45

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE CIENCIAS

PROGRAMACION ORIENTADA A EVENTOS Y GEOMETRIA ANALITICA

T E S I S

QUE PARA OBTENER EL TITULO DE:

A C T U A R I O

P R E S E N T A :

JORGE ANGEL ITURBIDE VILLALBA

Acompañado de un C.D.

DIRECTOR DE TESIS: DR. CARLOS HERNANDEZ GARCIADIEGO.

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

MAT. MARGARITA ELVIRA CHÁVEZ CANO
Jefa de la División de Estudios Profesionales de la
Facultad de Ciencias
Presente

Comunicamos a usted que hemos revisado el trabajo de Tesis:

"Programación orientada a eventos y geometría analítica"

realizado por Jorge Angel Iturbide Villalba

con número de cuenta 8620430-5 , pasante de la carrera de Actuaría

Dicho trabajo cuenta con nuestro voto aprobatorio.

Atentamente

Director de Tesis Propietario	Dr. Carlos Hernández Garciadiego	
Propietario	M en C. Ma. Guadalupe Elena Ibargüengoitia González	
Propietario	M en C. Elena de Oteyza de Oteyza	
Suplente	M en C. Javier García García	
Suplente	M en C. Emma Lam Osnaya	

Consejo Departamental de Matemáticas

M. en C. José Antonio Flores Díaz

DEDICATORIAS

Esta tesis está dedicada principalmente a mis padres

Quienes me han dado TODO, y aunque no les he sabido agradecer y reconocer esa totalidad completamente, esto quiero que sea el principio de lo que quiero comenzar a compensar. No tengo palabras con que agradecer todo esto que he recibido. *Muchas Gracias.*

A mi hermano

Quien ha sabido estimular de cierta forma que siga avanzando cada vez más y no pueda quedarme estático.

AGRADECIMIENTOS

También quiero agradecer a todas las personas (familiares y amigos) que me han apoyado y/o soportado no solo en la licenciatura.

A todos mis parientes (aunque algunos ya no están presentes) que me han proporcionado apoyo de TODO tipo aun desde antes de haber generado esta tesis, pues si no ha sido por esos *eventos* no hubiera llegado a este nivel ni lo hubiera concluido.

A mis amigos que me llegaron a destrabar cada que llegué a tener contingencias en algún momento.

Me abstengo de decir nombres, pues por mi pésima memoria podría omitir a alguien y me incomodaría de no mencionarle.

A *todos* los profesores que me dieron cultura y conocimientos. Viéndolo todo aun desde el origen y aunque no se puedan enterar. ***Muchas gracias***. Pero muy en especial al Dr. Carlos Hernández Garciadiego, por todo lo obtenido casi desde que ingresé a la Facultad, mientras estuve en el servicio social y por la tesis. No tengo palabras para agradecerle.

Indice

Introducción

Como ya se conoce, existen varios paquetes matemáticos que hacen hasta un desarrollo en el ámbito diferencial (Calculus, Derive, etcétera), pero no todos soportan el nivel actual del software en el Sistema Operativo Windows.

Ya existen algunos que sí lo hacen, tales como MathCAD o Maple. El problema es que no son en español, y aunque a nivel licenciatura ya los alumnos deben ser, si no bilingües, por lo menos tener nociones de comprensión de lectura del inglés técnico-básico.

Por otro lado, los pocos programas que existen para Windows funcionan como intérpretes, lo cual obliga al usuario a teclear sin errores las instrucciones requeridas para efectuar una tarea; implicando con esto que se tengan que memorizar una gran cantidad de comandos.

Si bien algunos paquetes proporcionan el análisis de la misma, éste es mínimo. Ahora bien, muchos de estos programas funcionaban solo bajo el sistema de MS-DOS, lo cual no permite que haya mucha interacción entre el usuario y el paquete.

La forma de interacción con estos programas. es que el usuario proporciona la ecuación mediante el mecanismo diseñado para tal efecto al programa, y este responde desplegando la gráfica en pantalla

Otro detalle dentro de estos paquetes en español o inglés, es que ninguno ha hecho el detalle en dar todas las características de datos complementarios con las distintas formas geométricas; es decir, dar las ecuaciones de sus directrices, proporcionar las coordenadas de los focos, etcétera.

Y como la mayoría de los alumnos tampoco tienen un dominio del uso de la paquetería o la computadora.

Por otro lado, también existen diferentes estilos de programación: Programación No Estructurada, Programación Estructurada, Programación Orientada a Objetos (OOP) y Programación Orientada a (o Basada en) Eventos.

Esta última es de las más sencillas, es Programación Estructurada usando objetos ya creados.

