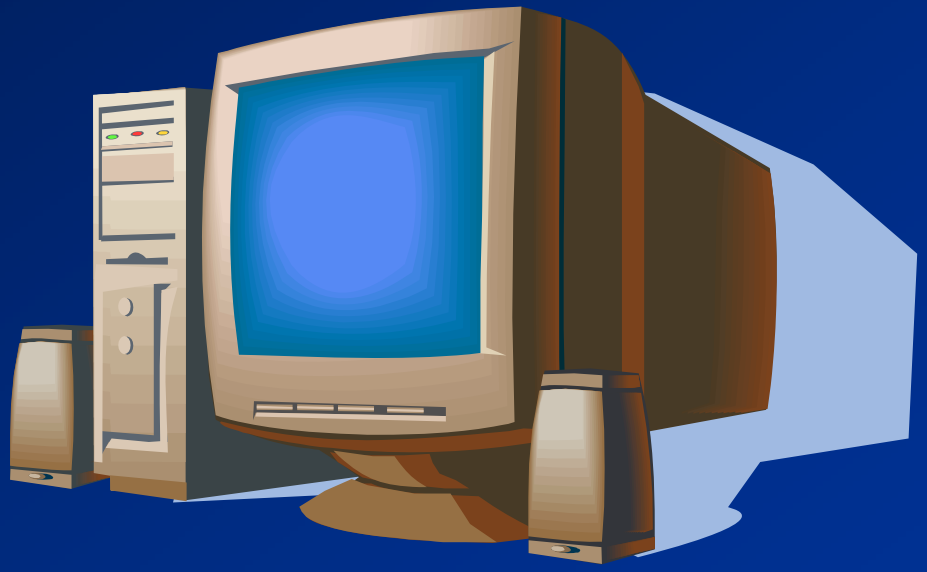
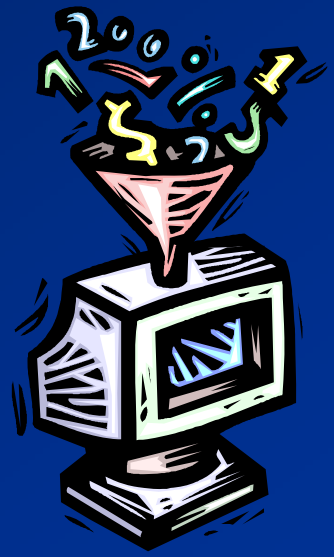
Generalidades del Computador Y Resolución de Problemas

REALIZADO POR:
ING. BELZYT GONZÁLEZ GUERRA



Departamento de Investigación de Operaciones y Computación

Generalidades del Computador Y Resolución de Problemas



Computador:

Es un manipulador de símbolos. Un procesador de los datos.

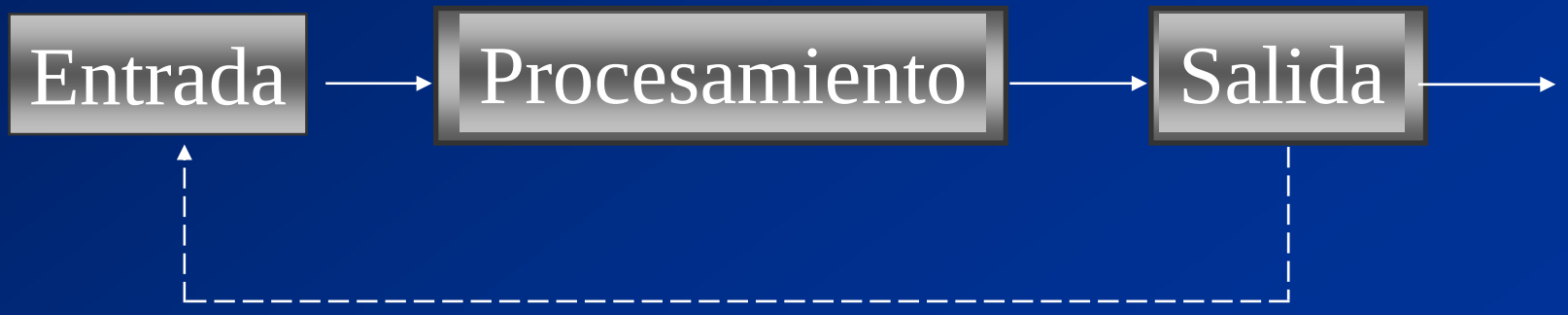
Procesamiento de Datos:

Manejo de los datos para sacarle utilidad o producir información.

Generalidades del Computador Y Resolución de Problemas

Ciclo de Procesamiento de datos

Existen tres pasos fundamentales:



Generalidades del Computador Y Resolución de Problemas

Esquema General Funcional del Computador

Unidades de Almacenamiento Secundario

Unidad Central de Procesamiento (CPU)

Unidades de Entrada

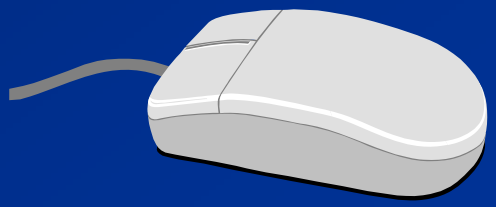
Unidades de Salida

Generalidades del Computador Y Resolución de Problemas

Esquema General Funcional del Computador

CPU

- Unidad de Almacenamiento Principal
- Unidad Arimético - Lógica
- Unidad de Control

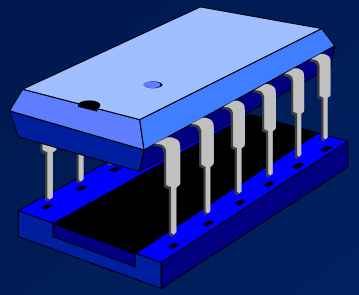


Dispositivos Periféricos

Todos aquellos conectados al CPU.

- Und. Entrada / Salida
- Und. Almacenamiento Secundario

Generalidades del Computador Y Resolución de Problemas



Unidad de Almacenamiento Principal ó Memoria Principal

Memoria RAM

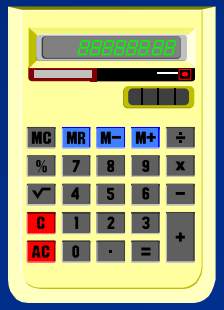
De Random Access Memory. Donde se guardan los datos del computador. La información se pierde al apagar el equipo.

Memoria ROM

De Read Only Memory.

Unidad Aritmético-Lógica

Es la encargada de realizar todas las operaciones aritméticas y lógicas necesarias para el procesamiento de los datos.

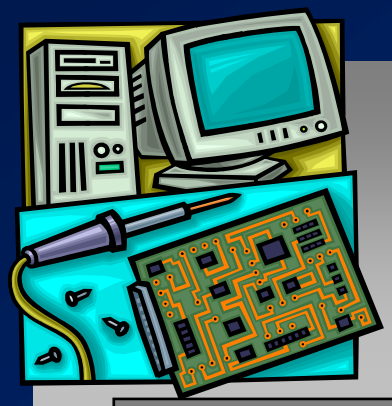


Generalidades del Computador Y Resolución de Problemas

Unidad de Control

Es la que “manda”. Está encargada de controlar y coordinar el funcionamiento del sistema, constituido por las demás unidades y periféricos.

Generalidades del Computador Y Resolución de Problemas



Generalidades del Computador Y Resolución de Problemas

La salida no es otra cosa que los resultados de procesar los datos según un programa.

Generalidades del Computador Y Resolución de Problemas

El Software puede ser dividido en:

- Sistemas Operativos
- **Lenguajes**
- Programas Utilitarios:
  - Procesadores de Texto
  - Hojas de Cálculo
  - Dibujo y Diseño
  - Mantenimiento
  - Entretenimiento
  - Tratamiento de Sonido e Imágenes

Generalidades del Computador Y Resolución de Problemas

A su vez en los lenguajes se tiene:

Assembler	
Basic	Visual Basic
Fortran	Visual Fortran
Turbo Pascal	Delphy
Borland C	**Builder C**
Java	Builder Java

Generalidades del Computador Y Resolución de Problemas

Un lenguaje de programación permite implementar los algoritmos en un computador, es un lenguaje escrito que posee un alfabeto y reglas sintácticas precisas.

Se suele decir que un algoritmo es una lista de pasos que ejecutados secuencialmente, resuelven un problema.

Generalidades del Computador Y Resolución de Problemas

Resolución de Problemas:
(con el Computador)

- Comprensión del Problema
- Desarrollo de un método de solución
- Codificación y Comprobación
- Ejecución y Depuración del Programa
- Validación de Resultados

Generalidades del Computador Y Resolución de Problemas

Análisis de Problemas:

- Las Incógnitas
- Los Datos
- Las Condiciones
- Información Adicional

Comprensión del problema

Generalidades del Computador Y Resolución de Problemas

Desarrollo de un método de solución

Procedimiento

Lista de pasos o instrucciones que permiten realizar una actividad cualquiera.

Paso Secuencial

Cada uno de los pasos que constituyen un procedimiento y que deben ejecutarse siguiendo un orden específico.

Generalidades del Computador Y Resolución de Problemas

Para que un procedimiento pueda ser considerado un **ALGORITMO** debe reunir las siguientes características:

- Finitud
- Buena Definición
- Generalidad
- Poseer Entrada
- Generar Salida o Resultado

Generalidades del Computador Y Resolución de Problemas

Un computador sólo “entiende” ciertos lenguajes particulares, conocidos como Lenguajes de Programación, en consecuencia el Algoritmo debe ser traducido a uno de estos lenguajes. Este proceso se conoce como CODIFICACION.

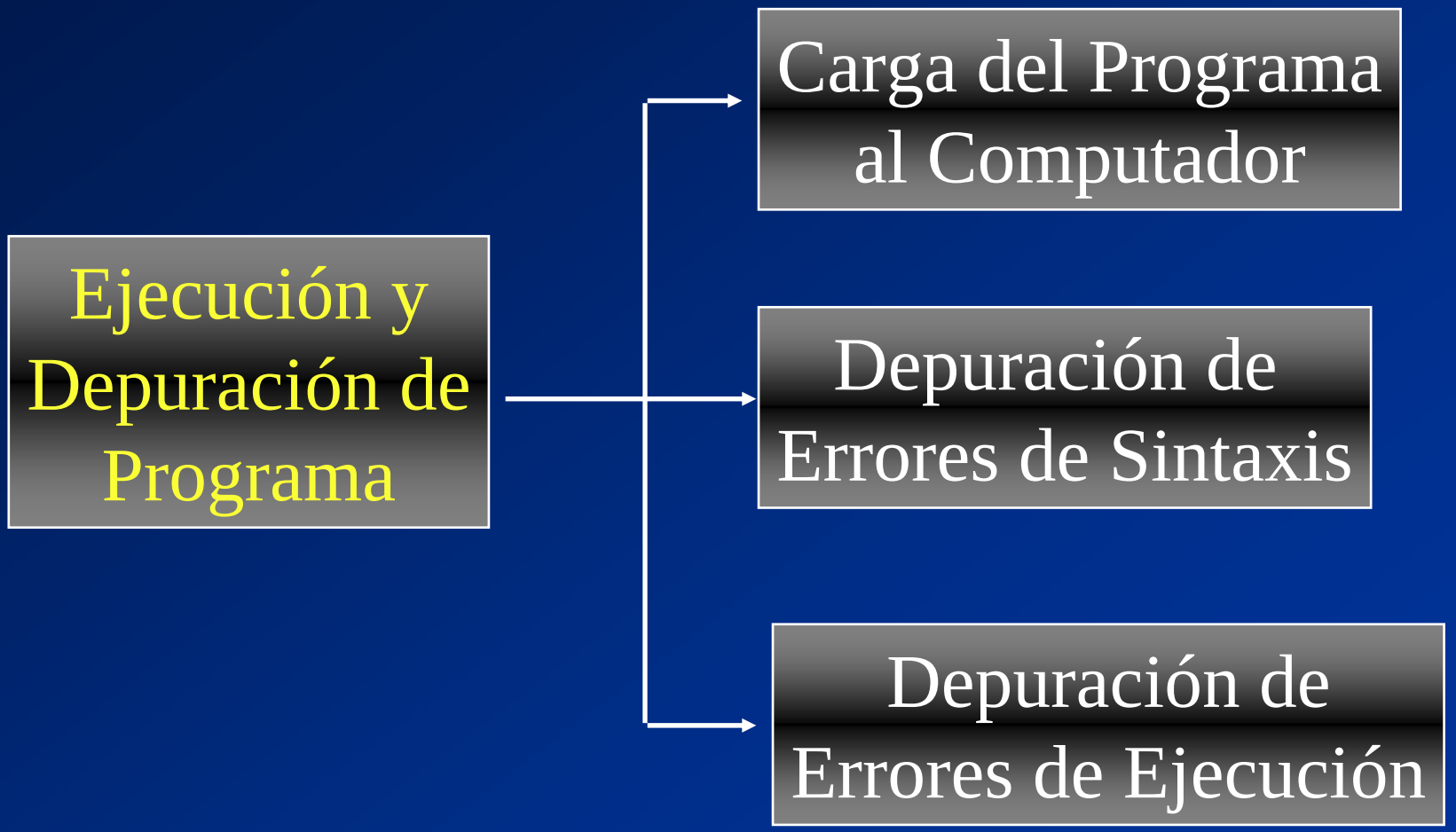
Generalidades del Computador Y Resolución de Problemas

Codificación y Comprobación del Programa

Codificación del Algoritmo Detallado

Revisión del Funcionamiento del Programa

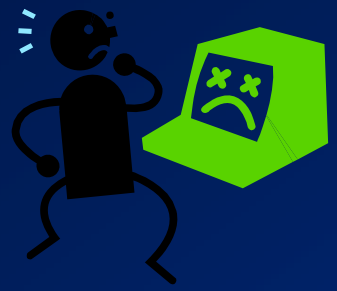
Generalidades del Computador Y Resolución de Problemas



Generalidades del Computador Y Resolución de Problemas

La salida no es otra cosa que los resultados de procesar los datos según un programa.

