

LENGUAJES ELECTRÓNICOS (Parte 1) Versión 8-3-12A

Lenguaje de programación

Un lenguaje de programación es un lenguaje artificial que puede ser usado para controlar el comportamiento de una máquina, especialmente una computadora. Estos se componen de un conjunto de reglas sintácticas y semánticas que permiten expresar instrucciones que luego serán interpretadas.

Debe distinguirse de “lenguaje informático”, que es una definición más amplia, puesto estos incluyen otros lenguajes como son el HTML o PDF que dan formato a un texto y no es programación en sí misma.

El programador es el encargado de utilizar un lenguaje de programación para crear un conjunto de instrucciones que, al final, constituirá un programa.

Existen dos tipos de lenguajes claramente diferenciados; los lenguajes de bajo nivel y los de alto nivel.

El ordenador sólo entiende un lenguaje conocido como código binario o código máquina, consistente en ceros y unos. Es decir, sólo utiliza 0 y 1 para codificar cualquier acción.

Los lenguajes más próximos a la arquitectura hardware se denominan lenguajes de bajo nivel y los que se encuentran más cercanos a los programadores y usuarios se denominan lenguajes de alto nivel.

Lenguajes de bajo nivel

Son lenguajes totalmente dependientes de la máquina, es decir que el programa que se realiza con este tipo de lenguajes no se pueden migrar o utilizar en otras máquinas.

Al estar prácticamente diseñados a medida del hardware, aprovechan al máximo las características del mismo.

Dentro de este grupo se encuentran:

- El **lenguaje máquina**, este lenguaje ordena a la máquina las operaciones fundamentales para su funcionamiento. Consiste en la combinación de “0” y “1” para formar las ordenes entendibles por el hardware de la máquina. Este lenguaje es mucho más rápido que los lenguajes de alto nivel. La desventaja es que son bastantes difíciles de manejar y usar, además de tener códigos fuente enormes donde encontrar un fallo es casi imposible.
- El **lenguaje ensamblador** es un derivado del lenguaje máquina y está formado por abreviaturas de letras y números llamadas mnemotécnicos. Con la aparición de este lenguaje se crearon los programas traductores para poder pasar los programas escritos en lenguaje ensamblador a lenguaje máquina. Como ventaja con respecto al código máquina es que los códigos fuentes eran más cortos y los programas creados ocupaban menos memoria. Las desventajas de este lenguaje siguen siendo prácticamente las mismas que las del lenguaje ensamblador, añadiendo la dificultad de tener que aprender un nuevo lenguaje difícil de probar y mantener.

Lenguajes de alto nivel

Son aquellos que se encuentran más cercanos al lenguaje natural de los seres humanos, que al lenguaje de la máquina.

Se tratan de lenguajes independientes de la arquitectura del ordenador. Por lo que, en principio, un programa escrito en un lenguaje de alto nivel, se puede migrar de una máquina a otra sin ningún tipo de problema.

Estos lenguajes permiten al programador olvidarse por completo del funcionamiento interno de la máquina para la que está diseñando el programa. Tan sólo es necesario un traductor que entienda el código en que está hecho el programa, como las características de la máquina en la cual queremos ejecutar dicho programa.

Suelen usar tipos de datos para la programación y hay lenguajes de propósito general (cualquier tipo de aplicación) y de propósito específico.

Lenguajes de Medio nivel

Se trata de un término no aceptado por todos. Estos lenguajes se encuentran en un punto medio entre los dos anteriores. Dentro de estos lenguajes podría situarse el lenguaje C, ya que puede acceder a los registros del sistema, trabajar con direcciones de memoria, todas ellas características de lenguajes de bajo nivel y a la vez realizar operaciones de alto nivel.

Generaciones

La evolución de los lenguajes de programación se puede dividir en 5 etapas o generaciones.

- Primera generación: lenguaje máquina.
- Segunda generación: se crearon los primeros lenguajes ensambladores.
- Tercera generación: se crean los primeros lenguajes de alto nivel. Ej. C, Pascal, Cobol, etc.
- Cuarta generación. Son los lenguajes capaces de generar código por si solos, son los llamados RAD, con lo cuales se pueden realizar aplicaciones sin ser un experto en el lenguaje. Aquí también se encuentran los lenguajes orientados a objetos, haciendo posible la reutilización de partes del código para otros programas. Ej. Visual, Natural Adabas...

Código fuente

El código fuente es el conjunto de instrucciones que conforman un programa (o subprograma o módulo). El código fuente debe ser compilado para poder ser interpretado y ejecutado por la computadora.

Código objeto

El código objeto es aquel programa que resulta de la traducción del lenguaje fuente (lenguaje entendible por el ser humano) a un lenguaje máquina, es decir a aquel que es inteligible por la computadora.

Compilación

La compilación traduce el código fuente (que depende del lenguaje de programación) a un lenguaje máquina (que depende del sistema de la máquina).

Existen lenguajes que son directamente ejecutados por un intérprete y no necesitan compilación. Este tipo de lenguajes de programación, no requieren un código a ser compilado, ya que consisten en scripts que son interpretados en tiempo real por un intérprete.

Por lo general, los lenguajes interpretados son de alto nivel y están orientados a objetos y eventos, lo que facilita la programación web y la programación cliente/servidor, por lo cual, actualmente son lenguajes con mucho auge en el ámbito informático.

En cuanto a los compiladores, cabe aclarar que no realizan la traducción directa para que pueda ser ejecutable por la computadora sino que debe utilizar también un programa montador o enlazador, conocido como **linker**, que es el que realiza la acción final que permite que el programa objeto pueda ser ejecutado.

Un **linker** (enlazador) es un programa que toma los objetos generados en los primeros pasos del proceso de compilación, la información de todos los recursos necesarios (bibliotecas), quita aquellos recursos que no necesita, y enlaza el código objeto con su(s) biblioteca(s) con lo que finalmente produce un fichero ejecutable o una biblioteca.

En el caso de los programas enlazados dinámicamente, el enlace entre el programa ejecutable y las bibliotecas se realiza en tiempo de carga o ejecución del programa.

Ejemplos de lenguajes de programación:

Lenguaje	Principal área de aplicación	Compilado/interpretado
ADA	Tiempo real	Lenguaje compilado
BASIC	Programación para fines educativos	Lenguaje interpretado
C	Programación de sistema	Lenguaje compilado
C++	Programación de sistema orientado a objeto	Lenguaje compilado
Cobol	Administración	Lenguaje compilado
Fortran	Cálculo	Lenguaje compilado
Java	Programación orientada a Internet	Lenguaje intermediario
MATLAB	Cálculos matemáticos	Lenguaje interpretado
Cálculos matemáticos	Cálculos matemáticos	Lenguaje interpretado
LISP	Inteligencia artificial	Lenguaje intermediario
Pascal	Educación	Lenguaje compilado
PHP	Desarrollo de sitios web dinámicos	Lenguaje interpretado
Inteligencia artificial	Inteligencia artificial	Lenguaje interpretado
Perl	Procesamiento de cadenas de caracteres	Lenguaje interpretado

Existen varias clases de programación, dependiendo de los métodos utilizados y las técnicas empleadas.

En la mayoría de los casos, las técnicas se centran en programación modular y programación estructurada, pero existen otros tipos de programación.

Programación estructurada

La programación estructurada utiliza un número limitado de estructuras de control, reduciendo así considerablemente los errores.

Las principales ventajas de la programación estructurada son:

- Los programas son mas fáciles de entender
- Se reduce la complejidad de las pruebas
- Aumenta la productividad del programador
- Los programas queden mejor documentados internamente.

Un programa esta estructurado si posee un único punto de entrada y sólo uno de salida, existen de "1 a n" caminos desde el principio hasta el fin del programa y por último, que todas las instrucciones son ejecutables sin que aparezcan bucles infinitos.

Programación modular

En la programación modular consta de varias secciones dividas de forma que interactúan a través de llamadas a procedimientos, que integran el programa en su totalidad.

En la programación modular, el programa principal coordina las llamadas a los módulos secundarios y pasa los datos necesarios en forma de parámetros.

A su vez cada modulo puede contener sus propios datos y llamar a otros módulos o funciones.

Programación orientada a objetos (POO)

Se trata de una técnica que aumenta considerablemente la velocidad de desarrollo de los programas gracias a la reutilización de los objetos.

El elemento principal de la programación orientada a objetos es el objeto.

El objeto es un conjunto complejo de datos y programas que poseen estructura y forman parte de una organización.

