forma distinta, sus propiedades en esencia son las mismas y su principal función es la misma, la decisión de emplear un elemento u otro es más bien basada en un aspecto estético.

2.7. Componente UpDown

Este componente se encuentra en la pestaña *Win32* de la barra de componentes.

Es una variante del **SCROLLBAR** presentando sólo las flechas (una para incrementar y otra para decrementar).

Sus propiedades más relevantes son: ***Min***, ***Max*** que permiten definir el mayor y el menor valor de este componente y ***Position*** que define el valor correspondiente e ***Increment*** que define los saltos en la representación del rango indicado por *Min* y *Max*.

Como verá tiene a su disposición una serie de objetos que puede emplear para presentar al usuario una interfaz amigable al mismo tiempo que limita la ocurrencia de ciertos errores, dependerá de su criterio cuándo emplear uno u otro, la recomendación es que si bien estos son elementos muy útiles no exagere en su uso.

3. Actividades de Laboratorio

- **Primera Actividad**
  - Se dice que un número es primo palindrómico cuando es primo y al invertir el orden de sus cifras también lo es, como ejemplo se puede presentar el número 13 que es primo y al ser invertido es el número 31 que también es primo, por lo que 13 y 31 son número primos palindrómicos. Genere un programa que solicite un número entero positivo e indique si el mismo es o no un número primo palindrómico, para esto cree una forma en la cual insertará, al menos, los siguientes componentes:
  - Una caja de texto donde el usuario escribirá el número, acompañado de su correspondiente etiqueta para guiar al usuario, una etiqueta más donde escribirá el resultado pedido e inserte un botón el cual etiquetará como "Calcular" y asociado al evento *click* programe la solución pedida.
  - Para practicar un elemento visto (pero no usado) en la segunda sesión de laboratorio verifique que el número escrito sea positivo y en caso de no serlo avise al usuario de esta situación mediante una caja de diálogo (repase **ShowMessage**).
  - El proceso básico de solución de este problema debe ser discutido con el profesor del laboratorio, determinen cuál será el proceso de solución.
- **Segunda Actividad**
  - La serie de Fibonacci comienza de la siguiente manera: 1 – 1 – 2 – 3 – 5 – 8 – 13... etc, así que diseñará un programa que permita a un usuario conocer cuál es el N-ésimo término de esta serie. Para realizar lo anterior cree una forma en la cual insertará, al menos, los siguientes componentes:
  - Un componente **TrackBar** cuyo valor mínimo sea 1 y valor máximo sea 20, con una etiqueta al lado que le haga saber al usuario que este elemento se empleará para indicar cuantos términos de la serie se desea visualizar, también deberá colocarse otra etiqueta que presente el valor seleccionado en el **TrackBar** (la misma deberá ir cambiando a medida que se modifica la posición del cursor, lo cual se hará asociado al evento *Change* sobre este componente).
  - Inserte un botón el cual etiquetará como "Calcular" y asociado al evento *click* programe el cálculo de los primeros N términos de la serie, siendo N el número arrojado por el **TrackBar**. Decida cual es el ciclo más conveniente a emplear.
  - El proceso básico de solución de este problema debe ser discutido con el profesor del laboratorio, determine cuál será el proceso de solución.

▪ Use otras etiquetas para documentar adecuadamente lo que el usuario observa.

4. Problemas Asociados

1) Diseñe un programa que averigüe si un número "M" es perfecto o no. Sabiendo que un número es perfecto cuando la suma de todos sus divisores, salvo él mismo, es igual al número. Por ejemplo: 6 es dividido de manera exacta por 1, 2 y 3 que al sumarlos da 6.
2) Determine si un número "N" entero y positivo, leído como dato es o no un número primo, sin emplear ni la multiplicación, ni la división en ninguna de sus variantes, ni una función o estándar del Borland C.
3) El número e se puede calcular mediante la siguiente serie:

   i. $$e = 1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \ldots$$

   Diseñar un programa en Builder C que calcule el valor del número e detendrá el cálculo de la serie cuando el valor del último término generado sea menor que 10^{-5}. Debe indicar igualmente cuantos término fueron empleados en el cálculo.
4) Para encontrar el máximo común divisor (*mcd*) de dos números se emplea el algoritmo de Euclides, que se puede describir así: Dados los enteros a y b (a b), se divide a por b, obteniendo el cociente q1 y el resto r1. Si r1 > 0, se divide b por r1, obteniéndose el cociente q2 y el resto r2. si r2 > 0, se divide r1 por r2. obteniéndose restos y cocientes sucesivos. El proceso continua hasta obtener un resto igual a 0. El resto anterior a éste es el máximo común divisor de los números iniciales. Diseñar un programa en Builder C que calcule el máximo común divisor mediante el algoritmo de Euclides.
5) Diseñe un programa en Builder C que genere la serie:

$$S = Ln(Ln(X)) + Ln(X) + \frac{Ln^2(X)}{2} + \frac{Ln^3(X)}{3} + \frac{Ln^4(X)}{4} + \ldots$$

   Debe detener el cálculo cuando el último término generado sea menor, en valor absoluto, aun valor EPS leído como dato.
6) Dado un ángulo expresado en grados, Diseñe un programa en Builder C para determinar el valor del seno del mismo utilizando el desarrollo en serie de Mc Laurin y comparar el valor contra el que proporciona la función *sin* del Borland C. El número de términos es un dato y recuerde pasar el ángulo a radianes. El desarrollo en serie es:

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

Funciones

1. Objetivos

- En Borland C

  Definir e invocar funciones.
- En el Ambiente Builder

  Emplear el evento **ONKEYPRESS** sobre el objeto **EDIT** y la propiedad **SETFOCUS**.