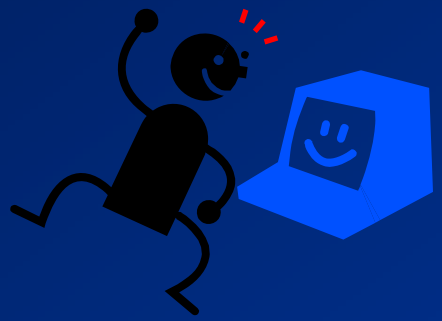
Generalidades del Computador Y Resolución de Problemas



Validación de Resultados

Comparación de los Resultados del Programa con el Modelo

Comprobación de los Resultados del Modelo con el Problema Real



PROGRAMACIÓN – BUILDER

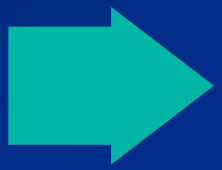
Resolución de Problemas:
(con el Computador)

- Comprensión del Problema
- Desarrollo de un método de solución
- Codificación y Comprobación
- Ejecución y Depuración del Programa
- Validación de Resultados

PROGRAMACIÓN – BUILDER

Análisis de Problemas:

- Las Incógnitas
- Los Datos
- Las Condiciones
- Información Adicional



Comprensión del problema

PROGRAMACIÓN – BUILDER

La salida no es otra cosa que los resultados de procesar los datos según un programa.

PROGRAMACIÓN – BUILDER

Desarrollo de un método de solución

Procedimiento

Lista de pasos o instrucciones que permiten realizar una actividad cualquiera.

Paso Secuencial

Cada uno de los pasos que constituyen un procedimiento y que deben ejecutarse siguiendo un orden específico.

PROGRAMACIÓN – BUILDER

Para que un procedimiento pueda ser considerado un **ALGORITMO** debe reunir las siguientes características:

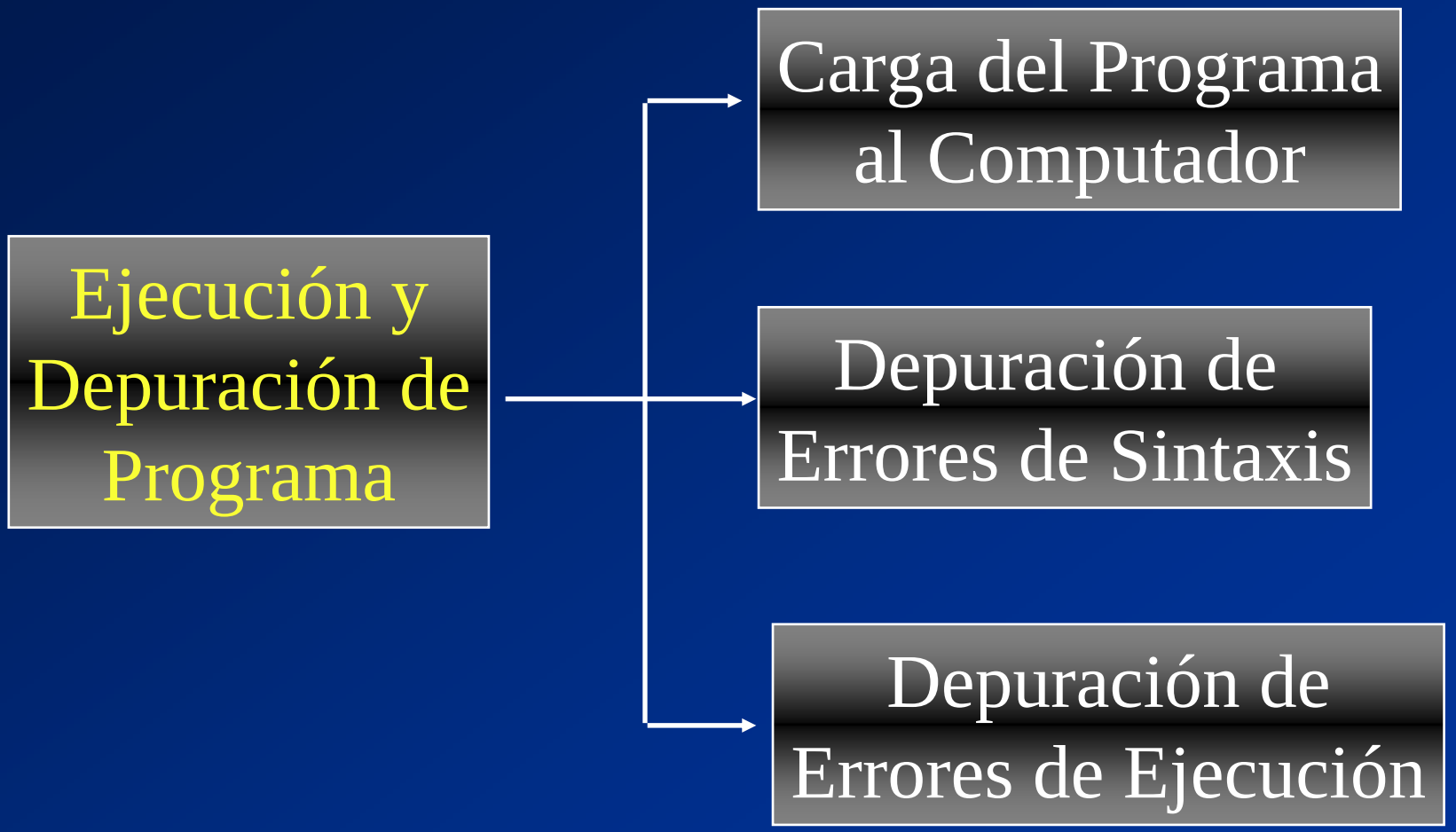
- Finitud
- Buena Definición
- Generalidad
- Poseer Entrada
- Generar Salida o Resultado

PROGRAMACIÓN – BUILDER

Un computador sólo “entiende” ciertos lenguajes particulares, conocidos como Lenguajes de Programación, en consecuencia el Algoritmo debe ser traducido a uno de estos lenguajes. Este proceso se conoce como CODIFICACION.

PROGRAMACIÓN – BUILDER

PROGRAMACIÓN – BUILDER



PROGRAMACIÓN – BUILDER

Ambiente

BUILDER C++

BUILDER C

Es un lenguaje que permite desarrollar programas enfocados a las interfaces gráficas de usuario, o GUI (del inglés, *Graphical User Interface*), en particular dentro del ambiente Windows.

Ofrece al programador la facilidad de disponer de una serie de herramientas que evita tener que estar pendiente de la mayor parte del manejo de los gráficos (dibujo de los controles, evaluación constante del ratón, saltos entre menús y pantallas, etc.).

Objetos

Es una pieza binaria de software que realiza una tarea de programación específica.

Sobre este elemento reposa el funcionamiento de un lenguaje al estilo de BUILDER, no es alcance de este curso programarlos, sólo se usarán.

Propiedades

Todos los Objetos con los que trabajamos en BUILDER tienen una serie de propiedades. Estas propiedades permiten personalizar el objeto indicando su posición, colores, título, etc.

Hay propiedades que sólo pueden ser establecidas o modificadas durante el diseño de la aplicación, otras a las que sólo se puede acceder mientras la aplicación se está ejecutando y un tercer grupo que pueden ser cambiadas en cualquier momento.

Eventos

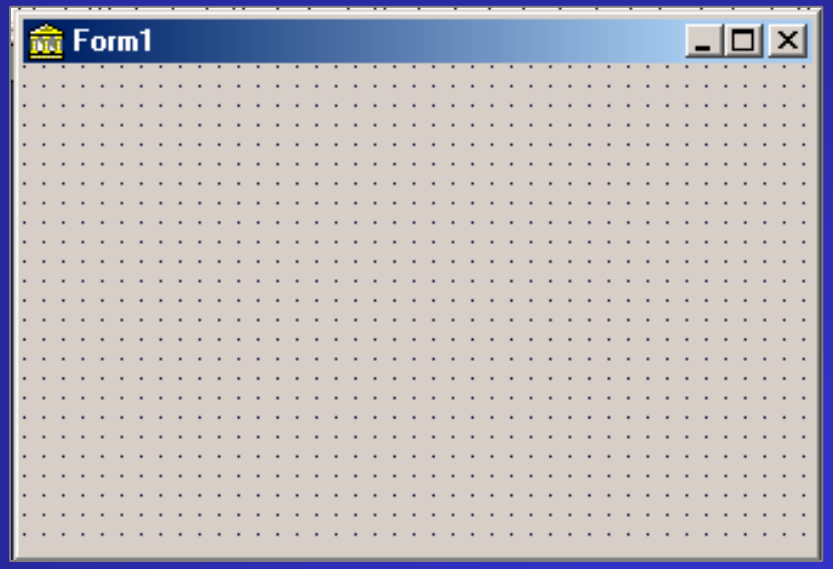
Windows es un entorno gestionado por eventos que son generados por una acción exterior por parte del usuario.

Cada uno de los controles que podemos insertar en un formulario, incluido el mismo formulario, puede recibir una serie de eventos.

Formularios

Es el término dado a una ventana personalizable; al menos deberá existir una en cualquier proyecto, siendo el punto central de cualquier aplicación estándar.

Inicialmente se empleará el Formulario estándar, tal como se muestra en la figura.



Formularios

La lógica asociada a los lenguajes como el BUILDER difiere de las generaciones previas de programación, no existe un "comienzo" como tal y una secuencia de ingreso de un valor (o grupo de ellos) un cálculo y la generación de la salida final.

El enfoque se centra en la ocurrencia de "eventos" tales como presionar el teclado, pasar el cursor sobre un elemento en la pantalla, presionar el botón del cursor, etc, luego, cada evento está asociado a una serie de instrucciones. Como se puede notar esta ocurrencia de eventos puede ser en cualquier secuencia.

El formulario es el elemento donde se asientan los otros elementos que generarán eventos.

Propiedades Claves de Formularios

La clase TFORM tiene muchas propiedades, algunas pocas veces se usan y otras son muy frecuentemente empleadas, primero veamos algunas en tiempo de diseño:

BorderIcons: Controla la aparición de los botones que aparecen en el borde del formulario.

BorderStyle: Indica el tipo de borde que poseerá el formulario, esto también se asocia a si la ventana será dimensionable o no.

Font: No sólo indica el tipo de fuente que tendrá el formulario, si no que esta propiedad se hereda a los componentes que se inserten en el mismo.

Métodos de Formularios

Close : Cierra un formulario.

Print: imprime el contenido de un formulario (sólo el área del cliente).

SetFocus: activa al formulario y lo trae al frente.

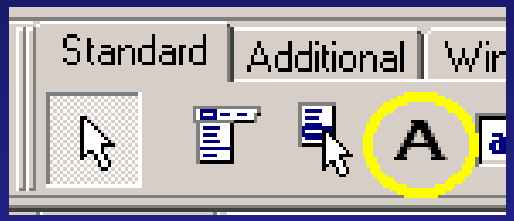
Show y ShowModal: Despliegan el formulario. Show lo hace en forma no modal (lo que permite activarse otros formularios mientras está visible).

Paleta COMPONENT

La paleta ***Component*** se emplea para seleccionar un objeto o un componente a fin de colocarlo sobre un formulario.