Sobre la base de lo antes señalado, este trabajo se divide en sí en varias partes:

- Describir y comparar muy vagamente los clásicos y principales estilos de programación actualmente usados.
- Mostrar una introducción al lenguaje Visual Basic versión 4, que fue con la que se creó al programa.
- Mostrar cada subdivisión de lo correspondiente a la Geometría Analítica Plana mencionada en el temario, en el programa GAWin.EXE.

Con respecto a los diferentes estilos de programación, se puede responder a varias preguntas:

- *¿Qué es la Programación Modular y Estructurada?*
- *¿Qué es la Programación Orientada a Eventos?*
- *¿Qué es la Programación Orientada a Objetos (Object-Oriented Programming, OOP)?*
- *¿Cuál es la diferencia entre la Programación Basada en Objetos y la OOP?*
- *¿Por qué se decidió utilizar el Lenguaje de Programación Visual Basic para Windows?*

Para la introducción al Lenguaje Visual Basic en forma introductoria para principiantes:

- *Introducción al Lenguaje de Programación Visual Basic para Windows, versión 4.0.*
- *Palabras reservadas*
- *Formas, Objetos y Eventos*
- *Cómo generar un programa.*
- *Un ejemplo simple.*

Y las subdivisiones del programa se muestran en cada diferente capítulo posterior a esta introducción en el archivo de ayuda *GAWin.HLP*.

Capítulo 1

Estilos de programación usuales.

Cuando comenzaron a surgir lenguajes de programación que permitían resolver problemas específicos mediante computadora, al ser esta una máquina relativamente lenta y costosa, era muy importante diseñar los programas de manera que, aunque se tardase mucho tiempo en elaborarlos, el tiempo de ejecución en la computadora no fuese excesivo.

Se planteaba, además, un grave problema cuando un programador diferente del autor de un programa tenía que modificarlo o corregirlo, debido a que cada uno lo realizaba según su criterio, sin modelo o estructura estándar alguna.

Todo esto indujo a intentar crear un modelo de programación que aportase las siguientes características:

- Adaptarse fácilmente a modificaciones.
- Propiciar una puesta a punto más rápida.
- Posibilitar una cómoda interpretación de los programas por diferentes programadores.

Para conseguir dichos propósitos se establecieron dos criterios generales de programación, conocidos con el nombre de *Programación Modular* y *Programación Estructurada*, que se desarrollarán en los apartados siguientes.

En la sección 1.1 se presenta una respuesta resumida de lo que es la programación modular y estructurada, que es algo ya común dentro de todos los actuales estilos de programación avanzados. En la sección 1.2 está un resumen breve a la programación orientada a eventos, explicando que es un evento y mostrando los más comunes, que se han de mostrar en la sección 2.2. En la sección 1.3 se da un breve resumen a la programación orientada a objetos, que ya es la más actual de todas, y que lleva dentro un estilo estructurado de programación. Y para resumir el porqué de la tesis, y está la sección 1.4, que responde sencillamente a cual es la diferencia entre la programación basada en objetos (o lo que es lo mismo, orientada a eventos), y la programación orientada a objetos.

1.1 ¿Qué es la Programación Modular y Estructurada?

Para aumentar la eficacia de la programación se necesita que los programas tengan una estructura fácil de interpretar, de modo que los haga más comprensibles, manejables y fiables.

Desde sus comienzos, la programación ha estado gobernada por varias metodologías. En cada punto crítico en la evolución de la programación se creaba un nuevo enfoque para ayudar al programador a manejar programas cada vez más complejos. Los primeros programas se crearon mediante un proceso de cambio de los conmutadores del panel frontal de la computadora. Obviamente, este enfoque solo es adecuado para programas pequeños. A continuación se inventó el lenguaje ensamblador que permitió escribir programas cada vez más largos. El siguiente avance ocurrió en los años 50's cuando se inventó el primer lenguaje de alto nivel (FORTRAN).

Mediante un lenguaje de *alto nivel*, un programador estaba capacitado para escribir programas que tuvieran una *longitud de varios miles de líneas*. Sin embargo, el método de programación usado en el comienzo era un enfoque que *no solucionaba mucho*. Mientras que esto está bien para programas relativamente cortos, se convierte en "*código espagueti*" ilegible y difícil de tratar cuando se aplica a programas más largos. La eliminación del código espagueti se consiguió con la creación de los *lenguajes de programación estructurados* en los años sesenta. Estos lenguajes incluyen Algol y Pascal. Los programas estructurados se basan en estructuras *de control bien definidas*, bloques de código, la ausencia (o mínimo uso) del GOTO, y subrutinas independientes que soportan recursividad y variables locales. Mediante la programación estructurada un programador puede crear y mantener programas de una longitud superior a 50 mil líneas. En la actualidad, la mayoría de los programadores ha estudiado las técnicas de programación estructurada y toma como un hecho los principios arquitectónicos introducidos por la programación estructurada; y presenta un conjunto de principios para la comprensión de los sistemas de software que son el resultado de años de observación y estudio de las técnicas de construcción de programas.