2. Introducción

El Borland C es un lenguaje cuyo enfoque es hacia lo que se dio a conocer como modularización, en donde se busca dividir el programa (por ende el problema planteado) en grupos compactos de sentencias que van resolviendo partes específicas del problema, lo anterior presenta una serie de módulos que resuelven cada uno una tarea específica.

Así la evolución de lo que se conocían como subprogramas decantaron en las llamadas funciones Borland C.

Existen dos tipos de funciones: las que devuelven vacío y las que devuelven valores. En ambos casos se tiene un grupo de sentencias a las cuales se les engloba con un identificador y, cada vez que se les necesite se invocan tales sentencias al llamarlas por el identificador dado, la divergencia entre ambos es básicamente operativa y serán estudiadas en las próximas prácticas empezando por las funciones que devuelven valores.

3. Definición de Funciones

Las funciones que devuelven valores tienen la siguiente estructura:

- una cabecera que comienza con el tipo de resultado asociado a la función y luego se indica el identificador o nombre con el que será llamada la función y la lista de *parámetros* o datos que alimentan la función.
- una sección de declaraciones igual a la empleada hasta los momentos
- y una zona de sentencias que son las que resuelven el problema propio de la función.

El esquema es:

Cabecera:	Tipo de la función *Identificador*(lista de *parámetros*)
Inicio de la Función	**{**
Declaraciones Locales:	<Declaración de variables locales a la función> <Declaración de otros procedimientos y funciones>
Cuerpo	**INSTRUCCIONES** ....
Fin de la Función	**}**

Tabla 1

Debe resaltarse que en la cabecera y al inicio se debe indicar el tipo de dato que devuelve la función, esto es debido a que la función devuelve un valor y al ser invocada se tiene un resultado asociado, haga memoria y piense en las funciones que seguramente ha usado, tales como las del Coseno, la Raíz Cuadrada y otras que ya vienen como parte del lenguaje pero que han sido desarrolladas bajo el mismo concepto.

En el cuerpo de la función que devuelve valor se debe colocar la instrucción **return,** seguida de una variable local a la función, al menos una vez y preferiblemente al final de la misma. La variable que sigue a la instrucción **return** es la que contiene el valor que se transmite o se envía al lugar donde se realiza el llamado. Dependiendo de la lógica propuesta puede presentarse la instrucción return más de una vez en el cuerpo de la función.

Es importante resaltar que debe garantizarse que la variable que se coloca en la instrucción **return**, al menos una vez, reciba un valor y debe ser declarada del mismo tipo de dato que la función.

Aunque de las siguientes operaciones algunas pueden realizarse sin generar errores, como buena práctica de programación y apegándose al espíritu de existencia de las funciones que devuelven valores, debe evitar lo siguiente:

- Leer datos dentro del cuerpo de sentencias de la función, todos los datos necesarios deben pasar como *parámetros*.
- Presentar el resultado en pantalla, la función sólo se usa para calcular y la presentación del resultado se realizará posiblemente en el segmento de programa que realizó el llamado.
- Pretender calcular varias cosas, para eso están las funciones vacías o que devuelven vacío.
- Realizar verificaciones sobre los datos de entrada e indicar si los mismos no sirven, esto debe efectuarse antes del llamado de la función (si no está del todo convencido recuerde la función que calcula la raíz cuadrada de un número, si se le introducen números negativos se produce un error de ejecución).

4. El intercambio de información: Los Parámetros

Como ya habrá observado, previamente se menciona en varias ocasiones la palabra *Parámetro* que como se indicó es la forma de suministrar datos a una función, como analogía considere una función como una taquilla en donde se realiza un cálculo específico, así que cada vez que se necesite ese cálculo se le invoca al emplear su identificador (imagine que es el nombre de dicha taquilla), los datos necesarios para realizar las operaciones pertinentes a la función se pasan por esta taquilla mediante una bandeja, esta bandeja vienen a ser los parámetros de entrada.

Más formalmente, los parámetros son canales de comunicación para pasar datos entre programas y subprogramas en ambos sentidos (en el caso de la función que devuelve valores un solo sentido). Los parámetros van asociados a variables; constantes, expresiones, etc., y, por tanto, se indican mediante los correspondientes identificadores o expresiones. Los parámetros que se emplean en la llamada a un subprograma son los que entregan la información, la reciben del mismo y se declaran en la cabecera del subprograma. Dependiendo de la tarea que desempeña el subprograma, el mismo puede no requerir información, por lo que no posee parámetros.

En una llamada a un subprograma tiene que verificarse que:

- El número de parámetros en la llamada debe ser igual al número de parámetros en la declaración.
- Los parámetros deben ocupar el mismo orden en cada una de las listas (llamada y declaración) y deben ser compatibles en tipo de dato.