Un objeto contiene varios datos bien estructurados y pueden ser visibles o no dependiendo del programador y las acciones del programa en ese momento.

Programación concurrente

Este tipo de programación se utiliza cuando tenemos que realizar varias acciones a la vez.

Se suele utilizar para controlar los accesos de usuarios y programas a un recurso de forma simultanea.

Se trata de una programación más lenta y laboriosa, obteniendo unos resultados lentos en las acciones.

Programación funcional

Se caracteriza principalmente por permitir declarar y llamar a funciones dentro de otras funciones.

Programación lógica

Se suele utilizar en la inteligencia artificial y pequeños programas infantiles. Se trata de una programación basada en el cálculo de predicados (una teoría matemática que permite lograr que un ordenador basándose en hecho y reglas lógicas, pueda dar soluciones inteligentes).

Podemos generalizar y decir que hay dos tipos de programación:

- Programación tradicional, modular o estructurada
- Programación orientada a objetos

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

En programación orientada a objetos un programa es considerado como un sistema de objetos interactuando entre sí, ambientes de desarrollo visuales facilitan aun más la construcción de programas y solución de problemas, porque permiten abstraer al ingeniero de software de todo el GUI (interfase gráfica) del problema, que constituye más del 60% del código normal de un programa. Es decir, en programación modular o estructurada un problema sencillo de información es descompuesto en una serie de módulos (llamados procedimientos o funciones) donde cada uno de ellos realiza una tarea específica, por ejemplo uno de ellos captura los datos, otro resuelve operaciones, etc.

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

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

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

Para propósitos de aprendizaje usaremos el enfoque, de programación en ambientes visuales y usando el lenguaje de programación C++Builder.
--

Proceso de creación de un programa:

Desde que decidimos realizar un programa hasta su puesta final en funcionamiento, existirán siempre los siguientes pasos que son ineludibles.

- El desarrollo lógico del programa para resolver un problema en particular. Acá interpretamos el enunciado del problema y pensamos una lógica para su solución.
- Escritura de la lógica del programa empleando un lenguaje de programación específico (codificación del programa).
- Compilación del programa hasta convertirlo en lenguaje de máquina.
- Prueba y depuración del programa.
- Desarrollo de la documentación.

Concentrémonos y analicemos un poco mas el paso primero.

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

Cualquier problema al cual buscamos su solución mediante la creación de un programa, tendrá las siguientes partes:

LA PRIMERA PARTE son todos los datos que el computador ocupa para resolver el problema, estos datos son almacenados internamente en la memoria del computador en las llamadas variables de entrada.

LA SEGUNDA PARTE son todas las operaciones generalmente algebraicas necesarias para solucionar el problema, generalmente esta parte del modelo es una formula (o igualdad matemática, ej. $X = y + 5$).

LA TERCERA PARTE es el resultado o solución del problema que generalmente se obtiene de la parte de operaciones del modelo y dichos datos están almacenados en las llamadas variables de salida.

LA CUARTA PARTE es decidir que información o variables de salida se van a desplegar en pantalla para responder al problema planteado originalmente.

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

Información son datos ya procesados que resuelven un problema planteado.

Nota:

Existirán problemas sencillos donde no se ocupan entradas o no se ocupan operaciones, pero todos ocupan salida.

Una formula grande o muy compleja puede ser más segura y fácil de resolver, si es descompuesta y resuelta en partes, juntando al final los parciales para obtener el resultado final.

Un problema puede tener más de una solución correcta.

Terminado este primer paso, los siguientes pasos implicaran conocer el lenguaje de programación a usar. A esto nos dedicaremos de aquí en adelante.

C++BUILDER - ELEMENTOS BÁSICOS

VARIABLES

Identificadores son conjuntos de letras y/o números que se utilizan para simbolizar todos los elementos que en un programa, son definibles por el programador o ingeniero de software, como son las variables donde se almacenan datos, funciones (pequeños módulos con código), etiquetas, clases, objetos, etc.

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

Una variable se define como un identificador que se utiliza para almacenar todos los datos generados durante la ejecución de un programa. Existen ciertas reglas en cuanto a variables:

- Claras y con referencia directa al problema.
- Sin espacios en blanco, ni símbolos extraños en ellas.
- Se pueden usar abreviaturas, pero solo de carácter general.
- No deben ser palabras reservadas del lenguaje.

Ejemplos de buenas variables: Nombre, Edad, SdoDiario, IngMensual, Perímetro, Calif1, etc.

TIPOS DE DATOS

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

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

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

Tipos de datos numéricos

Tipo	Tamaño	Rango	Utilidad
unsigned char	8	$0 \leq X \leq 255$	Números pequeños y valores de la tabla de caracteres de la PC.
char	8	$-128 \leq X \leq 127$	Números muy pequeños y caracteres de la tabla ASCII
short int	16	$-32,768 \leq X \leq 32,767$	Contadores y números pequeños
unsigned int	32	$0 \leq X \leq 4,294,967,295$	Números grandes y ciclos
int	32	$-2,147,483,648 \leq X \leq 2,147,483,647$	Contadores, números pequeños
unsigned long	32	$0 \leq X \leq 4,294,967,295$	Enteros positivos muy grandes, distancias
enum	32	$-2,147,483,648 \leq X \leq 2,147,483,647$	Conjuntos de valores ordenados
long	32	$-2,147,483,648 \leq X \leq 2,147,483,647$	Números grandes
float	32	$1.18 \cdot 10^{-38} < X < 3.40 \cdot 10^{38}$	Cálculos (7-dígitos)
double	64	$2.23 \cdot 10^{-308} < X < 1.79 \cdot 10^{308}$	Cálculos (15-dígitos)
long double	80	$3.37 \cdot 10^{-4932} < X < 1.18 \cdot 10^{4932}$	Financieros (18-dígitos)

Tipos de datos de cadenas de caracteres

Tipo	Longitud máxima	Memoria requerida	Usado para
ShortString	255 caracteres	2 a 256 bytes	Compatibilidad con versiones anteriores.
AnsiString	$\sim 2^{31}$ caracteres	4 bytes a 2GB	Cadenas de caracteres de 8-bit (ANSI)

WideString	~2^30 caracteres	4 bytes a 2GB	Caracteres unicode; servidores multi-usuario y aplicaciones multi-idioma.
------------	------------------	---------------	---

Tipos de datos Boolean

Las variables de este tipo pueden almacenar los valores: **true** y **false**.

Tipos de datos definidos por el usuario en

Apuntadores a objetos y apuntadores a funciones.
Tipos de datos enumerados: enum días {lunes, martes, miercoles, jueves, viernes, sabado, domingo};

OPERADORES ARITMETICOS

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

C++Builder reconoce los siguientes operadores aritméticos:

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

Todos pueden hacer operaciones con enteros y/o flotantes, excepto %, el cual solo acepta enteros.

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

Para resolver los problemas de potencias y raíces, se usan ciertas instrucciones especiales que proporciona el lenguaje, llamadas funciones matemáticas, en C++ existe toda una librería de instrucciones o funciones matemáticas.

Las funciones reciben uno o más datos o valores y regresan siempre un resultado, una de estas funciones matemáticas es:

pow() **Potencia**

Ej:

$y = 3 * x^{1/2}$ será **pow(double(x),double(1/2.0))**

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

JERARQUIA DE OPERACIONES

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

Si se quiere alterar el orden normal de las operaciones, entonces se debe usar paréntesis. Tampoco es bueno usar paréntesis de mas en una operación, esto solo indica que no se evaluó bien la fórmula.