La *Programación Estructurada* es un criterio de programación basado en el *Teorema de la Estructura* de Bohn y Jacopini que dice lo siguiente: "*Todo programa propio, es decir, con un solo punto de entrada y un solo punto de salida, puede ser escrito utilizando únicamente tres tipos de estructuras de control: estructura* ***secuencial****, estructura* ***condicional*** *y estructura* ***repetitiva***". (Como los detallaré más adelante).

Cualquier programa largo y complejo siempre se puede desarrollar mediante el anidamiento apropiado de estos tres tipos de estructuras.

La Programación Estructurada y la Programación Modular no son criterios contrapuestos, sino complementarios. El primero se encarga de desarrollar ***estructuradamente*** cada una de estas partes y el segundo tiende a dividir un programa en partes más pequeñas llamadas ***módulos***.

Los diagramas de flujo pueden ser de dos tipos.

- Diagramas de flujo del sistema.
- Diagramas de flujo de proceso.

A los diagramas de *Flujo del Sistema* también se les conoce como **Organigramas**. Los cuales describen el flujo de los datos y los diferentes soportes *físicos* que van a intervenir en la solución del problema.

Y a los diagramas de *Flujo de Proceso*, se les conoce más usualmente; con el nombre de **Ordinogramas**, *cuya utilidad es describir mediante símbolos gráficos, las acciones del algoritmo elegido para la resolución del problema*. Para su elaboración, se utilizan diferentes símbolos.

Se han utilizado los ordinogramas para hacer descripciones gráficas de algoritmos. Este procedimiento es útil cuando los algoritmos son sencillos, pero a medida que aumenta su complejidad, aumenta también la dificultad para representarlos gráficamente, ya que el tamaño de los ordinogramas crece desmesuradamente y sus líneas de conexión se complican en exceso.

Por esta razón, paralelamente a la programación estructurada, surge una nueva forma de descripción estructurada de algoritmos llamada **pseudocódigo.**

PSEUDOCÓDIGO.

Es una herramienta utilizada en el diseño de algoritmos para resolver problemas que, empleando los principios de la programación estructurada, permite expresar el flujo de ejecución de las instrucciones de una forma clara, sin ambigüedad alguna y usando el lenguaje materno.

El pseudocódigo no es un lenguaje de programación, es tan sólo una forma de diseñar la solución a un problema; de manera que su traducción posterior a un lenguaje de programación de alto nivel sea sencilla

Describiendo las estructuras básicas en pseudocódigo, se verán a continuación las tres *estructuras* básicas de control, en las que se apoya la programación estructurada representadas en pseudocódigo, y sus correspondientes ordinogramas:

ESTRUCTURA SECUENCIAL.

Una estructura secuencial es la que ejecuta las acciones una a continuación de otra en la que se realizan todas y en el orden en que están escritas, sin posibilidad de omitir ninguna de ellas.

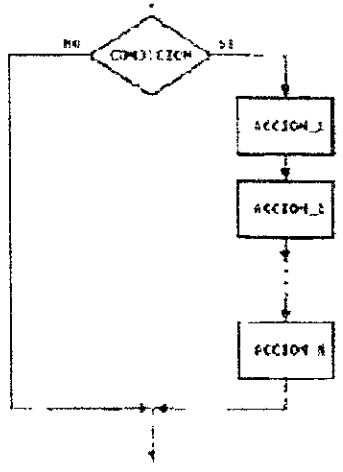

```
Accion_1
Accion_2
Accion_n
```

ESTRUCTURA CONDICIONAL.

Es la que realiza un conjunto u otro de instrucciones dependiendo de que una determinada condición se cumpla o no. Existen tres tipos de estructuras condicionales:

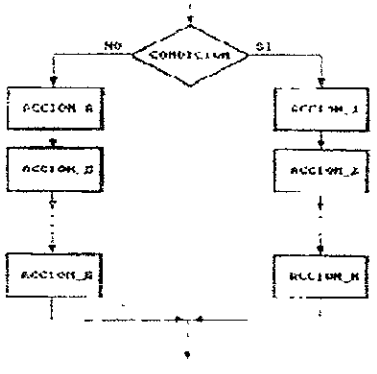
- **Estructura Condicional Simple.**

Se evalúa la condición y, sólo si es cierta, se ejecuta el conjunto de acciones correspondiente.