4.1. Parámetros por valor y por referencia o dirección

Paso por valor. Cuando un parámetro se pasa por valor, el subprograma hace una copia del valor de éste en una posición de memoria idéntica en tamaño pero distinta en ubicación a la del parámetro que se envía. Como el subprograma trabaja a partir de sus parámetros formales, si durante la ejecución se modifica un valor correspondiente a un paso por valor, el contenido de la posición de memoria del parámetro en el segmento de programa que realizó el llamado no se verá alterado.

Puede decirse que el paso por valor es un canal de transferencia de información con una sola dirección, es decir, de entrada al subprograma.

Lo anterior permite, si así se desea, que en el llamado sea colocado de forma indistinta una constante o una variable

Este es el único tipo de parámetro que emplearemos con las funciones que devuelven valores.

Parametros por Referencia o Direccion. Cuando el parámetro se pasa por dirección o referencia se hace una copia de la dirección donde se encuentra almacenada la variable original, por lo cual ésta sufrirá, en la llamada, toda modificación que se realice sobre su variable reflejo dentro de la función. Se denomina por Dirección porque el parámetro indicará la dirección de memoria donde se encuentra la original.

Puede decirse que el paso por dirección o referencia es un canal de transferencia de información bidireccional, es decir, de entrada y salida al subprograma.

4.2. Declaraciones Globales y Locales

Todos los objetos (constantes, tipos, variables, etc.) declarados en el programa principal son reconocidos y, por tanto, pueden ser empleados, por cualquiera de los subprogramas llamados por el programa principal. Se dice que su alcance o dominio es global.

Todos los objetos que un subprograma necesite para su uso exclusivo se definirán en la sección de declaraciones correspondiente a cada subprograma, serán propios de éste, e inaccesibles para el programa principal, esto es, que su ámbito es local, teniendo sólo vigencia dentro del subprograma u otro subprograma interno a éste.

5. Llamada a una función

El Borland C trata a la llamada de una función como un valor, por lo que puede aparecer en los mismos sitios en que pueda aparecer un valor, por ejemplo: asignación a una variable, como componente de una expresión, como parámetro de un procedimiento, etc, pudiéndose invocar tantas veces como se considere necesario y lo que le da una gran utilidad es que siempre se pueden emplear diferentes parámetros, y recuerde:

> Deben emplearse funciones que devuelven valores cuando sólo tenga que devolverse un solo valor simple al programa que realiza la llamada. En todos los demás casos se deben emplear funciones vacías.

6. El evento ONKEYPRESS y la propiedad SETFOCUS

Trabajando en el ambiente visual existen ciertas acciones a las que está acostumbrado el usuario, una de ellas cuando está escribiendo datos en una caja de edición, es presionar la tecla **Enter** para indicar que se ha concluido el ingreso de caracteres, lo que espera es que el cursor se ubique en el siguiente elemento de trabajo en la pantalla. Así que primero debe saber cuando el usuario presionó la tecla referida, para ello dispone de evento **ONKEYPRESS** que se activa cada vez que el usuario presiona una tecla.

Por ejemplo para una caja de edición (**Edit**) recuerde que cuando en tiempo de diseño desea asignarle un controlador de evento hace doble clic sobre este componente se le traslada al área escritura de sentencias y ve que ha sido creado el cuerpo de una función que responde al evento más común del componente (que en el caso de **Edit** es ***Change***) pero si se desea otro controlador de evento lo que se debe hacer es tocar, o marcar, el objeto con el cursor y en la caja de *Inspección del Objeto* (Object Inspector) presionar sobre la pestaña *Events*, en este lugar se observa todos los eventos a los que puede responder el objeto marcado, también verá que hay dos columnas: la de la izquierda tiene el nombre del evento y la de la derecha está inicialmente en blanco. Ubique la palabra **ONKEYPRESS** y en su columna de la derecha presione doble clic, ahora verá que pasa a la zona de escritura de código en una función de respuesta al evento de cuando el usuario presiona una tecla.

En la lista de parámetros verá que tiene una variable llamada **Key** tipo **CHAR**, esta variable contiene el valor de la tecla presionada.

Por otro lado, pero en línea con lo planteado en el primer párrafo de esta sección, la propiedad **SETFOCUS** permite que el "enfoque" sea trasladado a un objeto ubicado en la forma, antes de continuar es bueno aclarar que el "enfoque" se aplica en aquellos objetos sobre los cuales el usuario tiene interacción, por ejemplo una caja de edición en cuyo caso observará el cursor parpadeando dentro de dicha caja, en este caso parte de la atención del ambiente visual estará centrada en donde se tiene el enfoque.

Dado lo anterior una combinación del manejo del evento **ONKEYPRESS** y la propiedad **SETFOCUS** permitirá a un usuario escribir en una caja de diálogo y al presionar la tecla **Enter**, se hará un "salto" hacia otro objeto, por ejemplo otra caja de diálogo.