OPERADORES RELACIONALES

<	Menor
<=	Menor e igual
>	Mayor
>=	Mayor e igual
==	Igual
!=	Distinto

OPERADORES DE OPERACIONES CON BITS

<<	Corrimiento a la izquierda.
>>	Corrimiento a la derecha.
&	And
	Or
^	Xor

OPERADORES LÓGICOS

&&	And
	Or
!	Not

OPERADORES DE ASIGNACIÓN

=	Asignación
---	------------

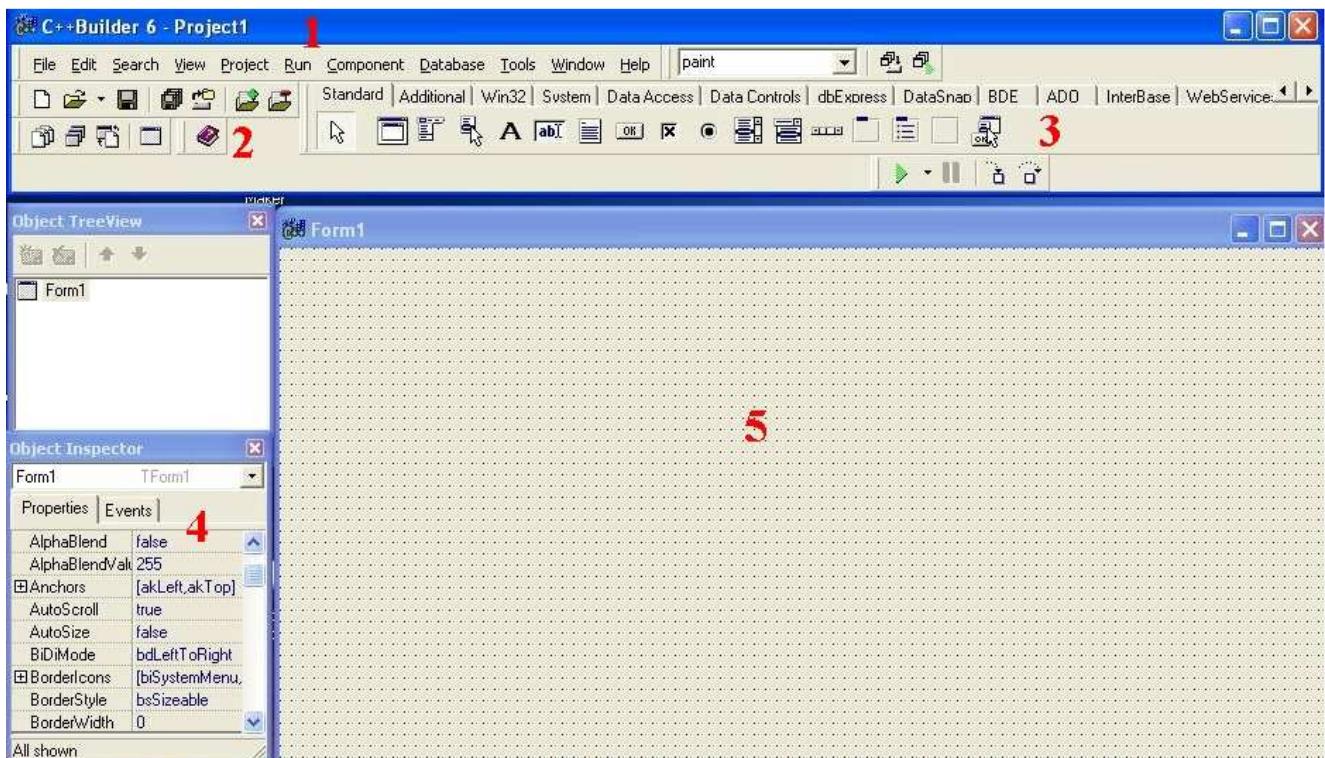
ENTRADAS/SALIDAS EN PROGRAMACION VISUAL

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

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

EL AMBIENTE DE CONSTRUCCIÓN DE PROGRAMAS

La siguiente pantalla es lo que se ve al ejecutar el C++ Builder.



Sus elementos básicos son:

- 1.- La barra de menús (file, edit , etc.);
- 2.- La barra de herramientas (icono de grabar, ejecutar, forma, etc.)
- 3.- La barra de componentes
- 4.- El Inspector de Objetos
- 5.- La forma activa o principal

FORM1 ACTIVA O PRINCIPAL

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

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

Esta forma o ventana es un objeto de C++, y como todos los objetos de C++ y del universo, tiene asociados propiedades y eventos.

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

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

nomobjeto→propiedad = nvovalor;

Ej.; Form1→Color=clRed;

Los objetos son sensibles a eventos.

Eventos, son todos aquellos sucesos de carácter externo que afectan o llaman la atención del objeto.

El objeto que acabamos de conocer es forma o ventana (Form).

En general todo objeto en C++ Builder cumplirá:

- ✓ Debe tener capacidad de detectar el evento

- ✓ Debe tener capacidad de reaccionar y emitir una respuesta, mensaje o conducta apropiada ante el evento detectado.

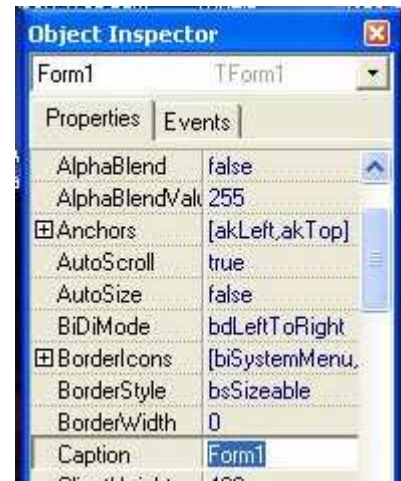
Evento es por ejemplo que se pulse la tecla ESC, o se haga click con el ratón en alguna parte del objeto. O sea esos eventos son disparados con acciones del usuario, o también del sistema operativo mismo.

Si bien el objeto detecta el evento, será el programador mediante instrucciones quien determinará que se hará con la respuesta del objeto.

El Inspector de Objetos es quien contiene todos los posibles eventos asociados al objeto.

En la figura de la derecha podemos ver los eventos asociados al objeto Form1.

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

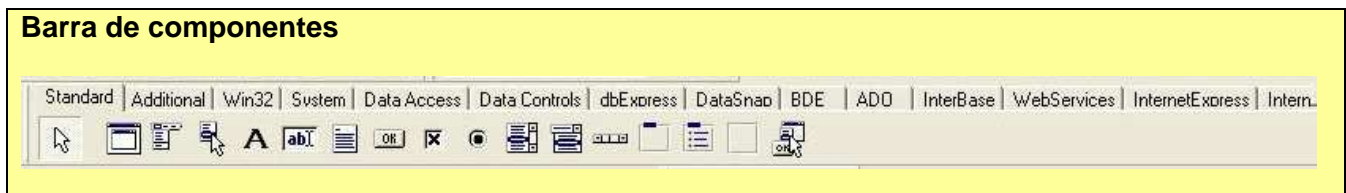


COMPONENTES

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

Es decir toda la interfase que se quiera manejar con el usuario del programa, no consiste más que de una colección de componentes agrupados en una forma o ventana.

La colección de componentes u objetos que pone a nuestra disposición C++Builder están agrupados en diversos folder o pestañas en la barra de componentes.



Cada una de estas pestañas nos dará acceso a componentes que iremos usando en nuestros programas. No vamos a describir en este momento todos los objetos presentes en cada pestaña, ya que los iremos conociendo de a poco.


Para incorporar un componente a una forma solo basta seleccionarlo con un click derecho en su icono y luego colocar el cursor dentro de la forma en el lugar donde se quiere que aparezca y volver a hacer un click derecho.

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

Es el Inspector de Objetos quien permite asociar o modificar propiedades específicas tanto a una forma como a un componente.

Ya en segunda instancia las propiedades de formas y componentes se pueden modificar también dentro del programa, usando instrucciones adecuadas.

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

COMPONENTE LABEL 

Label o etiqueta.