Sobre este elemento se ubicaran los diferentes componentes u objetos a emplear. Cuando hay muchos componentes en la pestaña seleccionada se observarán unas flechas que permiten tener acceso a los componentes no visibles.

ETIQUETAS



Componente **Label**

Las *Etiquetas* son controles que permiten colocar texto sobre una forma. En el fondo es un objeto que permite comunicarse de cierta forma con el usuario.

Una *Etiqueta* no puede poseer su propia ventana por lo que no puede recibir datos directamente desde el teclado.

Se ubica en la pestaña ***Standard*** de la barra de componentes.

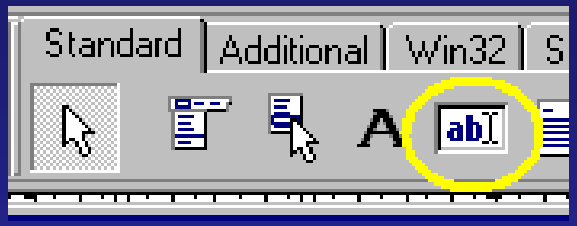
ETIQUETAS

La propiedad más importante es ***Caption*** que determina lo que será mostrado.

El contenido de una Etiqueta puede cambiar durante la ejecución del programa y es difícil prever el tamaño durante el tiempo de diseño; si se quiere que el propio control cambie su tamaño se usa la propiedad ***AutoSize*** en su valor **True**.

En caso de querer distribuir de forma uniforme el texto en varias líneas se coloca en **True** también la propiedad ***WordWrap***.

CONTROLES DE EDICIÓN



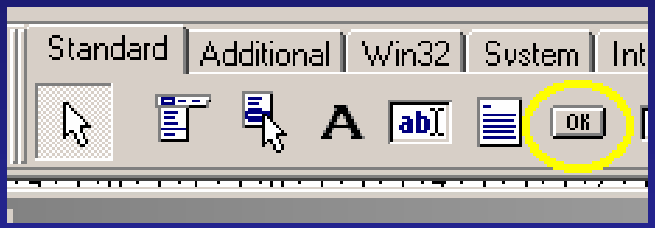
Componente **Edit**

Encapsula el control de edición básico de una línea, el texto sólo se puede alinear a la izquierda.

La propiedad principal es ***Text*** con la cual se obtiene lo escrito por el usuario.

Se ubica en la pestaña ***Standard*** de la barra de componentes.

BOTONES:



Componente Button

Es el más ampliamente usado al ser el botón estándar. Usualmente se emplea para iniciar acciones luego de ser pulsado (por lo que responde al evento ***OnClick***).

Posee una etiqueta que indica cual será su acción (modificable mediante la propiedad ***Caption***).

Se ubica en la pestaña ***Standard*** de la barra de componentes.

Modificando Propiedades

Las propiedades de cualquier objeto pueden ser modificadas en el momento que está funcionando el programa, en lo que se conoce como “tiempo de ejecución”. Para lo anterior se escribe:

Nombre del Control -> Propiedad = Valor

Por ejemplo:

Label1->Caption = “El Resultado es:”;

En este caso se modifica el mensaje que se escribe en una etiqueta y es una forma de presentar resultados.

Para trabajar con un lenguaje como el BUILDER, que funciona dentro del ambiente Windows se debe conocer los siguientes conceptos:

Objeto ó Componente: es una pieza binaria de software independiente, que realiza cierta función específica predefinida.

Propiedad: determina la operación de un componente.

Evento: es algo que ocurre como resultado de la interacción de un componente con el usuario de Windows.

Controlador de Evento: es una sección de código que se invoca en la aplicación en respuesta a un evento.

Cuando se hace doble click sobre un objeto el ambiente BUILDER cambia al editor y de una vez crea el cuerpo de la función que responderá a un evento sobre ese objeto. El evento asociado es el más común sobre el objeto sobre el cual se trabaja, si se desea codificar sentencias asociadas a otro tipo de evento se acude a la ventana del *Inspector de Objetos*, se toma la pestaña de *Eventos* y se hace doble click sobre el evento al que se quiere responder, esto hace que se presente el cuerpo de la función asociada.

OnChange: se activa cuando un control se modifica de una u otra forma. Dependiendo del componente el como se implementa.

OnClick: cuando se hace click con el ratón sobre el componente.

OnDblClick: Se activa cuando se hace doble click.

OnEnter: Ocurre cuando un componente de ventana recibe el enfoque.

OnExit: Ocurre cuando un componente de formulario pierde el enfoque como resultado de pasarse a un control diferente.

OnKeyDown: se activa cuando el usuario presiona una tecla mientras el control tiene el enfoque.

OnKeyPress: igual que el anterior pero sólo cuando las teclas presionadas son alfanuméricas o las teclas: Tab, Retroceso, Entrar o Esc.

OnKeyUp: Se activa cuando se libera una tecla.

OnMouseDown: Ocurre cuando se presiona el botón del ratón mientras está sobre el componente.

OnMouseMove: Ocurre cuando se mueve el ratón sobre el componente.

SISTEMAS DE NUMERACIÓN

Los computadores, como dispositivos electrónicos que son, ejecutan sus funciones mediante señales eléctricas y éstas pueden tener dos estados: encendido y apagado, verdadero y falso ó cero y uno.

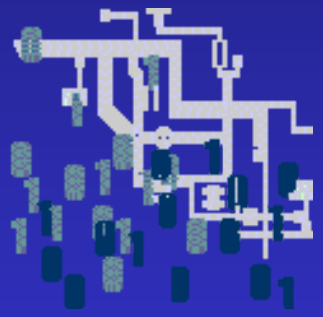
Por esta razón, la mínima unidad de información que puede manejar el computador es aquella que es capaz de adoptar dos estados posibles.

A esta unidad se le conoce con el nombre de **BIT** que significa **BInary digiT.**

SISTEMAS DE NUMERACIÓN

BYTE: Es una agrupación de ocho bits que son tratados como una unidad.

1	0	0	1	1	0	1	1



PALABRA: es la agrupación de uno ó más bytes, y está ligado al concepto de transferencia de datos entre los componentes del computador.

SISTEMAS DE NUMERACIÓN

La unidad de memoria llamada byte, se utiliza como unidad de medida de la capacidad de almacenamiento de datos en un computador. Como esta unidad es muy pequeña (un Byte es capaz de almacenar un caracter), se utilizan los múltiplos:

- Kilobytes (Kb) = 1.024 bytes
- Megabytes (Mb) = 1.000 Kb
- Gigabytes (Gb) = 1.000 Mb

Por ejemplo, algunos pendrives tienen una capacidad de almacenamiento de 1Gb.

SISTEMAS DE NUMERACIÓN

Tipos de datos

La forma de almacenar datos en un computador difiere respecto al tipo de datos que se desea almacenar. De esta forma tenemos datos numéricos y alfanuméricos.

✓ Los datos alfanuméricos se almacenan mediante un código denominado **ASCII** (American Standard Code for Information Interchange) donde un caracter ocupa un byte.

✓ Los datos numéricos se almacenan mediante la conversión al sistema binario.

TABLA DE CÓDIGO ASCII

Código ASCII de impresión de 8 bits (con caracteres gráficos) (PC)

Izqda.→ dcha. ↓	0000 0	0001 1	0010 2	0011 3	0100 4	0101 5	0110 6	0111 7	1000 8	1001 9	1010 A	1011 B	1100 C	1101 D	1110 E	1111 F
0000 0	NUL 0	16	SP 32	0 48	@ 64	P 80	. 96	p 112	Ç 128	É 144	á 160	⣿ 176	└ 192	╨ 208	α 224	≡ 240
0001 1	1	DC1 17	! 33	1 49	A 65	Q 81	a 97	q 113	ü 129	æ 145	í 161	⣿ 177	┴ 193	╤ 209	± 225	241
0010 2	2	DC2 18	" 34	2 50	B 66	R 82	b 98	r 114	é 130	Æ 146	ó 162	⣿ 178	┬ 194	╥ 210	⌠ 226	≥ 242
0011 3	♥ 3	DC3 19	# 35	3 51	C 67	S 83	c 99	s 115	â 131	ô 147	ú 163	┤ 179	├ 195	╙ 211	⌡ 227	≤ 243
0100 4	♦ 4	DC4 20	$ 36	4 52	D 68	T 84	d 100	t 116	ä 132	ö 148	ñ 164	┼ 180	─ 196	╘ 212	Σ 228	⌠ 244
0101 5	♣ 5	§ 21	% 37	5 53	E 69	U 85	e 101	u 117	à 133	ò 149	Ñ 165	╡ 181	┼ 197	╒ 213	σ 229	⌡ 245
0110 6	♠ 6	22	& 38	6 54	F 70	V 86	f 102	v 118	å 134	û 150	½ 166	╢ 182	╞ 198	╓ 214	µ 230	÷ 246
0111 7	BEL 7	23	' 39	7 55	G 71	W 87	g 103	w 119	ç 135	ù 151	¼ 167	╖ 183	╟ 199	╫ 215	τ 231	≈ 247
1000 8	BS 8	CAN 24	(40	8 56	H 72	X 88	h 104	x 120	ê 136	ÿ 152	¿ 168	╕ 184	╚ 200	╪ 216	∅ 232	248
1001 9	HT 9	EM 25	) 41	9 57	I 73	Y 89	i 105	y 121	ë 137	Ö 153	⌐ 169	╣ 185	╔ 201	┘ 217	θ 233	· 249
1010 A	LF 10	26	* 42	: 58	J 74	Z 90	j 106	z 122	è 138	Ü 154	¬ 170	▌ 186	╩ 202	┌ 218	Ω 234	· 250
1011 B	VT 11	ESC 27	+ 43	; 59	K 75	[91	k 107	{ 123	ï 139	ç 155	fi 171	╗ 187	╦ 203	⣿ 219	δ 235	√ 251
1100 C	FF 12	FS 28	´ 44	< 60	L 76	/ 92	l 108	| 124	î 140	£ 156	¼ 172	╝ 188	╠ 204	■ 220	∞ 236	η 252
1101 D	CR 13	29	– 45	= 61	M 77	] 93	m 109	} 125	ì 141	¥ 157	¡ 173	╜ 189	= 205	▌ 221	∅ 237	² 253
1110 E	SO 14	30	. 46	> 62	N 78	^ 94	n 110	~ 126	Ä 142	₧ 158	« 174	╛ 190	╬ 206	▌ 222	ε 238	■ 254
1111 F	SI 15	31	/ 47	? 63	O 79	— 95	o 111	127	Å 143	ƒ 159	» 175	┐ 191	╧ 207	■ 223	∩ 239	SP 255

Nota: En los PC se selecciona el carácter deseado pulsando simultáneamente la tecla ALT y el número decimal que aparece en la parte inferior de la celda correspondiente. Por ejemplo, al pulsar ALT-228 se selecciona Σ.

Datos Numéricos

Para que el computador realice las operaciones numéricas, desarrolla los siguientes pasos:

- ✓ Convertir datos decimales de entrada a binario
- ✓ Realizar operaciones aritméticas en forma binaria
- ✓ Convertir los resultados binarios a decimal.

SISTEMAS DE NUMERACIÓN

Conversiones

Para convertir de binario a decimal:

$$(11001)_2 = 1*2^4 + 1*2^3 + 0*2^2 + 0*2^1 + 1*2^0 = 25_{10}$$

Para convertir de decimal a binario:

35/2 = 17 resto 1
17/2 = 8 resto 1
8/2 = 4 resto 0
4/2 = 2 resto 0
2/2 = 1 resto 0
1/2 = 0 resto 1

$(100011)_2$

SISTEMAS DE NUMERACIÓN

Representación sin signo

Si el computador representa números enteros positivos (sin signo) que ocupen un byte, tenemos los valores extremos:

$(00000000)_2$ $(11111111)_2$

$(0)_{10}$ $(255)_{10}$

Si el computador representa números enteros positivos (sin signo) que ocupen dos bytes, tenemos los valores extremos:

$(0000000000000000)_2$ $(1111111111111111)_2$

$(0)_{10}$ $(65.535)_{10}$

SISTEMAS DE NUMERACIÓN

Representación con signo

Si el computador representa números con signo que ocupen un byte, tenemos:

Menor valor negativo	Mayor valor positivo
$(10000000)_2$	$(01111111)_2$
$(-128)_{10}$	$(127)_{10}$

Si el computador representa números con signo que ocupen dos bytes, tenemos:

Menor valor negativo	Mayor valor positivo
$(1000000000000000)_2$	$(0111111111111111)_2$
$(-32.768)_{10}$	$(32.767)_{10}$

SISTEMAS DE NUMERACIÓN

Aritmética Binaria

Suma binaria de dos dígitos:

0 + 0 = 0
0 + 1 = 1
1 + 0 = 1
1 + 1 = 0
Se acarrea 1

Ejemplo:

1101	13
+100101	+37
110010	050

Aritmética Binaria

Para la resta binaria se aplica la misma regla de la aritmética decimal:

$$16 - 12 = 4$$
$$16 + (-12) = 4$$

De esta forma se convierte la resta en una suma de un número positivo y uno negativo.