El siguiente segmento es un ejemplo de un caso como el explicado:

```
:
If (Key = = 13)
 Edit2->SETFOCUS;
```

En este ejemplo se evalúa si la tecla presionada (guardada en la variable *Key* proporcionada por la función **KEYPRESS**) es igual a la tecla **Enter**, (que en el caso de la tecla **Enter** el valor ASCII es 13), en caso de ser cierta la comparación se trasladará el enfoque a otra caja de edición (Edit2). Se debe aclarar que el evento **ONKEYPRESS** se activa cada vez que se presiona una tecla sobre el objeto.

7. Actividades de Laboratorio

Va realizar un programa que solicite al usuario las coordenadas x, y de tres puntos que formen un triángulo, también se solicitarán las coordenadas de un cuarto punto e indicará si ese último punto dado está dentro del triángulo formado por los tres primeros puntos.

- Cree una forma en la cual insertará, al menos, los siguientes componentes:
- Ocho cajas de edición y sus correspondientes etiquetas para solicitar las coordenadas de los puntos a trabajar (P1, P2, P3 y P4). En las cajas de edición debe evaluar la acción muy

común por parte del usuario de presionar la tecla **Enter** para indicar que finalizó de escribir el valor, en ese caso el enfoque debe saltar a la siguiente caja de edición, y en caso de la última el enfoque debe ser realizado al botón para calcular.

- Inserte un botón el cual etiquetará como "Calcular" y asociado al evento **click** programe lo solicitado, pero bajo los siguientes lineamientos:
  - Primero debe verificar si los tres primeros puntos forman triángulo, para ello genere una función que devuelva valor cuyo resultado sea: True ó distinto de cero si forman triángulo y False ó igual a cero en caso contrario, de no formar triángulo debe dárselo a conocer al usuario mediante una caja de diálogo.
  - Presente la solución mediante el siguiente razonamiento: si se forman tres pequeños triángulos teniendo como elemento común el punto a evaluar (P4), la suma de las áreas de los mismos debe ser igual al área del triángulo inicial si el cuarto punto se encuentra dentro, en caso contrario la suma de dichas áreas es mayor. Presente el mensaje pertinente en una etiqueta que aparecerá en el momento adecuado.

    Para calcular el área de un triángulo lo puede hacer mediante la siguiente fórmula:

    $$Area = \left(P * \left(P - A\right) * \left(P - B\right) * \left(P - C\right)\right)^{1/2}$$

    Donde P se define como el semiperímetro y $P = \left(A + B + C\right)/2$ y A, B y C son los lados del triángulo, dado por la distancia de sus vértices.

    En base a lo anterior programe una función que devuelva valor que calcule la distancia entre dos puntos, otra función que calcule el área de un triángulo según la fórmula dada y aplique adecuadamente las funciones anteriores para responder si el cuarto punto se encuentra, o no, dentro del triángulo originado por los tres primeros puntos.

El proceso básico de solución de este problema debe ser discutido con el profesor del laboratorio. Discutan cuál es el lugar más adecuado para declarar las funciones que serán usadas.

Es prudente discutir qué sucede si el punto se encuentra justo en el borde del triángulo.

5.- Recursividad

Un objeto es recursivo cuando puede definirse en función de sí mismo. El Borland C permite que las funciones se llamen o invoquen a sí mismos, es decir, que sean recursivos, y así resolver algunos problemas de manera sencilla y elegante, aunque difícil de visualizar por parte del programador y costosa desde el punto de vista de los recursos del computador. Lo último se debe a que cada llamada de función recursiva no es más que la llamada a una copia de sí misma. Comportándose cada copia como un subprograma independiente, ya que las variables locales y los parámetros correspondientes, existen de modo distinto en cada una de las llamadas.

Como ejemplo, tomemos la siguiente función que de forma recursiva calcula la suma de los valores enteros desde 1 a **N**.

```
Int Sumar (Int N)
{   If (N = = 1)
        Sumar = 1;
    Else
        Sumar = Sumar (N-1)+N; }
```

8. Problemas propuestos:

a.- Diseñar una función que devuelva el factorial de un número **N** entero y positivo.

b.- Realizar una función que devuelva el logaritmo en base diez de un número cualquiera.

c.- Escribir una función recursiva que determine el capital C_n obtenido, al situar un capital inicial C_0 a interés compuesto durante *n* años al interés anual *r* (expresado en porcentaje). Siendo la fórmula de interés compuesto:

$$C_n = C_0 * \left(1 + r / 100\right)^n$$

Trabajando con Arreglos - Vectores

1. Objetivos

- En Borland C

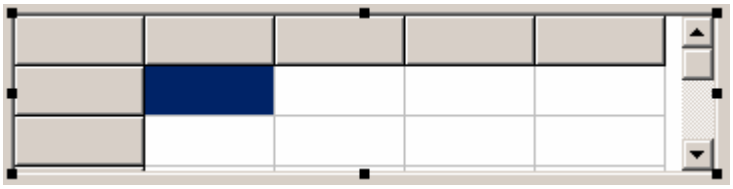
  Declarar y emplear correctamente la estructura de arreglos
- En el Ambiente Builder

  Emplear el objeto **StringGrid**

2. Introducción

Componente StringGrid

Es un elemento idóneo para manipular los valores contenidos en los arreglos de Borland C al permitir presentar caracteres en forma tabulada, lo puede localizar en la Barra de Herramientas, en la pestaña "***Additional***"; su inserción sobre la forma genera una cuadrícula similar a la mostrada:



De este componente las propiedades que debe manipular fluidamente son las siguientes:

ColCount y ***RowCount*** permiten definir el número de columnas y de filas que tendrá la rejilla.