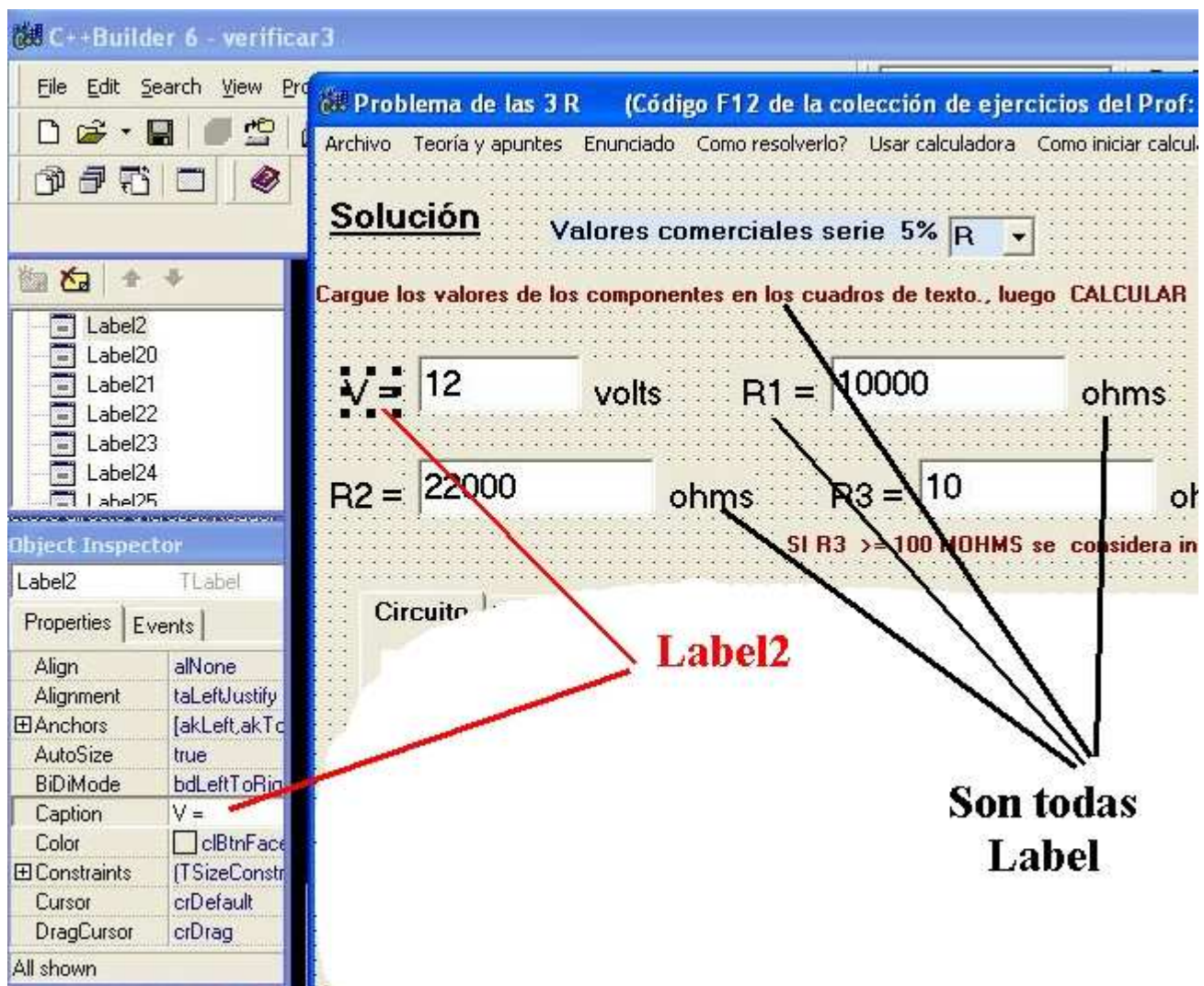
Este componente se utiliza para desplegar textos o mensajes estáticos dentro de las formas, textos tales como encabezados, solicitud al usuario del programa para que proporcione algún dato o información (edad, dame sueldo, etc.), en cierta forma hace las funciones de printf, cout, writeln, print, display, etc., pero solo cuando se consideran en su parte de mensajes.

También es un objeto en C++Builder y por tanto tiene asociados sus propias propiedades y eventos, al mismo tiempo como se está usando dentro del objeto form1, muchas propiedades que se definan para el objeto Form1, el objeto Label1 las va a heredar.

Si bien es cierto que el objeto se llama Label, pero cuando se ponen dentro de una forma C++Builder los va numerando automáticamente, si se ponen tres Labels en Form1, ellos se llaman o simbolizan como Label1, Label2, Label3, etc.

Es la propiedad Caption, la que lleva el contenido del mensaje que se quiere desplegar en la pantalla dentro del Label, para acceder daremos un click derecho a un lado de la propiedad Caption en el inspector de objetos, teniendo seleccionada el objeto Label en la Form.

Ejemplo: Dentro del programa Verificar3



COMPONENTE BUTTON 

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

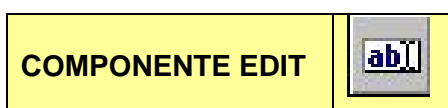
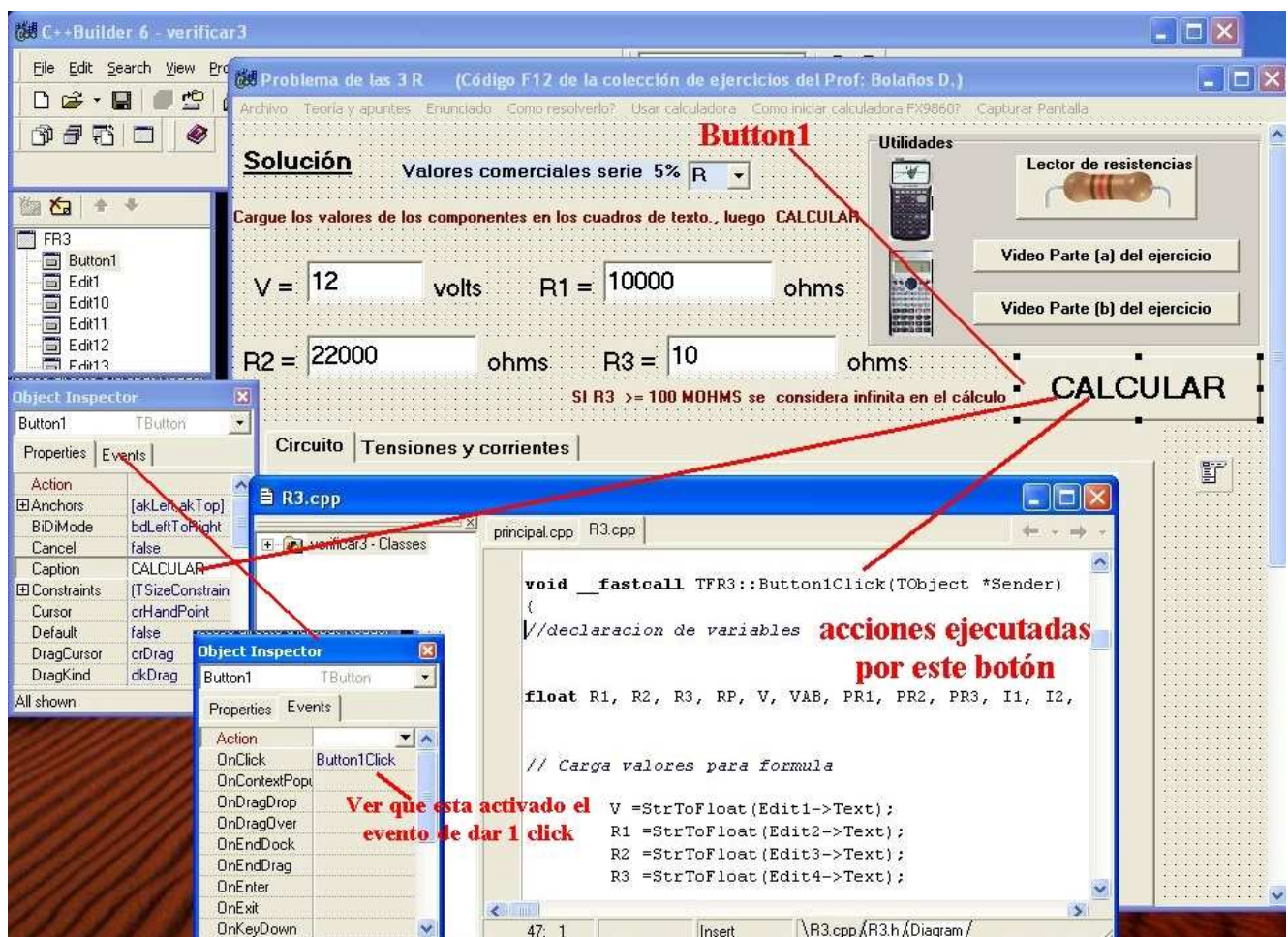
De este componente se maneja su propiedad Caption para etiquetarlo con la palabra "OK" o "ACEPTAR", "CALCULAR", etc, y su evento OnClick para activarlo, es en dicho evento donde se construye el código del programa.

Aunque no es un componente necesario en los programas, ya que el código se puede asociar o pegar a cualquier evento de cualquier forma, o componente del programa, Microsoft ya nos acostumbra a todos los usuarios a que un botón es el que ejecuta acciones.

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

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

Ejemplo: Mostraremos en la figura siguiente el botón CALCULAR en el Verificar3



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

Este componente Edit, es el equivalente a las variables en cualquier lenguaje de programación, mas la instrucción de captura de datos o despliegue de resultados correspondiente.