Complemento a dos

Para representar un número entero negativo, en el sistema binario se utiliza el complemento a dos.

00000100	+4	Número 4
11111011		Complemento a uno
+1		
11111100	–4	Complemento a dos

SISTEMAS DE NUMERACIÓN

Resta binaria

12 - 4 = 8

00001100	+12
11111100	–04
100001000	+08

4 - 12 = -8

00000100	+04
11110100	–12
11111000	–08

Otras Conversiones

Existen otros sistemas de numeración que guardan una estrecha relación con el binario.

✓ El sistema hexadecimal utiliza 16 dígitos para representar los números: 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F.

SISTEMAS DE NUMERACIÓN

Sistema Hexadecimal

Hexa	Binario	Hexa	Binario
0	0000	9	1001
1	0001	A	1010
2	0010	B	1011
3	0011	C	1100
4	0100	D	1101
5	0101	E	1110
6	0110	F	1111
7	0111		
8	1000		

CLASIFICACIÓN DE LOS DATOS

De los diversos criterios que existen nos interesan:

- Por su estructura: solemos agruparlos en datos simples y compuestos. los datos simples son aquellos que se manipulan en forma individual. los datos compuestos son datos simples relacionados, agrupados bajo alguna estructura.
- Por el tipo que representan: los clasificamos en numéricos, lógicos y alfanuméricos. los datos numéricos representan cantidades a ser operadas aritméticamente. los lógicos implican el cumplimiento de una condición. los alfanuméricos son aquellos que no pueden incluirse en las dos anteriores.

Tipos de Datos Enteros Empleados en *Borland C*:

Tipo de Datos	# Bytes	Rango
char	1	-128 a 127
unsigned char	1	0 a 255
unsigned int	2	0 a 65.535
int	2	-32.768 a 32.767
long int	4	-2.147.483.648 a 2.147.483.647
unsigned long	4	0 a 4.294.967.295

Tipos de Datos Reales Empleados en Borland C:

Tipo	# Bytes	Rango
float	4	$1.5x10^{-45}$ a $3.4x10^{38}$
double	8	$5.0x10^{-324}$ a $1.7x10^{308}$
long double	10	$3.4x10^{-4932}$ a $3.4x10^{4932}$

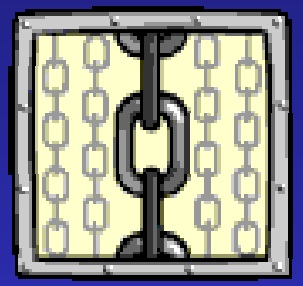
Cadenas Alfanuméricas

Las cadenas alfanuméricas son un tipo particular de datos y es una serie de caracteres (que pueden ser letras, números o una combinación de ambos) y que se usa de manera intensiva en los controles con los que el usuario tiene interacción en una aplicación BUILDER.

BUILDER tiene dos tipos de cadenas: Corta, Larga .

Cadenas Cortas

Este elemento puede representar caracteres alfanuméricos con un máximo de 255 caracteres y están siempre terminadas en el carácter nulo. Su forma de declaración es:



```
char  Cadena_Corta[ ];
```

Cadenas Largas

Este elemento a diferencia del anterior tiene un comportamiento dinámico y su tamaño sólo está limitado por la memoria disponible. Su forma de declaración es:

```
AnsiString Cadena;
```

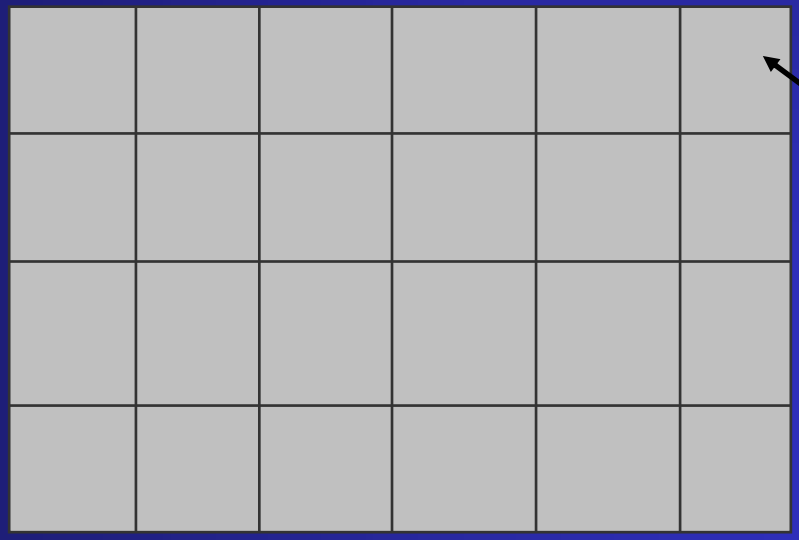
Son muy flexibles pero pueden volverse muy lento su manejo.

VARIABLE

- Los datos representados por las constantes pueden ser almacenados en una localidad de memoria del computador. El conjunto de letras y dígitos (identificador) que se emplea para representar esa localidad de memoria se le denomina variable.
- Al almacenar una constante en una localidad de memoria se pierde el contenido anterior de ésta. A este proceso se le suele llamar escritura destructiva en memoria.
- El contenido de una localidad de memoria puede ser invocado tantas veces como se desee sin que se altere su valor. A este proceso se le suele llamar lectura no destructiva en memoria.

ALMACENAMIENTO DE VARIABLES

El computador almacena los datos sirviéndose de su memoria. podemos imaginar la memoria como un casillero donde cada casilla representa una localidad de memoria. en cada una de ellas se puede almacenar una única constante.



Variables

Las variables tienen que ser declaradas antes de que puedan ser utilizadas.

Las variables se pueden declarar para las unidades y también se pueden declarar locales en las funciones.

Las variables sólo pueden poseer un valor a la vez, de hecho el proceso de asignar un valor a una variable se constituye en un proceso de escritura destructiva.

El que una variable se le asigne a otra no implica que la primera variable quedará vacía.

Identificadores:

Para identificar unidades, constantes, variables, funciones y otros elementos se deben respetar unas simples reglas:

- A pesar de poder usar letras y números en el identificador, el primer carácter debe ser una letra.
- No se pueden emplear palabras reservadas.
- No se pueden emplear caracteres distintos a letras y números salvo el carácter _
- **No** es irrelevante si el nombre tiene las letras en mayúscula o minúscula.

LA ASIGNACIÓN

Se define como asignación al hecho de darle un valor a una variable o la propiedad de un objeto. Consideraremos dos formas:

- Una desde el exterior llamada lectura de datos, que consiste en el ingreso de estos desde el exterior hacia la memoria y para ello utilizamos el teclado. cada dato introducido se almacena en la variable indicada en la instrucción de lectura.
- Otra interna al algoritmo en cuestión llamada instrucción de asignación y detallaremos más adelante.

Asignación

La asignación en Borland C se realiza según el siguiente esquema:

Identificador_de_Variable = *Expresión*

Siempre se evaluará primero la *Expresión* y luego el resultado será guardado en la variable indicada a la izquierda de el signo de asignación.

Tanto el resultado de la expresión como la variable que recibirá su resultado deben ser compatibles en el tipo de resultado o se presentará un error.

Operadores

Los operadores se emplean en *Borland C* para manipular datos dentro de expresiones, los más comunes son los siguientes:

Operadores Matemáticos

Operador	Descripción	Ejemplo
+	Suma	x= y + z
-	Resta	x= y - z
*	Multiplicación	X= y * z
/	División	X= y / z
%	Resto de la División Entera	X= y % z
++	Incremento	++X
--	Decremento	--y

Operadores de Relación

Operador	Descripción	Ejemplo
= =	Igual a	X = = Z
!=	Diferente de	X != Z
<	Menor que	X < Z
>	Mayor que	X > Z
<=	Menor o Igual a	X <= Z
>=	Mayor o Igual a	X >= Z

Operadores Lógicos

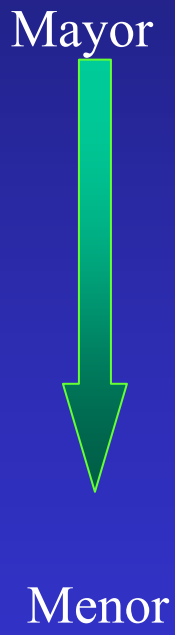
Operador	Descripción	Ejemplo
&&	And Lógico (“y”)	(x = = y) && (x < 10)
||	Or Lógico (“o”)	(x != 1) || (y <= 10)
!	Negación Lógica	!(x != z)

Tanto los operadores de Relación como los operadores Lógicos se ubican en expresiones de resultado tipo **Boolean** (Verdadero ó Falso, uno ó cero)

PRIORIDAD DE LOS OPERADORES

La jerarquía general de operación de los operadores estudiados es tal como se muestra:

PRIORIDAD	OPERADORES
1	! + - UNARIO ++ --
2	* / %
3	+ - BINARIO
4	< <= > >=
5	== !=
6	&&
7	||



Para alterar esta jerarquía se emplean los paréntesis.

Operadores de las Cadenas

Por ser un elemento vital en la interacción con el usuario es prudente detenerse en los principales aspectos de las cadenas alfanuméricas.

Concatenación

Se emplea el operador + para juntar cadenas.

Así por ejemplo: “primer” + “ ejemplo”, dará como resultado una única cadena “primer ejemplo”

Conversión de Cadenas

Conversión de Cadenas

Las propiedades que se emplean en los componentes de BUILDER 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.

Conversión de Cadenas

FUNCION	PARÁMETRO	RESULTADO
Valor.ToDouble( )	Alfanumérico	Real Largo
Valor.ToFloat( )	Alfanumérico	Real
Valor.ToInt( )	Alfanumérico	Entero
AnsiString(Valor)	Numérico	Alfanumérico
Valor.c_str( )	Alfanumérico Largo	Alfanumérico Corto

Funciones de Biblioteca

Cualquier otra operación en BUILDER, numérica o no, que no tenga un operador asignado para ello, debe codificarse.

En BUILDER existen bibliotecas de funciones ya codificadas, que podemos utilizar, siempre y cuando incluyamos en nuestro proyecto las librerías que las contienen.


```
# include <librería.h>
# include "librería.h"
```

Funciones de Biblioteca

BUILDER

Función	Descripción	Tipo del Argumento	Tipo del Resultado	Librería Asociada
acos (x)	Arco Coseno de x	Real	Real	math.h
asin (x)	Arco Seno de x	Real	Real	math.h
atan (x)	Arco Tangente de x	Real	Real	math.h
ceil (x)	Redondea x al entero más pequeño no menro que x	Real	Entero	math.h
cos (x)	Coseno de x (en radianes)	Real	Real	math.h
exp (x)	Función Exponencial e^x	Real	Real	math.h
fabs (x)	Valor Absoluto de x	Real	Real	math.h
floor (x)	Redondea x al entero más grande no mayor que x	Real	Entero	math.h
fmod(x)	Residuo de y/x como real	Real	Real	math.h

Funciones de Biblioteca

BUILDER

Función	Descripción	Tipo del Argumento	Tipo del Resultado	Librería Asociada
log (x)	Logaritmo Natural de x (base e)	Real	Real	math.h
log10 (x)	Logaritmo de x (base 10)	Real	Real	math.h
pow (x,y)	X elevado a la potencia y (x^y)	Real	Real	math.h
rand ( )	Genera un número entero aleatoriamente	Vacio	Entero	stdlib.h
random (x)	Genera un número aleatorio entero entre 0 y x-1	Entero	Entero	stdlib.h
randomize(x)	Inicializa el generador de números aleatorios con la semilla tomada del reloj	Vacio	Vacio	stdlib.h time.h
sin (x)	Seno de x (en radianes)	Real	Real	math.h
sqrt (x)	Raíz Cuadrada de x	Real	Real	math.h
tan (x)	Tangente de x (en radianes)	Real	Real	math.h

Creando un Proyecto

Al crear un proyecto por primera vez, BUILDER crea un mínimo de 4 archivos:

Archivo Fuente del Proyecto: que contiene el código de arranque, extensión cpp.

La Unidad del Formulario Principal: contiene la declaración de la clase del formulario principal. Será creado uno por cada nuevo formulario.

El ***archivo de Recursos del Formulario Principal*** y el ***archivo de Recursos del Proyecto***, que son archivos binarios que describen al formulario principal y al icono de la aplicación.