FixedCols y ***FixedRows*** permiten definir cuantas de las columnas y de las filas se verán con un relleno sólido; las mismas se ubican en el extremo izquierdo y el extremo superior de la cuadrícula formada.

Con ***Cells*** se puede acceder a una posición en particular de la rejilla, por ej:

StringGrid1->Cells[C][F]= Valor;

Col y ***Row*** permiten saber en tiempo de ejecución sobre cual columna y/o fila se encuentra el usuario trabajando, o indicando estos valores se mueve el enfoque en determinada casilla.

ColWidths permite establecer el ancho de una columna modificando el valor por defecto que se indica en ***DefaultColWidth*** propiedad que afecta a todas las columnas de la rejilla.

En este elemento es importante resaltar que en los subíndices primero se coloca la columna y luego la fila y que la numeración, en ambos casos, comienza en cero (0).

También se debe estar consciente que los valores guardados en las casillas son del tipo alfanumérico, por lo que si se quiere trabajar con números se deben emplear las funciones que ofrece Borland C para tales fines.

Si se quiere presentar un valor en una casilla de la rejilla, se asigna directamente empleando la propiedad ***Cells***, pero para que el usuario pueda escribir en las diferentes celdas se debe tener presente que en la barra de propiedades (en tiempo de diseño) se debe colocar en ***True*** el valor de ***goEditing*** en el apartado de ***Options***.

Variables Multidensionadas - Vectores

Esta herramienta permite emplear un único identificador para un grupo de posiciones de memoria homogéneas en el tipo de dato que contienen. Y se accede de forma individual a cada una de ellas mediante un subíndice. Con esto se forman los conocidos vectores y matrices.

Su forma de declaración puede ser de varias maneras, una de ellas es:

TIPO DE DATO *Identificador* [*Máximo número de Elementos*] ;

Máximo número de Elementos debe ser una constante del tipo Ordinal o discretos (valores enteros positivos), la cual define el número de casillas disponibles para trabajar o la Dimensión Máxima.

Para que un vector pueda ser empleado como parámetro de una función no es necesario colocar el *Máximo número de Elementos*, ya que el Borland C automáticamente le asignará el valor que contiene la declaración original

Para los alcances de este curso, *Máximo número de Elementos* no puede ser un dato y debe ser prefijado por el programador previendo los valores máximos con los que será usualmente empleado el programa, lo anterior implica que se debe ser consistente con esta declaración y se debe verificar que el usuario no sobrepase esta cantidad.

Sobre la base de lo anterior se debe resaltar que es usual solicitar al usuario la dimensión del arreglo, esto es lo que se conoce como la Dimensión de Trabajo y que, como se mencionó, nunca deberá sobrepasar la Dimensión Máxima.

Debe estar consciente de que difícilmente encontrará una función que ejecute una acción sobre todos los componentes con sólo indicar el identificador del arreglo, en la gran mayoría de los casos deberá ir casilla por casilla indicando la operación a ejecutar. De lo anterior podemos resumir, que trabajar con vectores termina siendo, en buena medida, una manipulación adecuada de los índices.

3. Actividades de Laboratorio

- Cree una forma en la cual insertará, al menos, los siguientes componentes:
- Un componente **TrackBar** cuyo valor mínimo sea 1 y valor máximo sea 30, con una etiqueta al lado que le indique al usuario que este elemento se empleará para dar la dimensión de trabajo del vector, también deberá colocarse otra etiqueta que presente el valor seleccionado en el **TrackBar**.
- Inserte un **StringGrid** bajo las siguientes condiciones: en el momento que sea cargada la forma deberá poner a punto los valores iniciales del **StringGrid** de tal manera que se visualice una rejilla en forma de arreglo lineal (recomendamos una distribución horizontal) con un ancho de casilla adecuado para colocar no más de seis (6) caracteres, pensando en trabajar con valores numéricos enteros (tipo **Int** serán adecuados)
- Coordine el valor inicial del **TrackBar** y del **StringGring**.
- Inserte un botón el cual etiquetará como “Calcular” y asociado al evento **click** programe un proceso que calcule la frecuencia de repetición de los elementos en el vector original.

El proceso básico de solución de este problema debe ser discutido con el profesor del laboratorio, determinen cuáles serán las funciones necesarias para la solución. Discutan cuál es el lugar más adecuado para declarar estos elementos.

- Presente el resultado pedido en un nuevo **StringGrid** que muestre dos filas, una para el vector reducido, en casilla de relleno sólido y la otra para el vector de frecuencias.

- Use otras etiquetas para documentar adecuadamente lo que el usuario observa.