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

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

Edit2->Text = 5;

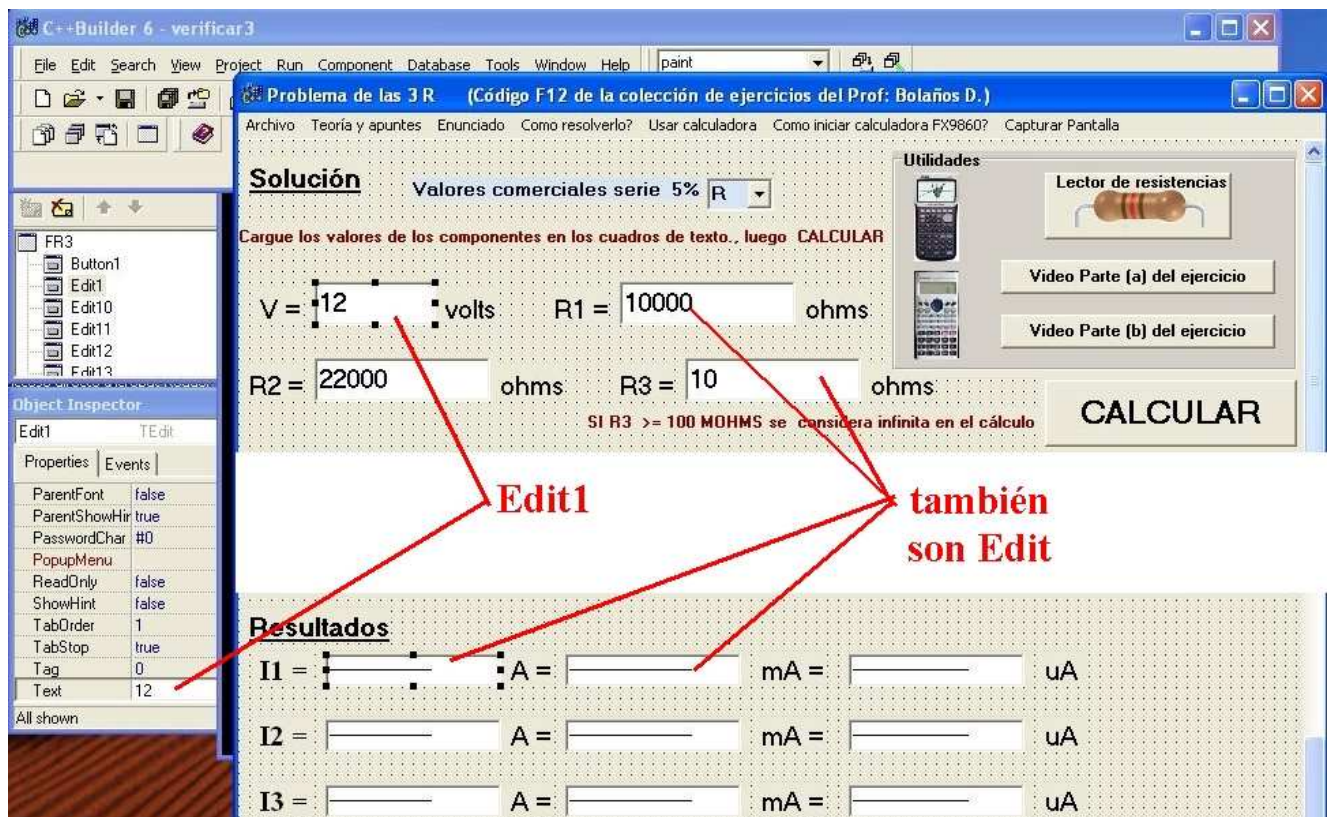
Label3->Caption = "PATO";

Edit4->Text = 3 * 6.2 ;

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

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

Ejemplo: Mostraremos en la figura siguiente algunos Edit en el Verificar3



Importantísimo: Como se mencionó el Edit toma lo que escribamos en el en forma de texto, por lo tanto, si queremos utilizar esa información como un número en una fórmula, deberemos transformarlo.

Quedara más claro al hacer nuestro primer programa, pero podemos decir que para cargar los contenidos de algunos Edit en variables que van a ser utilizadas en una fórmula, será:

// Ejemplo carga valores para utilizar en una fórmula

```
V =StrToFloat(Edit1->Text);  
RB1 =StrToFloat(Edit2->Text);
```

```
RAJ =StrToFloat(Edit3->Text);  
B1 =StrToFloat(Edit7->Text);  
RE =StrToFloat(Edit5->Text);
```

Así: `V =StrToFloat(Edit1->Text);`

Dice que se cargará el contenido del Edit1 en la variable **V**, pero antes se transformará de String a Float. O sea de cadena de caracteres (String) a número real (Float). El contenido será ahora tratado como número.

De igual manera, a la hora de mostrar resultados:

```
//Muestra resultado en cuadro de texto o Edit  
  
Edit9->Text=IB;  
Edit11->Text=IC;  
Edit12->Text=PTBJ;
```

Así: `Edit9->Text=IB;` Dice que se muestre el valor de IB en el Edit9.

COMPONENTE MainMenu

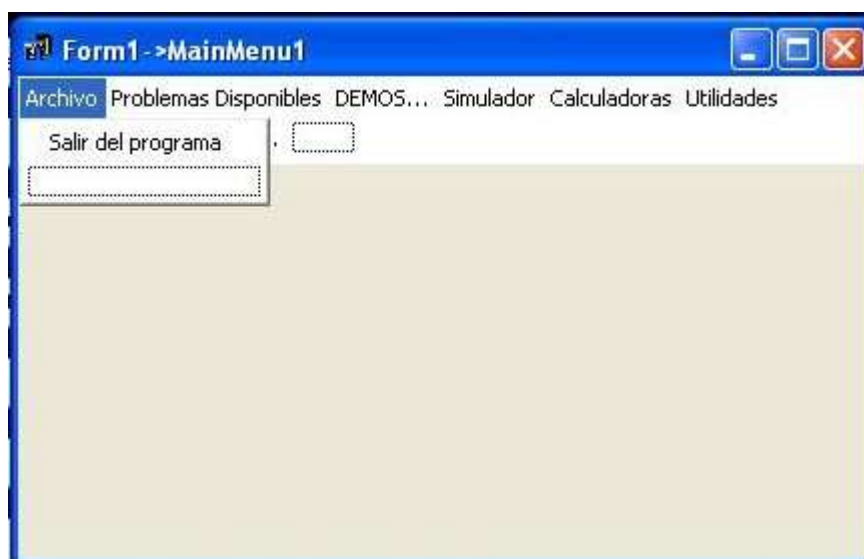


Con este componente MainMenu se forman las barras de menú normales en cualquier programa de Windows (la barra que contiene File, Edit, etc.) junto con sus opciones correspondientes.

Cuando se pone este componente en una forma, se deberá entender que form1 ahora mostrara o desplegará una barra de menú, tal como si fuese una ventana normal de Windows.

Para construir una barra de menú, seguiremos los siguientes pasos:

1. Poner un componente MainMenu en algún lugar de la forma donde no moleste otros objetos (esto es porque este componente queda flotando dentro de Form1 y solo activa o se convierte en barra de menú al momento de ejecución del programa).
2. Para cargarle las opciones solo daremos dobleclick dentro de este componente para que aparezca el siguiente diseñador de menús.



No es una nueva Form, se parece mucho, pero solo se activa cuando se hace un dobleclick en un componente MainMenu.

La barra superior es el menú, cada opción del menú (Archivo, Problemas Disponibles, DEMOS, etc.) pueden tener sus propias subopciones o submenús.

Para escribir las opciones (Archivo, Problemas Disponibles, DEMOS, etc.) primero hacer un click en la parte correspondiente (en el lugar que ocupan o donde se desplegaran) y escribir en la propiedad Caption del Inspector de Objetos la palabra correspondiente.

Se pueden agregar tantos menus y submenus como fueran necesarios en nuestro programa. Es conveniente mantener el orden que estamos acostumbrados en Windows, así el usuario se sentirá más a gusto.