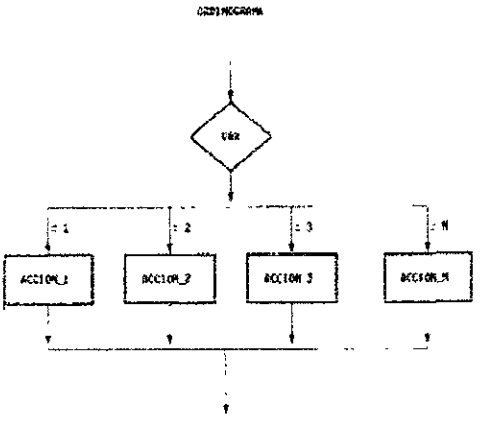
- **Estructura Condicional Doble.**

Es aquella que dependiendo de la certeza o falsedad de una condición se ejecutará un bloque de instrucciones u otro.



- **Estructura Condicional Múltiple.**

  Dependiendo del valor natural que tome la variable numérica implicada en la condición que vaya a evaluarse, se ejecutará una de las *n* acciones correspondientes.



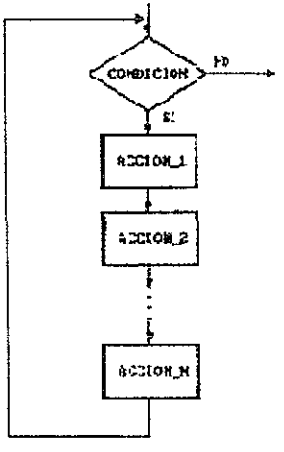
ESTRUCTURA REPETITIVA.

Esta estructura permite la *repetición* de un bloque de instrucciones dependiendo de la veracidad o falsedad de una condición.

- **Estructura Tipo MIENTRAS.**

  El bloque de acciones se repetirá *mientras* que la condición sea cierta.

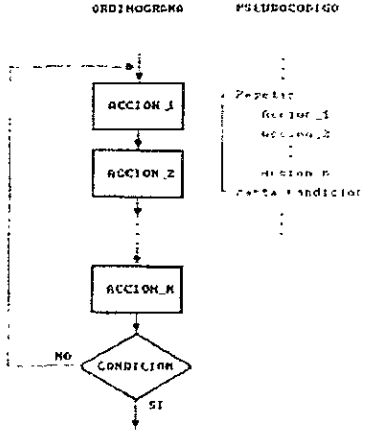
  La condición se evalúa al *comienzo* de la estructura. Esto implica que el bloque de instrucciones puede no ejecutarse ninguna vez si la condición de salida es inicialmente falsa.



- **Estructura tipo HASTA.**

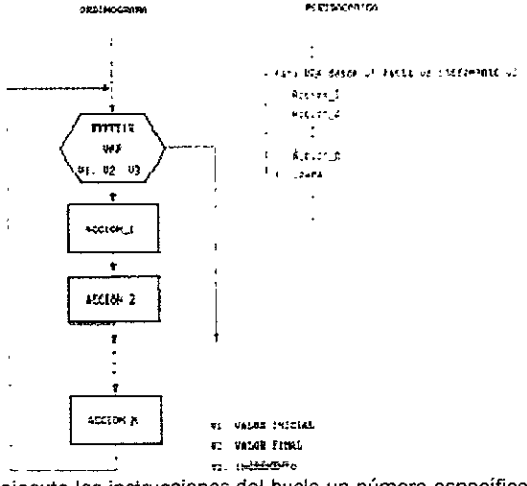
  En esta ocasión, el bloque de acciones se repetirá *hasta* que la condición sea cierta.

  Dicha condición se evalúa al *final* de la estructura. Esto implica que el bloque de instrucciones se ejecutará al menos una vez, aunque la condición de salida ya sea cierta al entrar en dicha estructura



- **Estructura tipo PARA.**

  Si el número de iteraciones (repeticiones) es fijo o se conoce de antemano, se puede utilizar una estructura tipo *Para* o *Desde* en lugar de utilizar una estructura tipo Mientras.



Esta estructura ejecuta las instrucciones del bucle un número específico de veces y controla *automáticamente* el número de repeticiones.

Para dicho control, hay que definir dentro de la estructura el nombre de una variable

numérica, su valor inicial, su valor final, y un incremento fijo. Es la propia estructura la que se encarga de inicializar la variable con el valor indicado, incrementarla en cada iteración y, cuando sobrepasa el valor final especificado, terminar la repetición del bucle. La evaluación de la condición de salida se hace al comienzo de cada iteración.

Toda estructura Para puede también realizarse con una estructura Mientras. La diferencia radica en que en la estructura Mientras hay que realizar, mediante instrucciones, la inicialización de la variable que controla el bucle, así como su incremento; y en la estructura Para esto se hace automáticamente.