BUILDER

Los archivos que BUILDER produce pueden dividirse en dos categorías: los archivos de los que depende para construir el proyecto y los que se crean cuando se compila y vincula un proyecto.

El conjunto mínimo de archivos comprende a los que poseen la extensión ***.cpp***, ***.dfm, .res, .h, .ddp*** y ***.bpr***.

Cualquier aplicación termina teniendo varios archivos fuentes, estos se denominan **Unidades**.

Unidades

Una aplicación GUI de BUILDER contendrá al menos dos unidades directamente visibles por el usuario:

- El archivo de cabecera que contiene la declaración de la forma y menciona a BUILDER la lista de componentes y los eventos que tendrá la aplicación. Tiene la extensión **.h**
- El archivo fuente que contiene 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. Tiene la extensión **.cpp**

Anatomía de una unidad

Las unidades deben seguir un formato predefinido, el que se muestra a continuación corresponde a una unidad básica cpp:

Inclusión de unidades de extensión **.h**

Declaración de constantes

declaración de identificadores exportados

Identificador de la forma

{

código real de la unidad

}

Encabezado de la función asociada al evento

{

Código asociado al evento

}

Instrucciones Compuestas

Esta Instrucción compuesta permite agrupar varias instrucciones como un solo bloque. Por si sola esta Instrucción no tiene ningún efecto operativo, pero siempre se usan en conjunto con otras, su forma de uso es:

```
{
    Grupo de Instrucciones;
}
```

El punto y coma (;) se coloca al final de cada una de las instrucciones salvo las excepciones previstas en el lenguaje.

Comentarios

Los comentarios son líneas de texto en el código fuente colocadas con fines de documentación. No son realmente parte del código y no se pueden ejecutar.

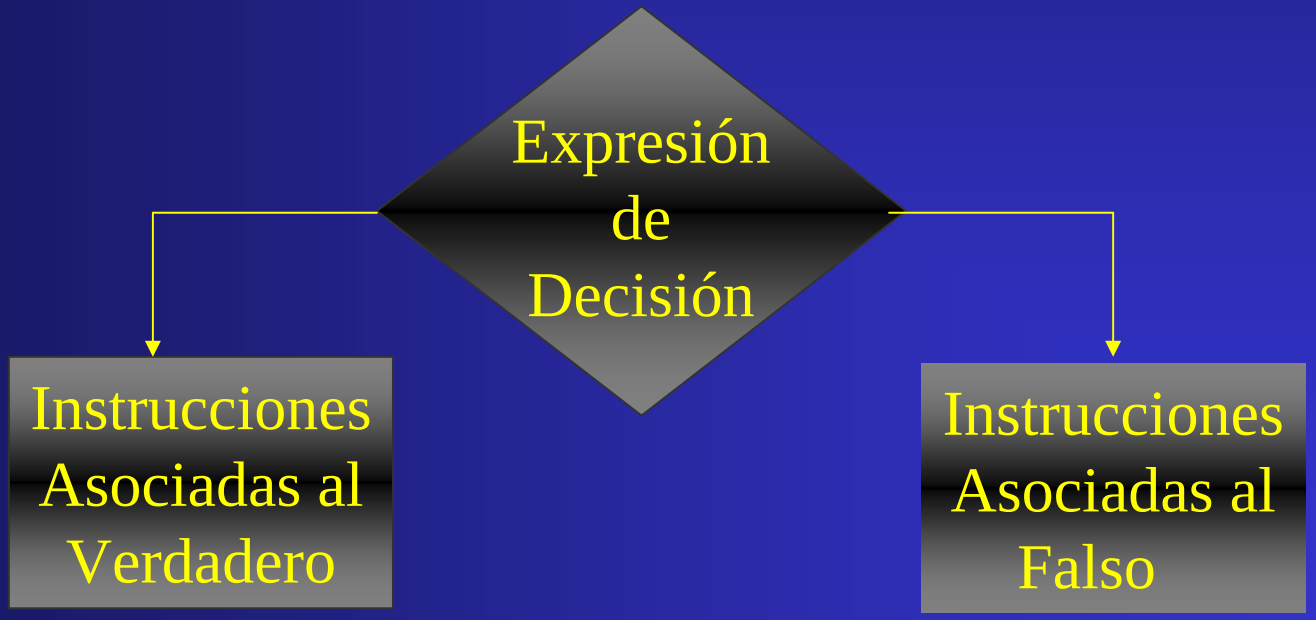
Existen dos formas de declararlos:

- Usando los Símbolos asterisco y barra inclinada, abriendo /* y cerrando */, para marcar inicio y fin del comentario.
- Usando dos barras inclinadas // al inicio, para comentarios en una sola línea.

TOMANDO DESICIONES

Instrucciones Compuestas - **PREGUNTAS** -

Es muy común que sea necesario evaluar una condición para luego realizar una sección de código u otra, en el caso de *Borland C* se dispone de la Instrucción **if-else** y es una forma de representar la siguiente situación:



Intrucciones Compuestas - **PREGUNTAS** -

La situación anterior se codifica como sigue:

if (*Expresión_Lógica)*

Instrucción Asociada al Verdadero;

else

Instrucción Asociada al Falso;

En caso de que la Expresión lógica resulte Verdadera (*TRUE ó distinto de cero*) se realiza su instrucción asociada y luego se salta el **else** y su instrucción, caso contrario el flujo va directamente a la instrucción asociada al Falso (FALSE ó igual a cero).

Instruccioness Compuestas - **PREGUNTAS** -

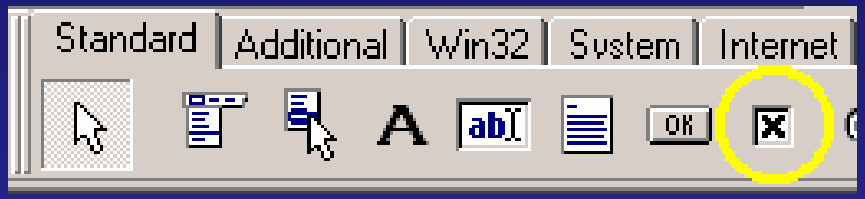
Es muy común que sólo sea necesario realizar una acción si se cumple una condición y de lo contrario se continúa la ejecución normalmente, esta variante se contempla en la Instrucción **if** al ser opcional el **else**, la forma de la Instrucción de esta manera será:

if (*Expresión_Lógica)*

Instrucción Asociada al Verdadero;

En caso de que la Expresión lógica resulte Verdadera (*TRUE ó distinto de cero*) se realiza su instrucción asociada continuando con el resto del programa, caso contrario el flujo salta la instrucción asociada al Verdadero.

Componente **CheckBox**



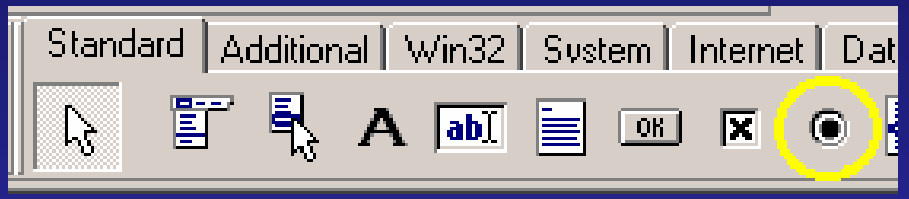
Variante de los botones, este componente presenta un cuadro con una etiqueta, el cuadro puede ser marcado o no en la ejecución siendo una decisión binaria.

Pueden ser marcados cuantos se deseen en un formulario.

La propiedad ***Checked*** indica los posibles valores que puede tomar el control (Marcado, No Marcado)

Se ubica en la pestaña ***Standard*** de la barra de componentes.

Componente **RadioButton**



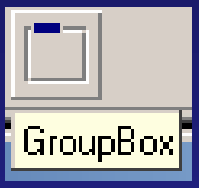
Otra variante de los botones, este componente presenta un círculo con una etiqueta, el círculo puede ser marcado o no en la ejecución siendo una decisión binaria.

Sólo puede ser marcado uno por grupo, de tal manera que al marcar uno el que estaba marcado deja de estarlo.

La propiedad ***Checked*** indica los posibles valores que puede tomar el control (Activado o No Activado).

Se ubica en la pestaña ***Standard*** de la barra de componentes.

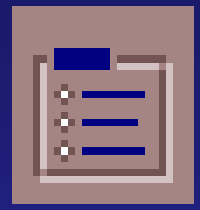
Componente **GroupBox**



Este componente más que emplearse para realizar acciones se usa para agrupar otros componentes, en particular los **RadioButton**, ya que si se desean agrupar dos o más de estos elementos es indispensable que se hallen contenidos en **GroupBox** distintos.

Su propiedad más empleada es ***Caption***. Se encuentra en la pestaña ***Standard*** de la barra de Componentes.

Componente **RadioGroup**



Este componente agrupa particularmente los **RadioButton**.

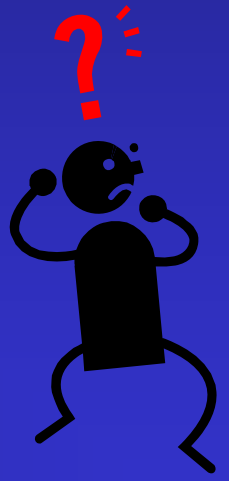
Sus propiedades más empleadas son ***Items,*** donde se le colocan los nombres de los distintos RadioButtons y ***ItemIndex*** que indica cuál ***RadioButton*** está marcado (valor cero el primero).

Se encuentra en la pestaña ***Standard*** de la barra de Componentes.

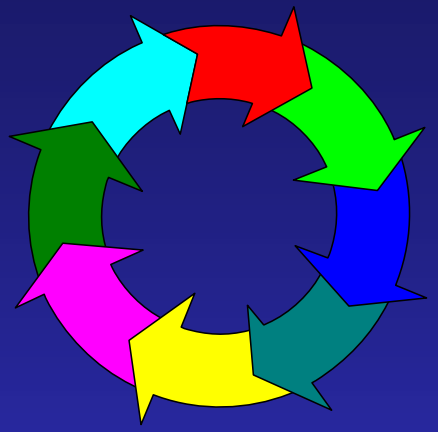
Instrucciones Compuestas - **PREGUNTAS** -

Borland C posee otra instrucción para realizar preguntas que es **switch** que permite ejecutar uno de varios bloques de código con base en el resultado de una expresión. En el fondo esta es una situación de varios **IF** anidados, su esquema es el siguiente:

switch (*Expresión*)**{**
    case *Valor 1* :*Instrucción 1*; **break**;
    case *Valor 2* :*Instrucción 2*; **break**;
    :
    case *Valor n* :*Instrucción n*; **break**;
default
    Instrucción asociada al default;
}



LAZOS ITERATIVOS

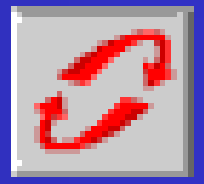


Instrucciones Compuestas - **CICLOS** -

Es una instrucción que permite repetir otra(s) instrucción(es) tantas veces como se necesite.

Existen diferentes tipos de ciclos y todos tienen en común:

- ✓Un punto de inicio.
- ✓Un cuerpo o el grupo de instrucciones a repetir.
- ✓Un punto de terminación.
- ✓Una prueba de una condición que determina cuando debe concluir el ciclo.



Ciclo **while**

Es un ciclo que contiene una condición de prueba que se verifica al inicio de cada iteración. Mientras esta condición sea verdadera (***True ó distinto de cero***), el ciclo continúa operando.

El esquema de esta instrucción es:

while (Expresión_Lógica)

Instrucción;

A diferencia de otros ciclos, en este no se sabe a priori cuantas vueltas se van a dar.

Ciclo **do - while**

Es un ciclo que contiene una condición de prueba que se verifica al final de cada iteración. El ciclo opera mientras que la condición evaluada sea verdadera (***True ó distinto de cero***).

El esquema de esta instrucción es:

```
do{
    Instrucción;
}while (Expresión_Lógica);
```

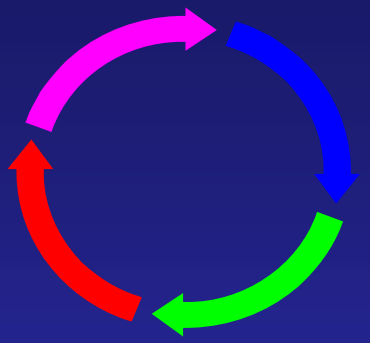
El ciclo siempre se dará al menos una vuelta, igual que el **while** necesariamente no se conoce de antemano el número de iteraciones que éste realizará.