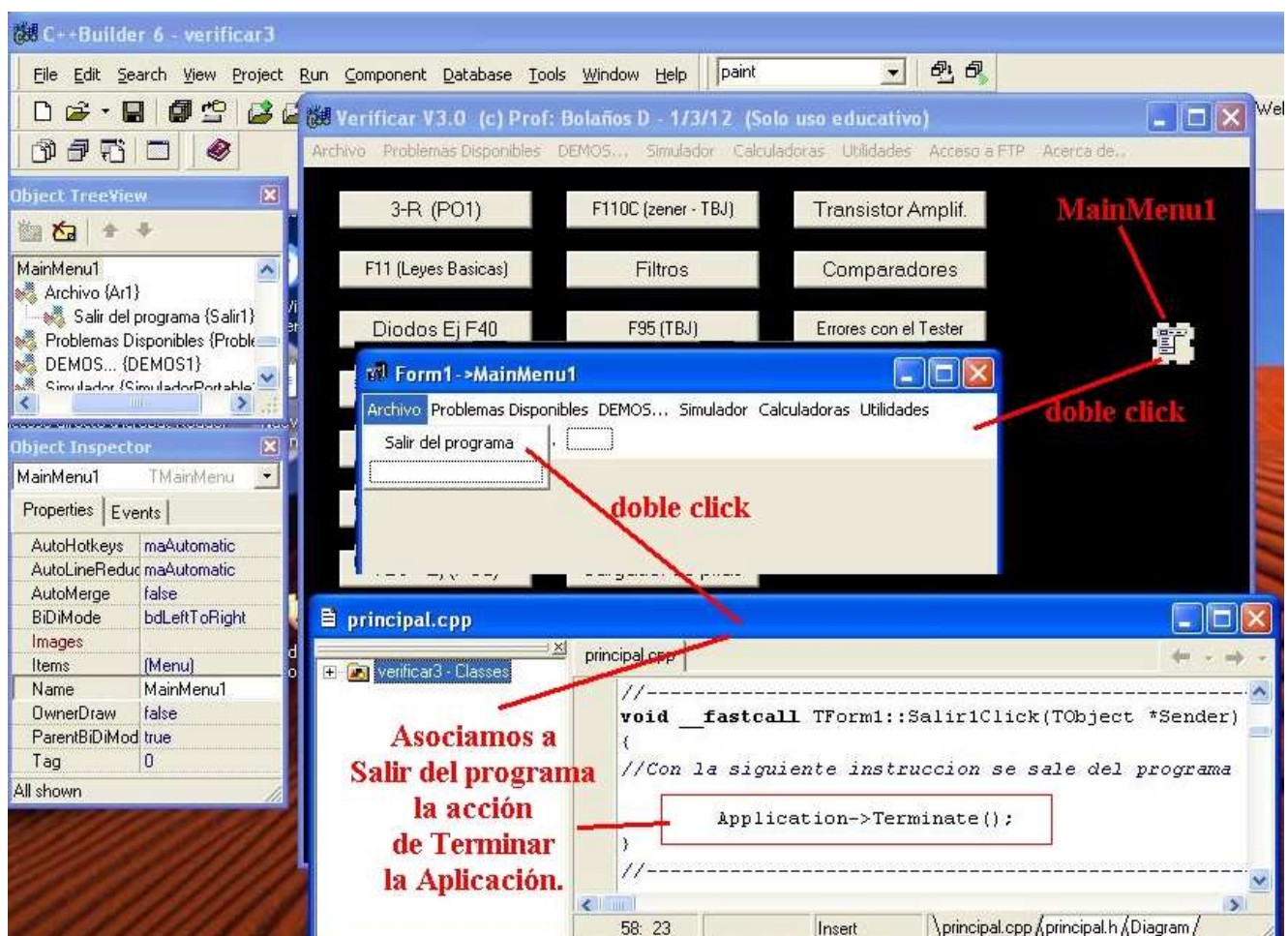
Para marcar una letra de una opción o subopción como **HotKey** solo poner el signo **&** antes de la letra correspondiente.

Ahora falta cargarles el código o programa a ejecutar por y a cada opción o subopcion del menú, recordar que estos eventos también se activan al hacer un click por parte del usuario en cada opción o a seleccionar la primera letra (si se uso el &).

Para cargar código solo debemos dar **dobleclick** en la opción o subopcion correspondiente y ya aparece el editor de programas, listo con el evento **OnClick** de la opción o subopcion correspondiente.

Para terminar y regresar a la forma principal solo click en la X del diseñador de menús.

Ejemplo: Usos de MainMenu en Verificar3



En el ejemplo hemos asociado el ítem del menú "Salir del programa" a la instrucción:

Application->Terminate();

Otras instrucciones que podríamos asociar más frecuentemente en un menú o submenú, pueden ser algunas de las siguientes:

```
//ejecuta un archivo de ayuda  
Application->HelpFile = ("F71HELP.HLP");  
Application->HelpCommand(HELP_CONTENTS, 0);
```

```
//La siguiente instrucción ejecuta una aplicación externa
```

```
ShellExecute(NULL,"open","Vn-570MS.exe",NULL,"",SW_SHOWNORMAL);
```

```
//ejecuta una forma (la llama por el nombre que pusimos en su propiedad NAME)
```

```
F71->Show();
```

```
// Ejecuta un archivo PDF
```

```
ShellExecute(NULL,"open","RESUMENDISIPA.PDF",NULL,"",SW_SHOWNORMAL);
```

```
//Muestra message emergente
```

```
MessageDlg("Visite: www.bolanosdj.com.ar", mtInformation, TMsgDlgButtons() << mbOK, 0);
```

COMPONENTE Memo



El componente Memo sirve para presentar varias y grandes líneas de texto.
Puede ser útil para presentar mensajes.

En el se puede escribir desde nuestro programa o traer la información desde un archivo de texto.

Ejemplo: Uso de Memo en Verificar3

Este es un Memo

Es importante que lea primero y entienda el apunte de POLARIZACIÓN del transistor.

El problema 1 se trata del caso de polarización FIJA.

Trate Ud. de armar las ecuaciones y aplicar todo el procedimiento que allí se menciona.

NO OLVIDE QUE DEBE AGREGARLAS EN SU TRABAJO A PRESENTAR.

Use el CONCEPTUALIZADOR para facilitar su resultado, pero entienda lo que esta haciendo.

Algunas operaciones con Memo:

<code>Memo1->Lines->Clear();</code>	Limpia contenido de un Memo
<code>Memo1->Lines->Add("Esto es lo que mostrará el memo");</code>	Escribir en un Memo
<code>Memo1->Lines->LoadFromFile("escrmemo.txt");</code>	Escribe en un Memo desde un archivo de texto
<code>Memo1->Lines->SaveToFile("Unidad1.txt");</code>	Escribe en un archivo de texto el contenido del Memo.

INSTRUCCIONES DE CONTROL DE PROGRAMA

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

Estas instrucciones se dividen en tres grandes categorías:

- Instrucciones Condicionales que en C++Builder se implementan con las instrucciones **if()** y **switch()**.
- Instrucciones de ciclos con: **for**, **while**, **do while**.
- Instrucción de salto incondicional: **goto**

Tratemos de entender la necesidad de este tipo de instrucciones a la hora de realizar nuestros programas.

Una de las más poderosas características de cualquier computador es la capacidad que tiene de tomar decisiones dentro de opciones pre-establecidas.

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

Todo lenguaje de programación cuenta con instrucciones que permitan establecer condiciones e instrucciones que pueden evaluar esas condiciones.

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

Aclaremos que los lenguajes visuales cuentan con componentes que permiten del mismo modo al usuario tomar decisiones, incluso directamente en pantalla, es decir existen los llamados componentes de selección y decisión.

El formato general de una instrucción condicional es:



Como se observa son cuatro partes bien diferenciadas entre si:

- **La propia instrucción condicional en si.**
- **La condición.**
- **El grupo cierto de instrucciones.**
- **El grupo falso de instrucciones**

Cuando el computador evalúa una condición, el resultado de esa evaluación solo puede dar que la condición es CIERTA o que la condición es FALSA.

Y dependiendo del resultado de la evaluación, se ejecuta las instrucciones del grupo CIERTO o las instrucciones del grupo FALSO

CONDICIONES SIMPLES

En general todas las condiciones simples se forman con;

Variables - Operadores relacionales - Constante o var.

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

Los operadores relacionales que reconoce el lenguaje C++Builder fueron mencionamos en la página 8.

NOTA: Observar y tener cuidado sobre todo con el operador de igualdad '=', y el operador relacional de comparación por igualdad '=='.

Es decir:

sueldo = 500 Se esta pidiendo cargar o asignar la variable sueldo con el valor 500

sueldo == 500 Se esta pidiendo que se compare el valor o dato que ya esta en la variable sueldo, contra el numero 500.

Solo este último formato es valido dentro de una condición, en una instrucción condicional.