Actividad Adicional

- Como ha podido experimentar la edición en el **StringGrid** no es tan cómoda como ha observado en programas comerciales que presentan una estructura similar, como una primera mejora lleve a cabo lo siguiente: cuando el usuario escriba un valor sobre una casilla y presione la tecla "*Enter*" o "*Retorno*" haga que el enfoque automáticamente pase a la siguiente casilla disponible, como ayuda le indicamos que pruebe con las propiedades **Col** y/o **Row** en un evento **KeyDown** o **KeyPress** sobre el **StringGrid**.

4. Problemas Asociados

A. Ordene de menor a mayor un arreglo unidimensional A de N componentes.

B. Dado un arreglo lineal A de N componentes, diseñe un proceso donde sea "eliminado" un valor seleccionado por el usuario. Debe entenderse por "eliminar" que se debe obtener que el arreglo A ahora sea de N-1 componentes.

C. Dado un arreglo lineal A de N componentes, diseñe un proceso que elimine los elementos repetidos dentro del vector original.

D. Dado un vector A de N componentes y un vector B de M componentes, obtener un tercer vector C que sea la unión de los vectores A y B. El resultado no debe presentar valores repetidos.

E. Dado un vector A de N componentes y un vector B de M componentes, obtener un tercer vector C que posea sólo los elementos comunes entre los vectores A y B. El resultado no debe presentar valores repetidos.

F. Dado un arreglo unidimensional A de N componentes, construya un segundo arreglo B también de N elementos, donde cada B(I) contenga la posición que ocuparía A(I) si el mismo estuviese ordenado de menor a mayor.

G. Inserte un valor dado como dato en la posición que le corresponde dentro de un arreglo A de N componentes, el cual está ordenado de menor a mayor.

H. Realice un programa que calcule el ángulo entre dos vectores.

I. Realice un programa que dada una cantidad de dinero ingresada por el usuario indique la cantidad mínima necesaria de cada tipo de billete y/o moneda de uso corriente para cubrir el monto ingresado. Desprecie los centavos. Resuelva este problema empleando arreglos lineales.

J. Realice un programa que dada una cantidad de dinero ingresada por el usuario presente tal cifra en palabras. Desprecie los centavos. Resuelva este problema empleando arreglos lineales.

Trabajando con Arreglos – Matrices

1. Objetivos

- En Borland C

  Declarar y emplear correctamente la estructura de arreglos
- En el Ambiente Builder

  Emplear el objeto **StringGrid**

2. Introducción

Componente StringGrid

Es un elemento idóneo para manipular los valores contenidos en los arreglos de Borland C al permitir presentar caracteres en forma tabulada, su inserción sobre la forma y su manipulación se detalló en el capítulo anterior.

En este elemento es importante resaltar que en los subíndices primero se coloca la columna y luego la fila y que la numeración, en ambos casos, comienza en cero (0).

Variables Multidensionadas - Matrices

Esta herramienta permite emplear un único identificador para un grupo de posiciones de memoria homogéneas en el tipo de dato que contienen y que el usuario visualiza comúnmente como una tabla. Se accede de forma individual a cada una de ellas mediante dos subíndices, uno que controla las filas y otro que controla las columnas.

Su forma de declaración puede ser de varias maneras, una de ellas es:

TIPO DE DATO *Identificador* [*Máximo número de Filas*] [*Máximo número de Columnas*];

Máximo número de Filas (Columnas) debe ser una constante del tipo Ordinal o discreto (valores enteros positivos), la cual define el número de filas ó columnas disponibles para trabajar o la Dimensión Máxima.

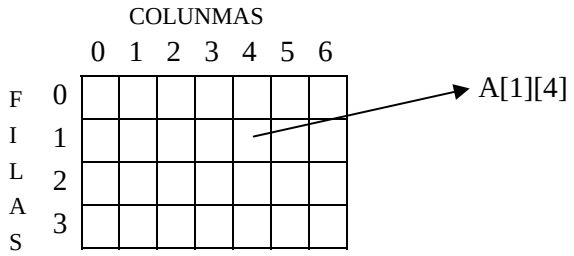
Para que una matriz pueda ser empleada como parámetro de una función no es necesario colocar el *Máximo número de Filas,* ya que el Borland C automáticamente le asignará el valor que contiene la declaración original, pero si es imprescindible colocar el *Máximo número de Filas.*

Para los alcances de este curso, *Máximo número de Filas (Columnas)* no puede ser un dato y debe ser prefijado por el programador previendo los valores máximos con los que será usualmente empleado el programa, lo anterior implica que se debe ser consistente con esta declaración y se debe verificar que el usuario no sobrepase esta cantidad.

Sobre la base de lo anterior se debe resaltar que es usual solicitar al usuario la dimensión de la matriz, esto es lo que se conoce como la Dimensión de Trabajo y que, como se mencionó, nunca deberá sobrepasar la Dimensión Máxima.

Como se mencionó en el capítulo anterior, difícilmente encontrará una función que ejecute una acción sobre todos los componentes con sólo indicar el identificador de la arreglo, en la gran mayoría de los casos deberá ir casilla por casilla indicando la operación a ejecutar. De lo anterior podemos resumir, que trabajar con matrices termina siendo, en buena medida, una manipulación adecuada de los índices.