Ciclo **for**

Es una estructura que se emplea para repetir un grupo de instrucciones en un programa, un número predeterminado de veces.

Primero se le asigna a la variable del contador el “valor inicial”. Luego antes de cada ejecución de la instrucción o instrucción compuesta del bucle, se comprueba si una expresión lógica es verdadera ó distinta de cero. Si esta expresión es verdadera, continúa el ciclo. La variable del contador se incrementa (o reduce) según el tipo ordinal.

Ciclo for



El esquema de esta instrucción es:

for(*Inicialización ; Expresión_Lógica ; Paso* **)**

Instrucción;

La primera vez que se ejecuta el ciclo for, se inicializan las variables en juego y se verifica la Expresión_Lógica, si esta es verdadera, se ejecuta la Instrucción.

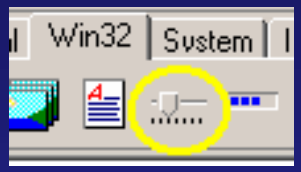
Las siguientes veces que se ejecuta el for, primero se ejecuta el paso, luego se verifica la Expresión_Lógica, si es verdadera se repite la Instrucción.

Componente **ProgressBar**

Es un componente que se ubica en la pestaña Win 32, es una representación visual que indica al usuario el avance de un proceso mediante una barra de avance rectangular, que de forma gradual va “llenándose” de izquierda a derecha.

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

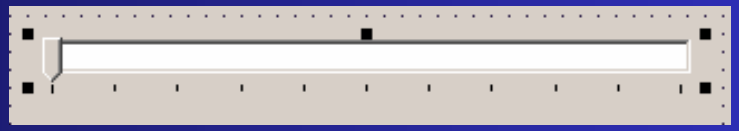
Componente **TrackBar**



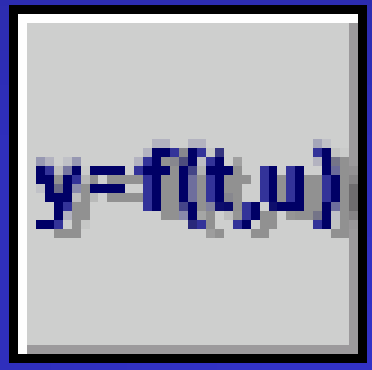
Este componente permite ingresar datos enteros por el usuario.

Su accionar se determina usando las propiedades ***Min*** y ***Max***, para asentar sus valores extremos. La posición de la barra de desplazamiento puede establecerse u obtenerse mediante la propiedad ***Position***.

Se ubica en la pestaña ***Win32*** de la barra de componentes.



FUNCIONES



FUNCIONES

Son secciones de código separadas del programa principal. Se ejecutan cuando se necesitan realizar acciones específicas en el programa.

En conjunto pueden denominarse subrutinas o subprogramas, siendo posible invocarlas o llamarlas (usarlas) varias veces y en cualquier momento de la ejecución del programa.

FUNCIONES

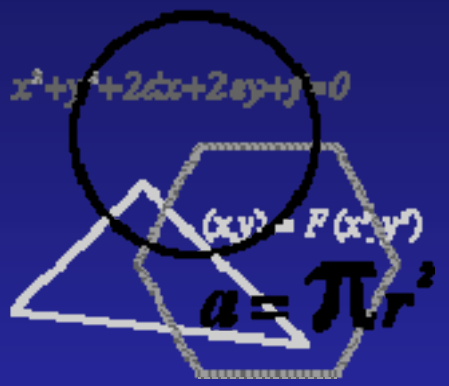
Se pueden clasificar en:

- ✓ Funciones invocadas por el usuario (Eventos).
- ✓ Funciones invocadas por código.

Asociado a las funciones se tiene el término parámetro: que es un dato que se pasa a una función o subprograma y se usa intercambiar información entre la función y el exterior.

FUNCIONES

Una función posee el siguiente esquema:



Void(u otro tipo)[_fastcall] *Identificador* (Lista de Tipo y Parámetros)

{ Zona de Declaraciones Locales

Zona de Instrucciones Asociadas a la función

}

Funciones que Devuelven Vacío

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

Cada función posee una cabecera seguida por un bloque similar al de un programa principal.
Una función es activada por una simple instrucción de ejecución de función.

Funciones que Devuelven Vacío

Una función que devuelve vacío, posee el siguiente esquema:

```
void Identificador (Lista de Parámetros con su Tipo)
{
        Zona de Declaraciones Locales

        Zona de Instrucciones Asociadas a la Función
}
```

Funciones que Devuelven Valores

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

Ambas se comunican usualmente mediante los parámetros.

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

Funciones que Devuelven Valores

Una función que devuelve valores, posee el siguiente esquema:

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

Funciones que Devuelven Valores

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

- Las Funciones que devuelven valores tienen un Tipo de resultado.
- Las Funciones que devuelven valores especifican siempre un valor.
- Dentro del cuerpo de la Función debe existir la instrucción **return** seguida del valor que se le quiere dar a la función.
- El llamado a una Función debe aparecer siempre en una expresión de su mismo tipo.

Funciones que Devuelven Valores

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

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

        return Potencia;
 }
```

Funciones

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

Funciones

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

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

Declaración de Funciones

Alcance de los Objetos

Locales

Son aquellos que sólo tienen vigencia en una parte del programa

Globales

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

Funciones

PARÁMETROS

Son aquellas variables que tienen vigencia solamente dentro de la función a quien pertenecen, pero transmiten sus valores al exterior de la función.

Parámetros

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

- ✓ **Por Valor**
- ✓ **Por Referencia ó Dirección**

Parámetros de Valor

Reciben el valor que es enviado y dentro del cuerpo de la función actúan como una variable local.

Lo anterior implica que esta variable puede ser modificada dentro de la función y la variable original permanecerá sin cambios.

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

Como ejemplo:

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

Parámetros por Valor

```
int Nombre(int Numero, int  A, int Z)
 {
        float P;

        A = A+(Z++) ;

        P = Numero*A/Z ;

        return P;
 }
```

Parámetros por Referencia ó Dirección

En este caso la variable original en la llamada sufrirá 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.

Ejemplo de su declaración es:

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

Parámetros por Dirección o Referencia

```
int Nombre( int * Numero, int * A, int * Z)
 {
        float P;

        *A = *A+(*Z++) ;

        P =*Numero*(*A)/*Z ;

        return P;
 }
```

Parámetros por Dirección o Referencia

Cuando se manda a ejecutar una función que contiene parámetros por referencia, se antepone el símbolo & a la variable a la cual se envía su dirección.

Ejemplo es:

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{ int A, Numero, A, Z;

     W=Nombre(&Numero, &A, &Z);

}
```

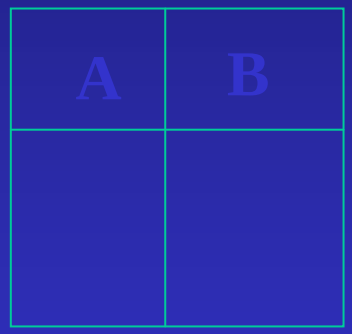
Ejemplo:

```
#include <vcl.h>
#pragma hdrstop

#include "funciones.h"
//---------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;

void Numero_1();
void Numero_2 (float A, float B);
void Numero_3 (float *X, float *Y);
float A, B;

//---------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
}
```



Ejemplo:

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
/*Programa de Funciones*/
    A = 5.4;
    B = -1.5;
    Numero_1();
    ShowMessage("EL VALOR DE A ES: "+ AnsiString(A));
    ShowMessage("EL VALOR DE B ES: "+ AnsiString(B));
    Numero_2(A, B);
    ShowMessage("EL VALOR DE A ES: "+ AnsiString(A));
    ShowMessage("EL VALOR DE B ES: "+ AnsiString(B));
    Numero_3(&A, &B);
    ShowMessage("EL VALOR DE A ES: "+ AnsiString(A));
    ShowMessage("EL VALOR DE B ES: "+ AnsiString(B));
}
//---------------------------------------------------------------------------
void Numero_1()
{
float A, X;
    X = 1;
    A = X + 1;
    B = 3 * A;
    ShowMessage("EN NUMERO_1 SE TIENE QUE X= "+ AnsiString(X));
    ShowMessage("EN NUMERO_1 SE TIENE QUE A= "+ AnsiString(A));
    ShowMessage("EN NUMERO_1 SE TIENE QUE B= "+ AnsiString(B));
}
```

A	B	A1	X1
5.4	-1.5	2	1
	6		

Ejemplo:

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
/*Programa de Funciones*/
    A = 5.4;
    B = -1.5;
    Numero_1();
    ShowMessage("EL VALOR DE A ES: "+ AnsiString(A));
    ShowMessage("EL VALOR DE B ES: "+ AnsiString(B));
    Numero_2(A, B);
    ShowMessage("EL VALOR DE A ES: "+ AnsiString(A));
    ShowMessage("EL VALOR DE B ES: "+ AnsiString(B));
    Numero_3(&A, &B);
    ShowMessage("EL VALOR DE A ES: "+ AnsiString(A));
    ShowMessage("EL VALOR DE B ES: "+ AnsiString(B));
}
//---------------------------------------------------------------------------
void Numero_2 (float A, float B)
{
int i;

    B = 2;
    i = 3;
    ShowMessage("EN NUMERO_2 SE TIENE QUE i= "+ AnsiString(i));
    ShowMessage("EN NUMERO_2 SE TIENE QUE A= "+ AnsiString(A));
    ShowMessage("EN NUMERO_2 SE TIENE QUE B= "+ AnsiString(B));
}
```

A	B	A2	B2	i
5.4	6	5.4	6	3
			2	

Ejemplo:

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
/*Programa de Funciones*/
    A = 5.4;
    B = -1.5;
    Numero_1();
    ShowMessage("EL VALOR DE A ES: "+ AnsiString(A));
    ShowMessage("EL VALOR DE B ES: "+ AnsiString(B));
    Numero_2(A, B);
    ShowMessage("EL VALOR DE A ES: "+ AnsiString(A));
    ShowMessage("EL VALOR DE B ES: "+ AnsiString(B));
    Numero_3(&A, &B);
    ShowMessage("EL VALOR DE A ES: "+ AnsiString(A));
    ShowMessage("EL VALOR DE B ES: "+ AnsiString(B));
}
//---------------------------------------------------------------------------
void Numero_3 (float *X, float *Y)
{
float A;
    *X = 1.5;
    *Y = 4.2;
    A = *X * *Y;
    ShowMessage("EN NUMERO_3 SE TIENE QUE X= "+ AnsiString(*X));
    ShowMessage("EN NUMERO_3 SE TIENE QUE Y= "+ AnsiString(*Y));
    ShowMessage("EN NUMERO_3 SE TIENE QUE A= "+ AnsiString(A));
}
```

	Y	X
A3	B	A
6.3	6	5.4
	4.2	1.5

Cajas de Diálogo

BUILDER provee la posibilidad de comunicarse con el usuario mediante cuatro 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 para dar un simple aviso como para solicitar confirmación o no de una acción.

Tipos de Cajas de Diálogo disponibles:

Las primera se invoca empleando una simple instrucción de llamado de función que devuelve vacio:

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

ShowMessage("Mensaje que aparece");

Tipos de Cajas de Diálogo disponibles:

MessageDlgPos: en aspecto es idéntico al anterior, sólo que uno puede indicar la posición de aparición de la caja de diálogo al indicar las coordenadas de la esquina superior izquierda.

Respuesta=MessageDlgPos("Mensaje que aparece",

Tipo de Caja de Diálogo, [lista de botones], Ayuda, X, Y, Botón por defecto, imagen);

Ayuda indica si este elemento posee Ayuda (Help) o no y X y Y son la posición en aparecerá la caja de diálogo.

Lista de Botones en las Cajas de Diálogo disponibles:

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

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

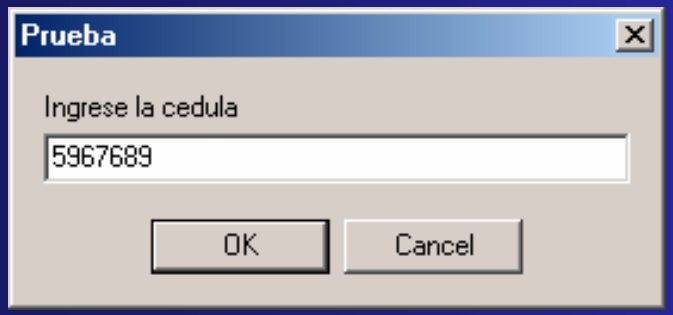
Tipos de Cajas de Diálogo disponibles:

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

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

Cajas de Edición

Las Cajas de Edición permiten mostrar información y solicitar que el usuario presione un botón como respuesta, pero si se quiere, en este estilo solicitud de datos por parte del usuario, se pueden emplear las funciones ***InputBox*** (que retorna una cadena alfanumérica) asociada a ***InputQuery*** (que retorna un valor **Boolean**).



InputBox

Cajas de Edición

InputBox se emplea como función y su sintaxis es:

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

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

InputQuery retorna *true* o f*alse* si el usuario presionó el botón OK en el InputBox (aunque se debe verificar si no cambió el contenido de la caja).

Datos Definidos por el Usuario

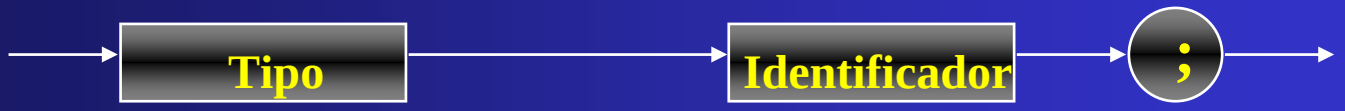
y Datos Compuestos

Tipos de Datos Definidos por el Usuario

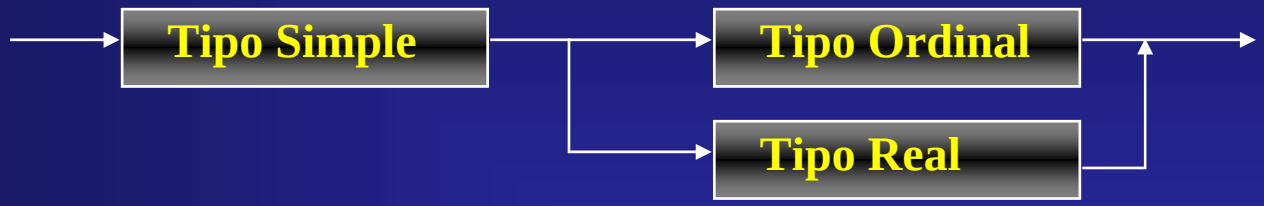
El programador puede definir tipos de datos adicionales a los estándar del lenguaje, ya que es posible que para la resolución de un problema el uso de los tipos predefinidos no garantice una solución óptima.

¿Cómo se define un tipo?

La definición la realizamos en la sección declaración de variables globales o locales y se realiza según el siguiente esquema:



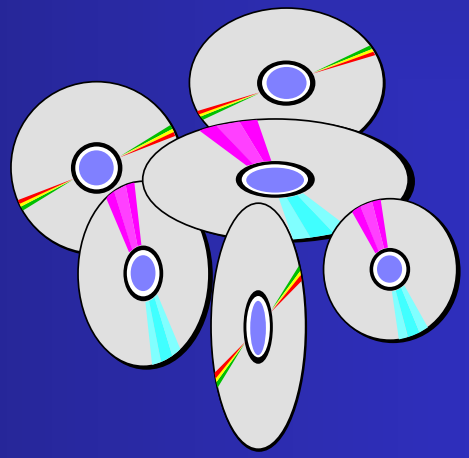
Tipo Simple



Tipo Ordinal

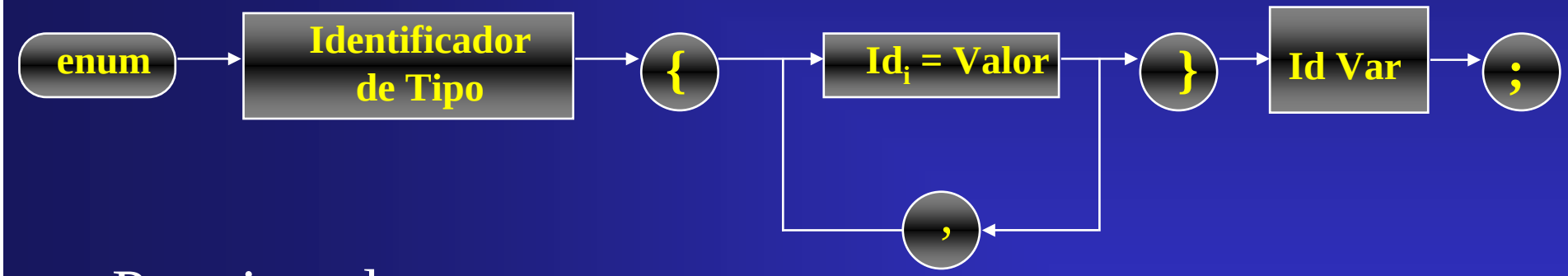
Son los tipos de datos básicos disponibles en *Borland C*
Cada tipo de dato está compuesto por un conjunto de valores distintos ordenados.
Luego todos los operadores relacionales pueden aplicarse:
> >= < <= = <>

- El número de elementos contenidos en un tipo de dato se denomina cardinalidad. El número ordinal del primer valor en un conjunto es 0.
- En los tipo ordinal existen dos grupos principales: los estándar (internos al sistema) y los definidos por el usuario.



Enumerado

El diagrama de sintaxis para declaraciones es el siguiente:



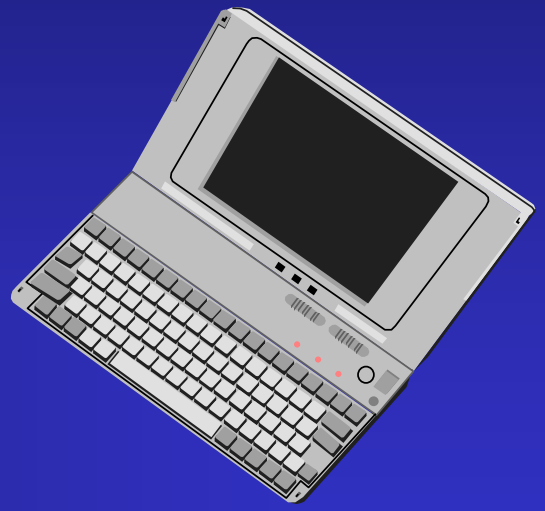
Por ejemplo:

enum DiaLab {Lunes, Martes, Miercoles, Jueves, Viernes}A, B;

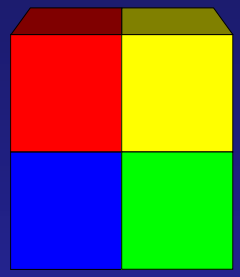
A menos que se indique, Id1 tomará el valor cero y los demás son consecutivos.

En los Tipos de Datos Complejos tenemos los siguientes:

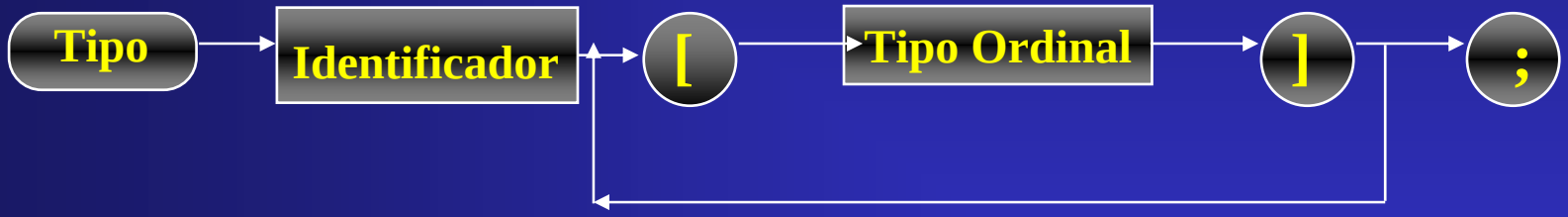
Arreglos
Estructuras
Uniones



Tipo Arreglo



Los arreglos poseen un número fijo de componentes de un tipo.



El tipo Ordinal, uno por cada dimensión, especifica el número máximo de elementos.

Arreglos

Vectores - Matrices

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.

La forma de declaración de un vector puede ser de esta forma:

Tipo *Identificador* [*Nº Máximo de Elementos*];

Ej: float A[10];

Arreglos

Vectores - Matrices

Si se quiere declarar una matriz se hace lo siguiente:

Tipo *Identificador* [Nº Máximo de Filas] [Nº Máximo de Columnas];

Ej: unsigned B[10][20];

si lo que se desea es declarar un vector de valores constantes lo que se hace es:

Tipo *Identificador* [*Nº máximo de Elementos*] = (*lista de valores separados por coma*);

Nota: El primer elemento de un arreglo, será el elemento cero.

Componente **StringGrid**

Se encuentra en la pestaña ***Additional*** de la Barra de Herramientas. Permite insertar en el formulario una rejilla que permite la escritura de múltiples valores, este componente es ideal para manipular la información de variables multidimensionadas (vectores y matrices).

Este componente de forma automática incorpora barras de desplazamiento horizontal y vertical en caso de ser necesarias.

Componente **StringGrid**

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 ejemplo:

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

Componente **StringGrid**

En este elemento es importante resaltar que en los subíndices primero se coloca la columna y luego la fila.

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***.

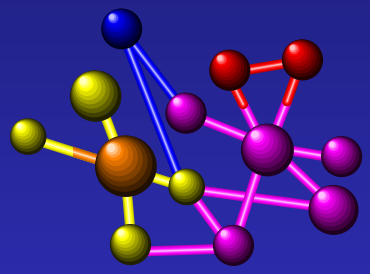
Arreglos Como Parámetro

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

```
Ej:
        {
          int A[10][10];
                funcion (A);
        }
void funcion (int A[ ][10])
{
```

Tipo Estructura

Se caracteriza por poseer más de un valor basado en los tipos de datos que lo conforman.



struct *Identificador_Estruc*{
Tipo1 *Var1;*
Tipo2 *Var2, Var3;*
} *Identificadores_de_Variables*;

Para accesar a los elementos de una estructura se utiliza el operador “punto”. Ejemplo:

```
struct ES {      int  A;
                 float  Z;
                 } M;
```

```
{
    M.A = 8
    M.Z = 7.3 }
```

Estructuras Como Parámetro

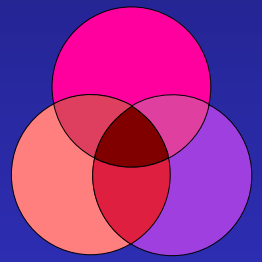
Las estructuras solamente se envían como parámetros a una función por dirección, ya su nombre contiene la dirección de la misma.

```
Ej: {struct ES {int  A;
                float  Z;} M;


       funcion (M);
          }
void funcion (ES B)
{
}
```

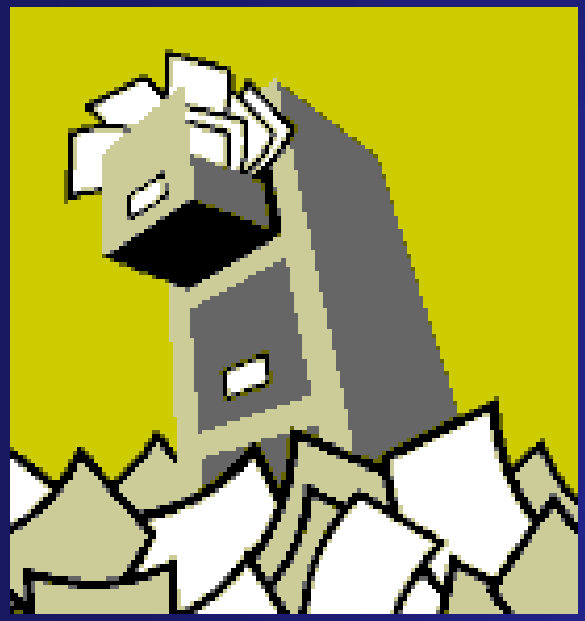
Tipo Unión

Es una posición de memoria compartida por varias variables de diferentes tipos, su sintaxis es:



union *Identificador_union* { Tipo1 *Var1*;
Tipo2 *Var2*;
} *Identificador_Variables*;

El tipo debe ser tomado de cualquier tipo predefinido. Para accesar a los elementos de la unión se utiliza el operador “punto”.



Archivos

Definición de Archivo

Muchas aplicaciones involucran el almacenamiento de grandes volúmenes de datos en forma semipermanente, ya que se desea retener los datos de una ejecución a otra o porque son muy numerosos para guardarlos en la memoria principal del computador.

Para almacenar estos datos se utiliza algún dispositivo periférico externo, también llamado memoria secundaria, el cual debe poder ser leído por el computador. A estos dispositivos se les llama **Archivos**.

Archivos

Un archivo se puede definir como un conjunto de informaciones almacenadas sobre un periférico de entrada/salida e identificado por un nombre.

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

Archivos Tipo Texto

Antes de que una variable tipo archivo ó tipo FILE sea empleada, ella debe ser asociada a un archivo externo mediante la Función: fopen( ).

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

fopen ( )

Asigna el nombre de un archivo externo a una variable tipo archivo o tipo FILE.