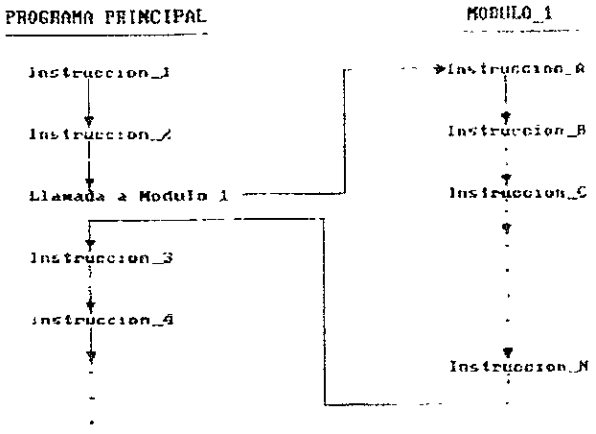
De lo anterior se puede deducir que toda estructura Para se puede escribir como una estructura Mientras, pero no todas las estructuras Mientras se pueden escribir como una estructura de tipo Para; tan solo aquéllas cuya condición de salida venga determinada por una variable numérica con incremento fijo.

Estas fueron las principales estructuras que *crean* a un ***programa estructurado***.

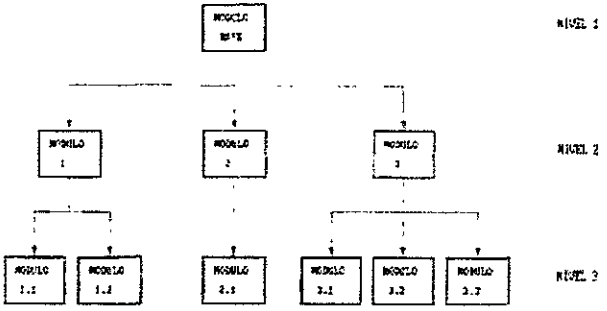
En lo que se refiere más a la ***programación modular***, ésta técnica consiste en dividir un programa en partes bien diferenciadas lógicamente, llamadas *módulos*, que puedan ser analizadas y programadas por separado.

Un *módulo* se puede definir como un conjunto formado por una o varias instrucciones lógicamente enlazadas.

A cada módulo se le asigna un nombre, para poder identificarlo. Cuando en un punto de un programa se referencia un módulo, el programa le cede el control para que se ejecuten todas sus instrucciones. Finalizado el mismo, el control se devuelve al punto del programa desde donde se llamó al módulo, y se continúa con la ejecución de la instrucción siguiente a la que realizó la llamada. El orden de ejecución es así:



Si un módulo es lo suficientemente grande o complicado puede subdividirse en otros módulos, y éstos, a su vez, en otros, pudiéndose representar un programa un programa con un diagrama similar al siguiente.



Aquí se observa que siempre debe existir un *módulo raíz* o *principal*, que es el encargado de controlar y relacionar a todos los demás.

Si los módulos que componen un programa son parte integrante del mismo, se les llama *rutinas*, *subrutinas*, *procedimientos* o *funciones*. Si por el contrario, son programas independientes, reclamados desde otros programas, se les denomina *subprogramas*.

No hay una norma fija para dividir un programa en módulos. Se hace al criterio del programador. Sin embargo, se deben seguir unas normas más o menos generalizadas:

- Cada módulo sólo puede tener un punto de entrada y otro de salida.
- El módulo principal debe ser conciso, mostrando claramente los módulos que lo componen y su relación. Es decir, debe indicar la solución completa del problema.
- Los módulos deben tener la máxima independencia entre ellos.
- Un módulo debe representar por sí mismo una estructura lógica coherente y resolver una parte bien definida del problema.

La programación modular es, pues, una técnica que se basa en el desarrollo de programas de lo general a lo particular, o dicho de otra forma, del conjunto al elemento. (*Diseño TOP-DOWN*). Se comienza considerando *qué* funciones debe realizar el programa desde un punto de vista muy general, dándolas como resueltas y dejando su diseño (*cómo*) para un paso posterior. De esta manera se avanza hasta llegar al máximo nivel de detalle.

Si dentro de un módulo aparece una función cuyo desarrollo completo queremos posponer, este lo representaremos indicando en ese punto el nombre que tendrá el módulo que lo desarrollará, ocupándonos posteriormente de dicho desarrollo.

La programación modular aporta una serie de ventajas frente a la programación convencional:

- Los programas son más sencillos de escribir y depurar, pues se pueden hacer pruebas parciales con cada uno de sus módulos.
- La corrección o modificación de un módulo se hace más cómoda y, en general, no tiene por

qué afectar al resto de los módulos

- Un programa se puede ampliar fácilmente con solo diseñar los nuevos módulos necesarios.
- Un mismo módulo escrito una sola vez puede ser referenciado desde varios puntos del programa, evitando la repetición de instrucciones ya escritas.

1.2 ¿Qué es la Programación Orientada a Eventos?

Los eventos son sucesos que pueden ser detectados por los distintos elementos u objetos que componen las aplicaciones (en este caso, en Visual Basic; son los formularios, controles, etc.). Así, dependiendo del tipo de *evento* ocurrido y del *objeto* que lo ha detectado, se ejecutará una u otra sección del código.