Para trabajar una matriz, casilla por casilla, se deberá indicar primero la fila y luego la columna que ocupa dicha casilla. Por ejemplo para la matriz de nombre A que se muestra:



3. Actividades de Laboratorio

- Cree una forma en la cual insertará, al menos, los siguientes componentes:
- Dos componentes **TrackBar** cuyo valor mínimo sea 1 y valor máximo sea 30, con una etiqueta al lado que le indique al usuario que estos elementos se emplearán para dar la dimensión de trabajo de la matriz, también deberá colocarse dos etiquetas que presente el valor seleccionado en los **TrackBar**.
- Inserte un **StringGrid** bajo las siguientes condiciones: en el momento que sea cargada la forma deberá poner a punto los valores iniciales del **StringGrid** de tal manera que se visualice una rejilla en forma de tabla con un ancho de casilla adecuado para colocar no más de seis (6) caracteres, pensando en trabajar con valores numéricos enteros (tipo **Int** serán adecuados)
- Coordine el valor inicial de los **TrackBar** y del **StringGring**.
- Inserte un botón el cual etiquetará como "Calcular" y asociado al evento **click** programe un proceso que calcule los puntos de silla de la matriz original.

El proceso básico de solución de este problema debe ser discutido con el profesor del laboratorio, determinen cuáles serán las funciones necesarias para la solución. Discutan cuál es el lugar más adecuado para declarar estos elementos.

- Presente el resultado pedido en un nuevo **StringGrid** que muestre un vector con los puntos de silla de la matriz.
- Use otras etiquetas para documentar adecuadamente lo que el usuario observa.

Actividad Adicional

- Como ha podido experimentar la edición en el **StringGrid** no es tan cómoda como ha observado en programas comerciales que presentan una estructura similar, como una primera mejora lleve a cabo lo siguiente: cuando el usuario escriba un valor sobre una casilla y presione la tecla "*Enter*" o "*Retorno*" haga que el enfoque automáticamente pase a la siguiente casilla disponible de esa fila, y al presionar la tecla "*Enter*" o "*Retorno*" en la última casilla de una fila, haga que el enfoque automáticamente pase a la primera casilla de la siguiente fila. Como ayuda le indicamos que pruebe con las propiedades **Col** y/o **Row** en un evento **KeyDown** o **KeyPress** sobre el **StringGrid**.

Trabajando con Archivos Tipo Texto

1. Objetivos

- En Borland C

  Utilizar correctamente los archivos secuenciales o tipo texto
- En el Ambiente Builder

  Utilizar los objetos **DRIVECOMBOBOX**, **DIRECTORYLISTBOX** y **FILELISTBOX.**

2. Introducción

En esta práctica Ud. va a enfrentarse a problemas donde deberá manipular archivos, los cuales en este caso serán de los llamados tipo texto o secuenciales, para esto y dejando de lado los problemas propios de ellos, se enfrenta ante la necesidad de ofrecer al usuario una pantalla amigable que evite, en lo posible, que deba escribir los nombres de tales archivos con sus ubicaciones, para ello debe presentar elementos que se han vuelto comunes en los sistemas operativos de ambiente gráfico, que por un lado facilita la labor del usuario y que por el otro minimiza las posibilidades de errores. Tales componentes se presentan a continuación:

Componente DRIVECOMBOBOX

Este componente se encuentra en la pestaña *Win 3.1* de la barra de componentes.

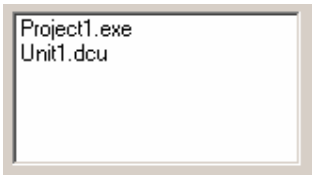
Permite presentar las unidades y conexiones de red disponibles en el equipo que se ejecuta la aplicación, el objeto colocado en la forma se observa como se muestra:

Componente 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. De forma automática si el usuario hace doble *click* se expande (o contrae) la rama de un subdirectorio que contenga subdirectorios sin que Ud. como programador tenga que preocuparse como se realiza tal acción, ya que es una acción propia de este elemento.

Componente FILELISTBOX



Este componente se encuentra en la pestaña *Win 3.1* de la barra de componentes.

Presenta todos los archivos de un subdirectorio, indicando su nombre y extensión. Uno puede filtrar los archivos a ser mostrados, ya que se pueden emplear los comodines usuales en sistema operativo (***** para sustituir cada carácter en cualquier cantidad y **?** para sustituir un único carácter).

Por sí solos estos tres elementos trabajan de forma totalmente independiente y se deben sincronizar en su accionar para lograr que cuando un elemento cambie los otros también lo hagan de manera adecuada e instantánea a los ojos del usuario. Para lo anterior se debe entender que cuando se cambia el elemento que muestra las unidades (**DRIVECOMBOBOX**) el

efecto también debe ser provocado en el elemento que muestra el árbol de directorios (**DIRECTORYLISTBOX**) y a su vez, cuando éste último cambia debe reflejarse en la lista de archivos (**FILELISTBOX**).