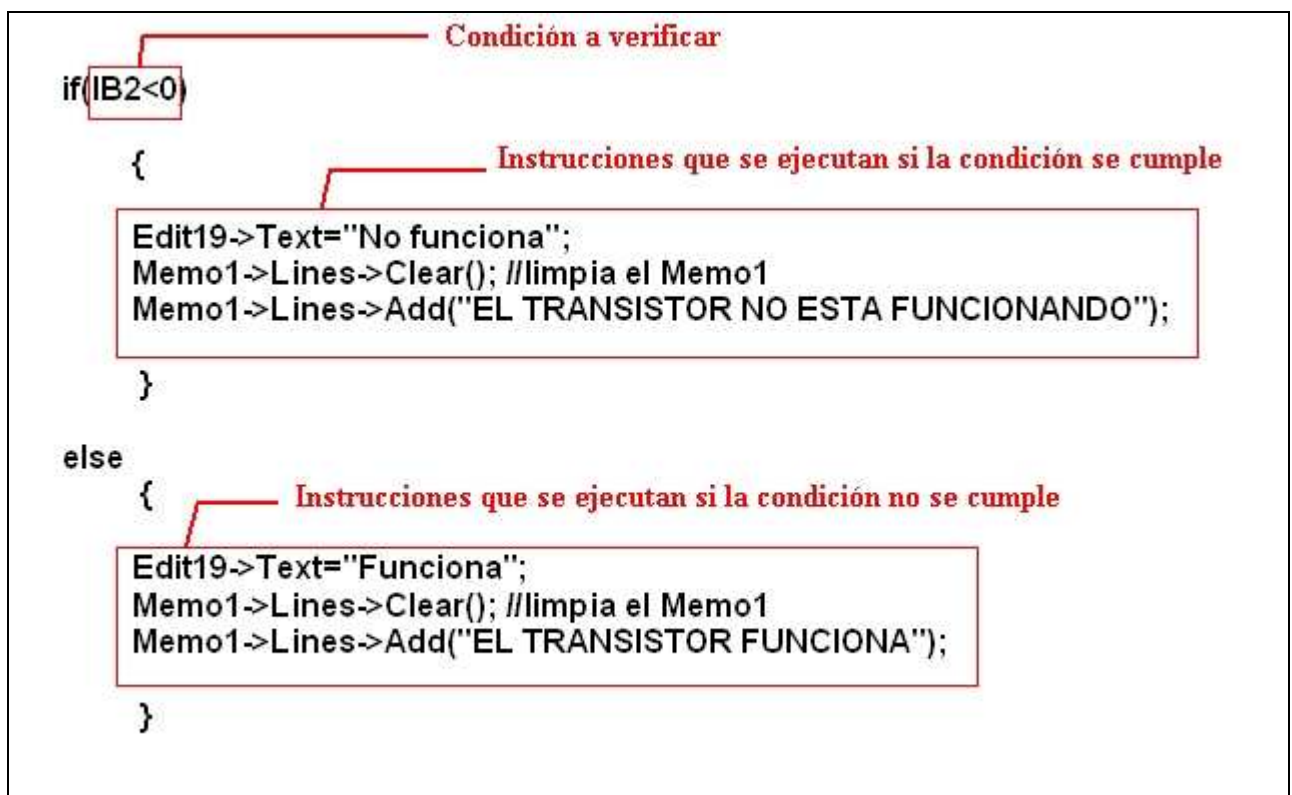
INSTRUCCIÓN IF

Es la instrucción condicional más usada en los diversos lenguajes de programación, su formato completo es:

Ejemplo:

```
if(IB2<0)
{
    Edit19->Text="No funciona";
    Memo1->Lines->Clear(); //limpia el Memo1
    Memo1->Lines->Add("EL TRANSISTOR NO ESTA FUNCIONANDO");
}
else
{
    Edit19->Text="Funciona";
    Memo1->Lines->Clear(); //limpia el Memo1
    Memo1->Lines->Add("EL TRANSISTOR FUNCIONA");
}
```

Describiremos a continuación las partes que integran esta estructura, con el fin de comprender su función y poder utilizarlas.



IMPORTANTE:

- ✓ Observar donde van y donde no van los puntos y comas.
- ✓ La condición va entre paréntesis.
- ✓ Si un if no ocupa un grupo falso de instrucciones, entonces no se pone el else, y la llave antes del else si terminaría con punto y coma.

Ejemplo:

```
if (VMEDIDO >1.999)
```

```
Label32->Caption="ERROR";  
  
else  
  
Label32->Caption=FormatFloat("###0.000",VMEDIDO);
```

Ejemplo:

```
if (pruebaled==1)  
  
Label32->Caption="ERROR";
```

CONDICIONES COMPUESTAS

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

Por ejemplo que el computador muestre en pantalla un listado de alumnos de un colegio que estudien Electrónica y tengan promedio mayor que 7.

Una condición compuesta se define como dos o más condiciones simples unidas por los llamados operadores lógicos. Recordemos que los operadores lógicos son, And Or, Not

Ejemplo:

```
if((VDE=2) && (VT<3))  
  
{  
  
Memo1->Lines->Clear(); //limpia el Memo1  
Memo2->Lines->Clear(); //limpia el Memo2  
Memo1->Lines->Add("LA TENSIÓN NO ES LA DESEADA");  
Memo2->Lines->Add("SELECCIONE OTROS COMPONENTES");  
  
}
```

INSTRUCCION SWITCH

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

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

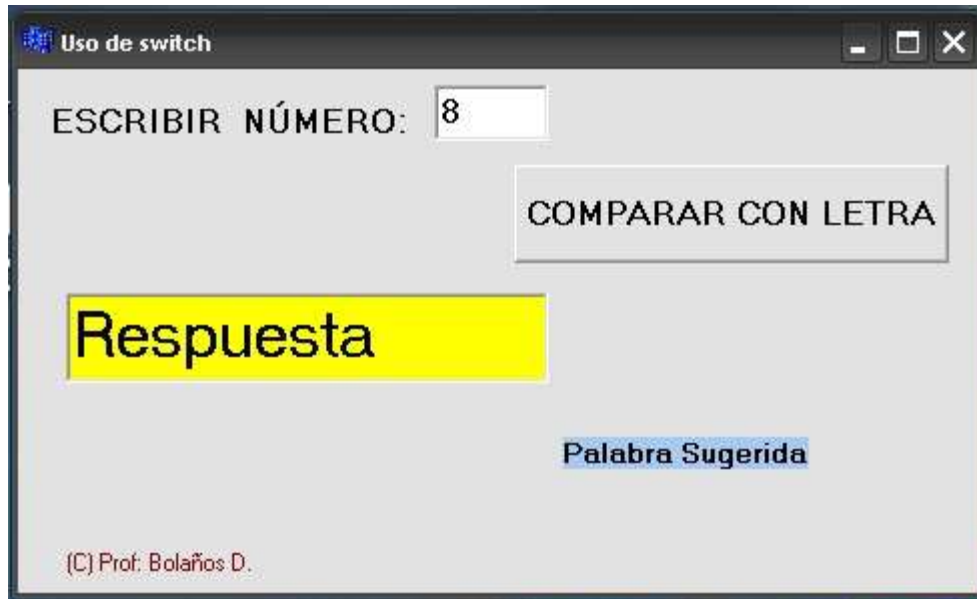
Notas:

Solo se puede usar como variable de condición una variable entera (int) o variable char.

Instrucciones de tipo `switch()` se utilizaban para construir programas de selección de menús, donde al usuario se le planteaban dos o tres problemas distintos y el propio usuario seleccionaba cual de ellos quería ejecutarse.

Ejemplo en el que se usa la instrucción `switch`:

Este programa pide al usuario que digite un número, si el numero está entre 1 y 8 devuelve la letra correspondiente al orden en el abecedario, además de una palabra sugerida que comienza con esa letra.