Los eventos pueden ser producidos por diferentes fuentes:

- El usuario: Cuando éste realiza un click con el Ratón, cuando pulsa una tecla, etc.
- La computadora: Cuando se produce un aviso del temporizador del sistema, errores de ejecución, etc.
- La aplicación: Cuando una instrucción de la misma invoca directamente un evento.
- Otras aplicaciones: Cuando se produce un intercambio de datos con otras aplicaciones *Windows* que se estén ejecutando concurrentemente.

Las secciones del código que se ejecutan en función del evento ocurrido y del objeto que lo ha detectado son pequeños subprogramas denominados *procedimientos de evento.* Estos procedimientos deben ser escritos por el programador en lenguaje *Basic*, y deben contemplar las acciones que deben realizarse como respuesta a dicho evento.

Un *procedimiento general* le dice a la aplicación cómo realizar una tarea específica. Una vez que se ha definido el procedimiento, éste debe ser explícitamente *llamado* por su aplicación.

Cuando un objeto -como en Visual Basic- reconoce que un evento ha ocurrido, éste automáticamente invoca al procedimiento del evento con el nombre que corresponde al evento. Porque el nombre establece una asociación entre el objeto y el código, los procedimientos del evento son dichos para ser unidos a controles y formas.

Por otro lado, un *procedimiento de evento* permanece inactivo hasta que se le llama para responder a los eventos causados por el usuario (click o doble click del Ratón, oprimir una tecla) o por el sistema (un evento de cronómetro o carga).

En resumen:

Un **evento** es una acción reconocida por una *forma* o un *control*.

- Todos los objetos consisten en una serie de eventos predefinidos.
- Cuando se produce un evento, se ejecuta el código asociado con el procedimiento correspondiente

Las entradas en los programas basados en Windows se forman por varios eventos:

- Las teclas activan los eventos.
- El Ratón también sirve como activador de eventos cuando se presiona algún botón.
- Los eventos mismos provienen de otras ventanas, menús y botones.

La programación convencional se desarrolla con base en procedimientos:

⇒ La *inicialización* permite preparar la pantalla, activar las variables o abrir los archivos.

⇒ El *ciclo* espera la llegada de cualquier carácter proveniente del teclado o la lectura de los registros de un archivo. Una vez procesado dicho carácter o registro, el programa retoma el modo de espera de la siguiente instrucción.

⇒ La *finalización* realiza las funciones opuestas a la inicialización. En esencia, se trata de una verificación del estado en que se encuentra el sistema.

En las aplicaciones tradicionales y de procedimientos, la aplicación misma controla cada parte del código que se ha de ejecutar. La ejecución inicia con la primera línea de código y sigue un camino predefinido en donde la aplicación activa todos los procedimientos necesarios.

En la programación por eventos los procedimientos se ejecutan por la acción de un evento específico, lanzado por el usuario o el sistema. El orden de ejecución de los procedimientos depende de los eventos, y estos últimos provienen de las acciones realizadas por el usuario. Esta es la esencia tanto en los sistemas GUI (*Graphical User* Interface o Interfaz Gráfica del Usuario) como en la programación por eventos. Lo anterior significa que el usuario tiene el control y el código obedece a sus acciones.

Windows es un sistema basado en los eventos generados por el usuario. Tal es el caso de las herramientas de desarrollo conocidas como RAD (Rapid Application Development) que están teniendo una amplia aceptación en el mercado de los ambientes PC. Estas herramientas tienen su origen en las herramientas de fabricación de prototipos y en los entornos de programación ***visual***. Prometen dos ventajas importantes: un ciclo de desarrollo más corto y más flexible permitiendo saltar directamente del prototipo a la aplicación terminada; la segunda ventaja consiste en que un usuario final razonablemente sofisticado puede desarrollar aplicaciones. Sin embargo tienen sus limitaciones: requieren de un entendimiento profundo de los requerimientos del negocio o empresa, no eliminan la necesidad de un diseño fuerte y de hábil programación. En gran medida las bondades de las herramientas RAD se derivan del uso de la metodología de objetos. Si su aplicación intercepta un evento destinado a la misma, se ejecuta el código correspondiente. Cuando el evento en cuestión no tiene el reconocimiento de la aplicación, el sistema activa el Administrador de eventos. Estos son algunos eventos comunes en Visual Basic: *Click* (hacer clic), *DblClick* (hacer doble clic), *Load* (cargar), *Unload* (descargar), *KeyPress* (presionar tecla), *Resize* (redimensionar), *Paint* (pintar), *DragDrop* (arrastrar y soltar), *DragOver* (arrastrar), *GotFocus* (enfocar), *KeyDown* (mantener presionada la tecla), etcétera.