Una vez colocados los elementos anteriores sobre la forma se debe escribir las siguientes líneas de código en los eventos pertinentes, por ejemplo:

```
void __fastcall TForm1::DriveComboBox1Change(TObject *Sender)
 {
   DirectoryListBox1->Drive = DriveComboBox1->Drive;
 }

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

Archivos Tipo Texto

Los archivos son una forma de guardar datos de manera permanente. Pueden ser empleados como receptores de información final o pueden emplearse para servir de puente entre un programa y otro.

De manera informal, un archivo se puede definir como una unidad de información que tiene nombre.

Debido a que los archivos de caracteres se emplean frecuentemente, existe un tipo de archivo estándar denominado **TEXT**. Estos archivos están compuestos por líneas separadas por marcas de fin de línea (*EOLN*). Ya que no poseen una estructura en particular salvo los elementos que marcan final de línea y fin de archivo, los archivos tipo texto son muy simples. Esta simpleza tiene sus puntos favorables y sus objeciones.

Para manipular este tipo de archivo se deben emplear ciertos procedimientos, los mismos se describen a continuación:

FOPEN

Para trabajar con los archivos tipo texto todos los procedimientos disponibles se relacionan con el mismo mediante tipo archivo, esta variable se declara como **FILE *** en la zona de declaraciones, y la misma se debe inicialmente asociar con un nombre de archivo en el sistema operativo, para esto se emplea la función **fopen**.

Un archivo externo es típicamente un nombre de archivo de disco (***DOS***), pero también puede corresponder a un periférico.

Su estructura es la siguiente:

F = fopen (“NombreArchivo”, “Modo”);

F : es una variable tipo archivo (FILE)

NombreArchivo: es una cadena alfanumérica que indica el camino y el nombre completo del archivo que se desea manipular.

Modo: Es la forma como se manipulará dicho archivo

wt: Abre un archivo por primera vez para escribir en él.

at: Abre un archivo existente para anexarle datos al final del mismo.

Rt: Abre un archivo existente para leer información.

Si se abre un archivo para escribir o anexar datos se pueden emplear las funciones de Salida para archivos: **fprintf**.

Que en su forma más general se invoca como:

fprintf(F, "Formato", Var1, Var2,…, VarN**);**

F : es una variable tipo archivo (FILE)

Formato: es una cadena alfanumérica que indica el tipo de las variables que se van a escribir en el archivo.

%f: float.

%d: entero con signo.

%u: entero sin signo

%c: carácter alfanumérico

%s: cadena alfanumérica.

Var1, Var2, ..., VarN: Variables a escribir su valor en el archivo.

Si se abre un archivo para leer datos se pueden emplear las funciones de Entrada para archivos: **fscanf**.

Que en su forma más general se invoca como:

fscanf(F, "Formato", &Var1, &Var2,…, &VarN**);**

F : es una variable tipo archivo (FILE)

Formato: es una cadena alfanumérica que indica el tipo de las variables que se van a escribir en el archivo.

%f: float.

%d: entero con signo.

%u: entero sin signo

%c: carácter alfanumérico

%s: cadena alfanumérica.

Var1, Var2, ..., VarN: Variables a leer su valor en el archivo.

FCLOSE

Finaliza la asociación existente entre un archivo en el disco y la variable tipo archivo que se indicó previamente mediante ***fopen***.

Su estructura es la siguiente: **fclose**(F);

3. Actividades de Laboratorio

- Cree una forma en la cual insertará, al menos, los siguientes componentes:
- Un componente **DRIVELISTBOX**, un componente **DIRECTORYLISTBOX** y un componente **FILELISTBOX** que permita al usuario navegar por las unidades del equipo en que trabaja

para poder ubicar archivos, en particular el componente **FILELISTBOX** debe ser preparado para sólo mostrar archivos con extensiones .**Dat**.

- Programe el evento clic del botón para que calcule la media y la desviación estándar de los valores que se encuentran en dicho archivo.

4. Problemas Asociados

A. En un archivo tipo texto se encuentran las coordenadas de un indeterminad número de puntos en el espacio, dados por sus coordenadas **X**, **Y** y **Z**. Realice un programa que indique que porcentaje de esos puntos se encuentran dentro de una esfera con centro en el origen y de radio **R** (dado como dato por el usuario).

B. Un móvil se desplaza a través del espacio desde un punto inicial de salida hasta un punto final de llegada. Cada tanto se hacen mediciones de sus coordenadas espaciales (en metros), llegando a registrarse diez (10) puntos sucesivos. Por tanto, la trayectoria del móvil queda subdividida en tramos rectilíneos no uniforme en el tiempo. En cada medición se registra, además, la hora, minutos y segundos de pasada del móvil.

   Los datos están incorporados en un archivo tipo texto donde cada línea corresponde a una medición en tiempo y espacio.

   Se pide escribir un programa en Builder C que genere una tabla con la información de la velocidad del móvil en cada tramo (en m/seg)

C. Escriba un programa que permita a un profesor usar el computador para calificar a sus alumnos en un examen de opciones múltiples, que consiste en diez preguntas. El archivo de datos contendrá dos líneas para cada alumno: la primera tendrá el nombre del alumno, y la segunda, las respuestas a las diez preguntas. Cada pregunta de opción múltiple tiene cinco items y una sola respuesta correcta, y posee un valor de dos puntos. El archivo tendrá como primera línea la clave de respuestas para efectuar la corrección.

   La salida del programa debe consistir en lo siguiente:

   - El nombre y la calificación para cada alumno
   - La calificación promedio del grupo