El código en el botón "COMPARAR CON LETRA" se encuentra el siguiente código:

```
int param0;

param0 =StrToFloat(Edit1->Text);

switch (param0)
{
    case 1:// Escribe letra A
        Edit2->Text="A";
        Label2->Caption="Alarma";
        break;

    case 2://
        Edit2->Text="B";
        Label2->Caption="Bota";
        break;
    case 3://
        Edit2->Text="C";
        Label2->Caption="Casa";
        break;

    case 4://
        Edit2->Text="D";
        Label2->Caption="Dedal";
        break;

    case 5://
        Edit2->Text="E";
        Label2->Caption="Elefante";
        break;

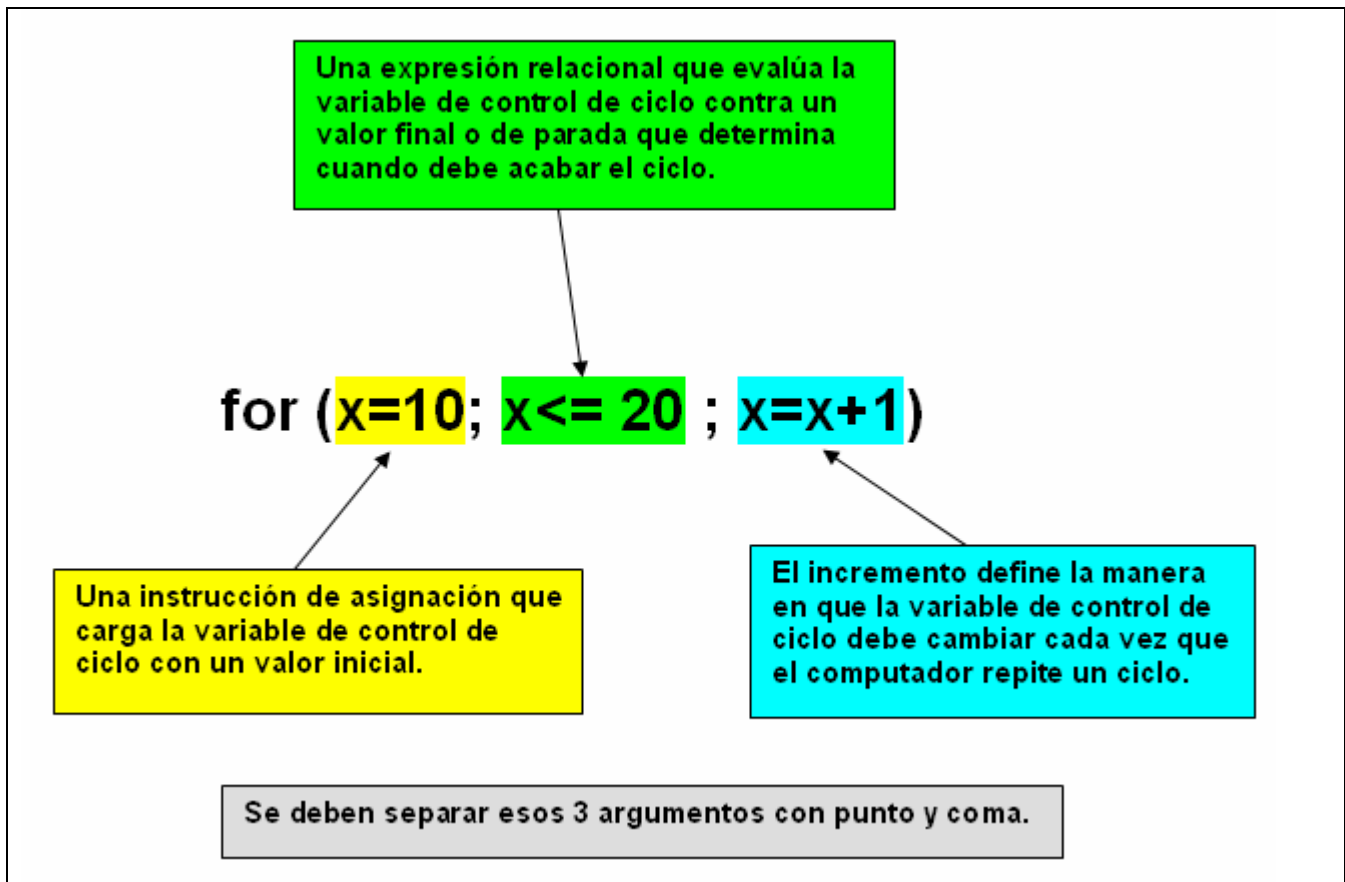
    case 6://
```

```
    Edit2->Text="F";  
    Label2->Caption="Fauna";  
break;  
case 7://  
    Edit2->Text="G";  
    Label2->Caption="Gato";  
break;  
  
case 8://  
    Edit2->Text="H";  
    Label2->Caption="Hola";  
break;  
  
default:  
  
    Edit2->Text="NO PERMITIDO";  
    Label2->Caption="Elegir otra";  
}
```

Ciclo FOR

Este ciclo es uno de los más usados para repetir una secuencia de instrucciones, sobre todo cuando se conoce la cantidad exacta de veces que se quiere que se ejecute una instrucción o conjunto de instrucciones. Resuelve el problema de repetir todo el programa, o cierta parte del programa cuando se debe ejecutar más de una vez.

Ejemplo de las partes del ciclo:



Casos Particulares;

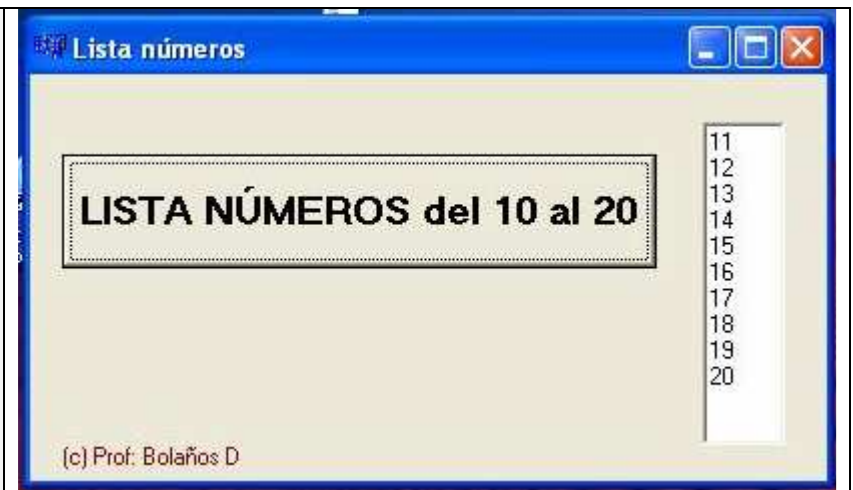
- El ciclo comienza en uno y se incrementa de uno en uno, este es el caso más general.
- Valor inicial puede ser diferente de uno.
- El valor inicial puede ser negativo.
- Los incrementos también pueden ser diferentes al de uno en uno.
- Incluso pueden ser decrementos, solo que en este caso, recordar; el valor inicial de la variable debe ser mayor que el valor final.

Solo para los casos de incrementos y decrementos de uno en uno es posible substituir:

- $x = x + 1$ por $x++$
- $x = x - 1$ por $x--$

Ejemplo:

El siguiente programa genera un listado de números de 11 al 20 y los muestra en un Memo al presionar el botón.

<pre>int n; Memo1->Lines->Clear(); for (n=10; n<= 20 ; n=n+1) { //aquí el listado de instrucciones Memo1->Lines->Add(n); }</pre>	
---	--

Ciclo WHILE

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

Su formato general es:

```
Cargar o inicializar variable de condición.  
while (condición)  
{  
Grupo cierto de instrucciones;  
Instrucción(es) para salir del ciclo;  
}
```


While puede llevar dos condiciones, en este caso inicializar 2 variables de condición y cuidar que existan 2 de rompimiento de ciclo.

El grupo cierto de instrucciones puede ser una sola instrucción o todo un grupo de instrucciones.

La condición puede ser simple o compuesta.

A este ciclo también se le conoce también como ciclo de condición de entrada prueba por arriba, porque este ciclo evalúa primero la condición y posteriormente ejecuta las instrucciones.

El siguiente programa es un ejemplo donde se usa while. Se listan números en forma decreciente mediante la resta a partir de un número (Final) hasta otro número (Inicial). Se efectúa la resta sucesiva dentro de un while. La acción se ejecuta la presionar el botón.

<pre>//Uso de while int I,F,D,C; I =StrToInt(Edit1->Text); F =StrToInt(Edit2->Text); D =StrToInt(Edit3->Text); C =F; Memo1->Lines->Clear(); while (C>=I) { Memo1->Lines->Add(C); C=C-D; } Memo1->Lines->Add("FIN"); Memo1->Color=clYellow;</pre>	
--	--

Ciclo DO WHILE

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

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

Su formato general es:


```
Cargar o inicializar variable de condición.

do
{
Grupo de instrucciones;
Instrucción(es) para salir del ciclo;
}

while (condición)
```

El siguiente programa es un ejemplo donde se usa do while. Se listan números en forma creciente hasta un número (FINAL).

Se efectúa la suma sucesiva dentro de un do while. La acción se ejecuta la presionar el botón.

<pre>int cuenta,final; cuenta = 0; Memo1->Lines->Clear(); final =StrToInt(Edit1->Text); do { cuenta = cuenta + 1; Memo1->Lines->Add(cuenta); } while (cuenta < final);</pre>	
--	--

Fin de la Parte 1