1.3 ¿Qué es la Programación Orientada a Objetos (Object-Oriented Programming, OOP)?

La vasta mayoría del software que se desarrolla en la actualidad está siendo elaborado con técnicas y lenguajes de programación tradicionales, sin el beneficio de los objetos. Fuera de la comunidad académica y de investigación, son pocos los esfuerzos que se realizan para apoyar a los programadores para que usen los lenguajes orientados a objetos y determinen su factibilidad para proyectos grandes y complejos. Esta poca disposición a emigrar es alimentada por la existencia de inversiones considerables en tecnologías tradicionales de software, pero sobre todo, por la falta de entendimiento de las técnicas orientadas a objetos. Hará falta una nueva generación de programadores para aceptar completamente el uso de objetos. Los lenguajes tradicionales con extensiones de objetos dominaron el mercado durante 90's, aunque un mercado significativo para lenguajes no tradicionales y más "puros" en el sentido de orientación a objetos, tales como Smalltalk y Eiffel se desarrollará lentamente.

La programación orientada a objetos es una nueva manera de enfocar la programación, y aunque la programación estructurada nos ha llevado a excelentes resultados cuando se ha aplicado a programas moderadamente complejos, llega a fallar en algún punto cuando el programa alcanza un cierto tamaño. Para poder escribir programas de mayor complejidad se necesitaba un nuevo enfoque en la tarea de programación. A partir de este punto se inventa la Programación Orientada a Objetos (*Object-Oriented Programming, OOP*), que es otro paso evolutivo en el diseño y construcción de software; y fue uno de los primeros sistemas de programación en que se reconoció que el cambio y la evolución de un programa no sólo son inevitables, sino deseables. La OOP busca minimizar el impacto del cambio a través de la técnica de *encapsulado*. También apoya la reutilización de código ya existente y hace que ésta sea práctica Pues aunque los lenguajes estructurados reconocen como meta la reutilización de código, no han podido producir códigos susceptibles de reutilizarse con facilidad.

Aún cuando se citan las bibliotecas en lenguajes estructurados como medios para reutilizar código, la estabilidad necesaria de las bibliotecas es un impedimento para adaptarlas de manera fácil a problemas especiales de caso. La programación orientada a objetos supera esta restricción a través de los *nexos dinámicos*. La OOP simplemente formaliza prácticas que los buenos programadores ya utilizan con los *lenguajes estructurados*. Si ya se sabe que se debe intentar limitar el empleo de *variables globales*, o al menos aislar y agrupar los procedimientos que comparten datos, ya se están practicando muchos de los principios de la OOP. Con los objetos, la codificación en esencia se traduce en la construcción de procedimientos o *métodos* · Es simplemente el empacado de los componentes de programa lo que resulta diferente. *La OOP*

requiere que se describan en forma explícita las relaciones entre los procedimientos y datos compartidos. La diferencia con la OOP estriba en que se subraya por igual la importancia de datos y acciones. La programación estructurada tiende acentuar las acciones. Funciones y procedimientos se consideran como las unidades básicas de *construcción*. Esta perspectiva da origen a la técnica de refinamiento descendente por pasos (top-down stepwise refinement). Las acciones abstractas se descomponen en pasos cada vez más concretos hasta que cada uno de ellos pueda especificarse como una proposición en lenguaje de programación. Sin embargo, muchos problemas se resuelven mejor trabajando de abajo hacia arriba (bottom up) o en ambas direcciones al mismo tiempo, o incluso utilizando un tercer sistema en el cual la estructura ni siquiera tiene una parte superior –***sistemas manejados por eventos***. Si se quiere ser más estricto acerca del significado exacto de la OOP, es aquella programación por medio del envío de mensajes a objetos. Todos los objetos de la colección comparten ciertas características –por ejemplo: el conocimiento de su ubicación en pantalla y la capacidad para cambiar de ubicación, activarse o desactivarse. Desde la perspectiva del programador, la selección de un objeto dentro de la colección, proporciona información acerca del tipo exacto del objeto.

Los ambientes de manejo de ventanas, como Microsoft Windows y el escritorio del ambiente de programación Smalltalk, proporcionan ejemplos en los que el tipo preciso de un *objeto* no siempre se conoce. Por ejemplo, los objetos que se presentan en la pantalla podrían ser: *barras de desplazamiento, ventanas, iconos, menús, botones* u otro tipo de *controladores*. Cada uno de estos objetos responde a los mensajes que envía el usuario en forma de teclas oprimidas o clics del Ratón; y cosas distintas sucederán dependiendo de cuál es el objeto al que se apunta cuando se envía un mensaje. Por ejemplo, si el apuntador del Ratón está orientado para minimizar, cuando se oprima el botón izquierdo del Ratón; la ventana o aplicación se transformará en icono. Por otra parte, si el apuntador del Ratón está dirigido a la sección de guardar archivo de un menú de procesador de palabras, entonces, el archivo que en ese momento se estaba editando se guardará en disco.