```
fe = fopen( "Nombre", "Modo");
```

De tal manera que todas las operaciones siguientes que utilicen ***fe***, trabajarán sobre el archivo externo identificado en *Nombre*. *Nombre* es un dato de tipo alfanumérico corto.

A continuación se muestran los ***"Modos"*** asociados al uso de archivos tipo Texto:

rt: Abre un archivo existente para leer información.

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

donde: *fe* es una variable tipo FILE que se ha de asociar al archivo externo "Nombre".

La información se comienza a leer a partir de la primera línea de datos.

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

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

donde: *fe* es una variable tipo **FILE** que se ha de asociar al archivo externo "Nombre".

Si ya existe un archivo con el mismo nombre, la información existente se pierde.

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

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

donde: ***fe*** es una variable tipo **FILE** que se ha de asociar al archivo externo "Nombre".

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

Una cualquiera de las funciones anteriores debe ser llamada antes de emplear las funciones de Entrada/Salida para archivos: **fprintf( )** y **fscanf( ).**

Que en su forma más general se invocan como:

```
fprintf ( fe , “Formato”, V1, V2)
fscanf ( fe , “Formato”, &V1, &V2)
```

fclose:

Cierra un archivo existente.

fclose (***fe***);

donde: ***fe*** es una variable tipo FILE que se ha de asociar a un archivo externo.

Cierra el archivo, y coloca la marca de finalización de archivo si este se abrió para escritura.

Formato del printf y scanf

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

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

- ✓% indica el sitio donde se escribirán los datos
- ✓Tam especifica el número de caracteres que contendrá el dato a escribir ó leer
- ✓.Prec nos dá la precisión
- ✓Tipo indica el tipo de dato de la variable
- ✓Var (1, 2, ..., N) es el nombre de la variable que se desea escribir

TIPO DEL ARGUMENTO

TIPO	SIGNIFICADO
d, I	Número entero tipo int
u	Número entero sin signo tipo unsigned int
f	Decimal punto flotante tipo float
c	Carácter sencillo tipo char. En scanf no salta los espacios en blanco, en este caso utilice %ls.
s	Cadena de caracteres tipo char*. En scanf agrega \0 al final
%	No es convertido a ningún argumento, imprime un porcentaje. En scanf no hace asignación alguna.

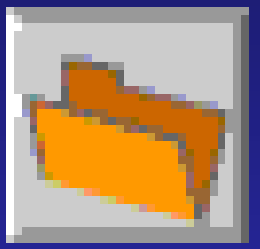
TAMAÑO DEL ARGUMENTO

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

CODIGOS DE BARRA INVERTIDA (ESCAPE)

CODIGO	SIGNIFICADO
\f	Salto de página
\n	Salto de línea
\t	Tabulación horizontal
\”	Comillas dobles
\’	Comilla simple
\0	Carácter nulo
\\	Barra invertida
\v	Tabulación vertical
\o	Constante octal
\x	constante hexadecimal

Escribir datos en un archivo ó crear un archivo:



Ejemplo:

```
fe = fopen (“nombre.dat”, “wt”);
fprintf(fe, “%d %f \n%d”, a, b, c);
fclose (fe);
```

La variable ***fe*** fue declarada tipo FILE. Las variables a, b, y c fueron declaradas tipo int.

Los datos se agregan al principio del archivo.

Agregar datos a un archivo:

Ejemplo:

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

La variable ***fe*** fue declarada tipo FILE. Las variables a, b, y c fueron declaradas tipo int.

Los datos se agregan al final del archivo.

Leer datos de un archivo:

Ejemplo:

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

La variable ***fe*** fue declarada tipo FILE. Las variables a, b, y c fueron declaradas tipo int.

Los datos se comienzan a leer desde el principio del archivo.

feof (Función)

Devuelve un valor distinto de cero si el apuntador se encuentra al final del archivo.

```
feof ( fe );
```

donde: ***fe*** es una variable tipo FILE que se ha de asociar a un archivo externo.

Leer datos de un archivo utilizando la marca de fin de archivo:

Ejemplo:

```
fe = fopen("nombre.dat", "rt");
while (!(feof(fe))) {
   fscanf(fe,"%d %d %d", &a, &b, &c);
   }
fclose (fe);
```

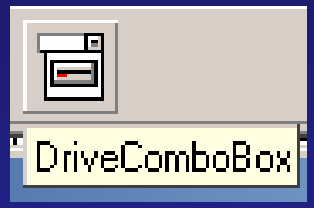
Creando una caja de Diálogo de Directorios

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

Para lo anterior se dispone de los componentes:

- **DirectoryListBox**
- **DriveComboBox**
- **FileListBox**

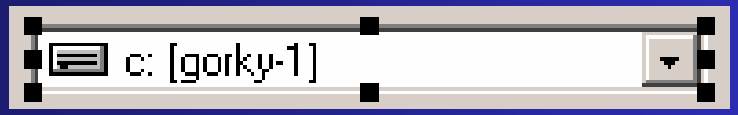
De todas maneras debe conocer que también es posible emplear cajas estándar de Windows.



DriveComboBox

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

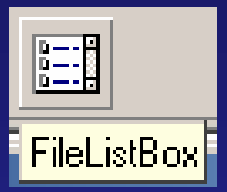
Permite presentar las unidades y conexiones de red disponibles en el equipo que se ejecuta la aplicación.



DirectoryListBox

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

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

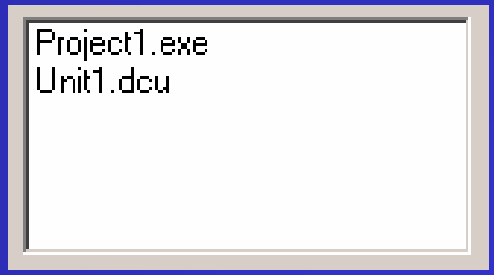


FileListBox

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

Presenta los archivos de un subdirectorio.

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



Caja de Diálogo de Directorios

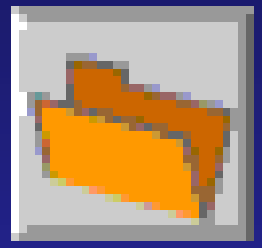
Una vez colocados los elementos anteriores sobre la forma se debe escribir las siguientes líneas de código:

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

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

Por Ejemplo:

para abrir un archivo para escribir datos:



```
fe = fopen (“nombre.dat”, “wt”);
fprintf(fe, “%d %f \n%d”, a, b, c);
fclose (fe);
```

La variable ***fe*** fue declarada tipo FILE. Las variables a, b, y c fueron declaradas tipo int.

Los datos se agregan al principio del archivo.

Cuadros de Diálogo Comunes

FILE OPEN

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

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

MENUS Y

FORMULARIOS

Controles de Edición

Componente **MaskEdit**

Es el componente ***Edit*** con un filtro o máscara de entrada.

Este control valida el texto que el usuario ingresa contra el código valido de la mascara, también da formato al texto que es ingresado.

Se ubica en la pestaña ***Additional*** de la barra de componentes.

Controles de Edición

Existen otros controles que también permiten que el usuario ingrese información:

Componente **Memo**

Es un control de edición de múltiples líneas apropiado para solicitar información de gran longitud.

Componente **RichEdit**

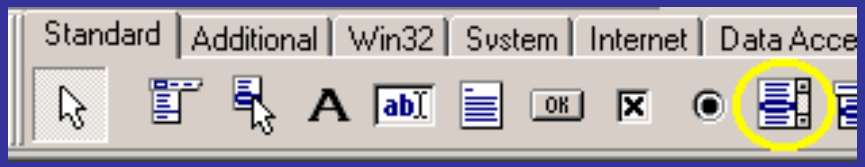
Está basado en el control Win32 de edición rica en texto, permitiendo cambiar fuentes, usar negritas, cursivas, modificar párrafo, etc; lo anterior es ofrecido por el programador ya que no existe opciones accesibles por el usuario.

Componente **BitBtn**

Es el botón anterior pero extendido para mostrar un mapa de Bits, los cuales pueden ser predefinidos o pueden ser de uno mismo.

Se activa en la pestaña ***Additional*** de la barra de componentes.

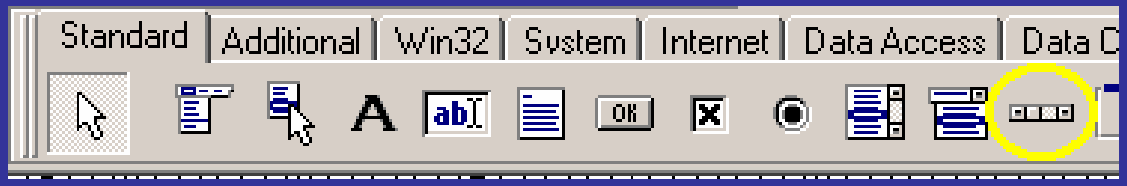
Componente **ListBox**



Representa un cuadro de lista estándar de Windows, el cual presenta una lista de opciones de entre las cuales puede elegir el usuario. Si el cuadro de lista posee más elementos de los que puede mostrar a la vez, aparecen barras de desplazamiento que permiten ver el resto.

Se ubica en la pestaña ***Standard*** de la barra de componentes.

Componente **ScrollBar**

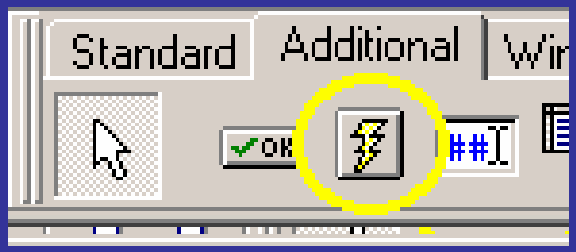


Presenta una barra de desplazamiento independiente (al no estar atado a un control de edición).

Su accionar se determina usando las propiedades ***Min*** y ***Max***, para asentar sus valores extremos. La posición de la barra de desplazamiento puede establecerse u obtenerse mediante la propiedad ***Position***.

Se ubica en la pestaña ***Standard*** de la barra de componentes.

Componente **SpeedButton**



Es un botón para ejecutar comandos o poner a punto modos.

A diferencia de los anteriores no es un componente de ventana ya que se diseñó para construir barras de herramientas, estando limitado a no poder recibir el enfoque de entrada ni el de tabulación.

Tiene comandos que le permiten presentar imágenes que simbolizan su actividad; pueden ser agrupados en arreglos de actividades comunes.

Trabajar con varios formularios

Una vez que ha estado trabajando sobre un formulario y necesita un segundo, lo que se debe hacer es insertar un nuevo formulario en el proyecto, para lo anterior elija: ***File | New Form*** y en el objeto por Ud. Elegido respondiendo a un evento determinado inserte el siguiente código:

```
form2.showmodal;
```

Cuando ejecute el programa le dará un mensaje de error indicando que el segundo formulario no está en la lista de USES, acepte y Builder lo incorporará, vuelva a ejecutar.

Trabajar con varios formularios

Con lo explicado en la lámina anterior se presenta el segundo formulario en estado modal, lo que implica que no puede ser empleado ningún otro formulario del proyecto hasta que se cierre el actual.

Para cerrar el formulario se puede presionar el botón de cierre del mismo (de estar activado), o escribiendo la siguiente instrucción

```
ModalResult = mrCancel;
```

Trabajar con varios formularios

Si se desea trabajar con una variable declarada en otra función se deben hacer varios pasos:

- ✓La variable se debe declarar
- ✓En la otra unidad debe agregar la primera.

Para hacer referencia a un componente, primero se coloca el nombre de la unidad, flecha (->), el nombre del componente y su propiedad.

Inserción de Menús

Los menús son un elemento importante en la mayoría de las aplicaciones en Windows. A continuación se indicarán los pasos básicos para poder insertar un menú en un formulario:

Coloque un componente **MainMenu** (primer componente en la pestaña *Standard* de la paleta de componentes). El trabajo realmente lo hacen los sub-componentes.

Se hace doble click en el componente **MainMenu** y aparece lo que se llama **Menu Designer**, que se ve como un formulario en blanco en espera de que sea ensamblado el menú.

Inserción de Menús

Para ir creando los elementos del menú ponga un identificador en la propiedad ***Name***, luego en ***Caption*** coloque la etiqueta a ser leída (si quiere subrayar alguna letra en particular anteponga el carácter &).

En este punto puede con el cursor indicar si quiere agregar nuevos elementos al menú, abajo o a un lado, en cualquier caso repetirá los pasos previos.

Verá que al mismo tiempo sobre la forma se va duplicando el menú que se va creando en el **Menu Designer**, cuando cierra este puede asignar el código a los elementos creados en el menú.