Los objetos en el escritorio, como los que se acaban de describir, se conocen como ***objetos polimórficos***. Esto significa que comparten ciertas características comunes como la capacidad para colocarse en una sola colección (por ejemplo un escritorio) y la capacidad para responder a los mismos mensajes. El polimorfismo está relacionado de manera directa con la *herencia*. La programación por herencia le permite definir nuevos objetos al describir la forma en que difieren de objetos ya existentes. ***Herencia y polimorfismo*** son bastante complejos y requieren la comprensión de algunos conceptos preliminares.

La OOP toma las mejores ideas incorporadas en la programación estructurada y las combina con nuevos y potentes conceptos que permiten organizar los programas de forma más efectiva. Permite descomponer un problema en subgrupos relacionados. Cada subgrupo pasa a ser un objeto autocontenido que contiene sus propias instrucciones y datos que se relacionan con ese

objeto. De esta manera, la complejidad se reduce y el programador puede tratar programas más largos.

Pero, ¿qué es un objeto? Cuando el código y los datos están enlazados de ésta manera (***encapsulación***); se ha creado un ***objeto***. *Un objeto es una pieza que tiene propiedades y una conducta específica.* Los objetos se comportan como los tipos de datos, pero ellos son normalmente mucho más complejos. El acceso a los objetos es a través de *propiedades, métodos y eventos*. Las propiedades son las variables y las constantes de los objetos. Al cambiar el valor de las propiedades se cambia el estado del objeto. Los Métodos son los comandos ó funciones que están asociadas con el objeto. Alguna operación que se desee llevar a cabo sobre un objeto, será hecha en conjunción con sus métodos. *En Visual Basic los objetos pueden ser formas, ventanas, botones, animaciones,* etc.

El concepto de *objeto* es a la vez simple y poderoso: *cada objeto conoce información y sabe como operar con ella.*

Mensajes y Métodos.

A diferencia de los sistemas tradicionales, en el enfoque de la OOP los objetos son elementos del software que tienen la posibilidad de actuar unos con otros. Esta interacción entre objetos se logra a través del envío de *mensajes* pidiendo que se ejecute un *método* específico. Así, un método describe una secuencia de acciones a realizar por un objeto; el método puede considerarse análogo a un procedimiento o una función en Pascal, la diferencia radica que en lugar de formar parte de un gran programa está incluido en el objeto. El objeto que envía un mensaje se conoce como *transmisor* y al que lo recibe como *receptor*. Este receptor determina por sí mismo que método (es decir, comportamiento o manipulación) debe llevar a cabo. La estructura del objeto protege sus descripciones al resto del sistema. El programador sabe que un objeto puede realizar una función particular, pero no los detalles de cómo la realiza. De esta forma la abstracción de datos es natural y hay ocultación de la información.

Clases e Instancias.

Las *clases* son plantillas para objetos. Las clases son como los tipos en un lenguaje tradicional de programación. La diferencia es que los usuarios pueden definir nuevos tipos en un lenguaje de OOP. Entre los tipos con los que la mayoría de los programadores están familiarizados se encuentran enteros, caracteres, números de punto flotante y cadenas (en Pascal) Si una clase es como un tipo en un lenguaje tradicional de programación, entonces un objeto es como una variable. Se hace referencia a los objetos como *instancias* de una clase. Una definición de clase por sí misma no crea ningún objeto. Sin embargo, la definición de clase puede utilizarse como un tipo para crear un objeto. A las *clases* algunas veces se les denominan *tipos definidos por el usuario.*

Todos los lenguajes de OOP, comparten tres características además. *encapsulación, polimorfismo y herencia.* Analizando estos conceptos:

Encapsulación.

La encapsulación es el mecanismo que agrupa el código y los datos que maneja y los manipulan protegidos frente a cualquier a cualquier interferencia y mal uso. En un OOP, el código y los datos pueden empaquetarse de la misma forma en que se crea una unidad autocontenida. Dentro de la caja son necesarios tanto el código como los datos. Cuando el código y los datos están enlazados de esta manera, se ha creado un *objeto*. En otras palabras, un objeto es el dispositivo que soporta encapsulación.

En un objeto, los datos y el código, o ambos, pueden ser *privados* o *públicos* para ese objeto. Los datos o el código privado sólo los conoce y son accesibles por el objeto. Es decir, una parte del programa que está fuera del objeto no puede acceder al código o a los datos privados. Cuando el código o los datos son públicos, otras identidades del programa pueden acceder a ellos. Normalmente las partes públicas de un objeto se utilizan para proporcionar una interfaz controlada a las partes privadas del objeto.

Para todos los propósitos, un objeto es una variable de tipo definido por el usuario. Puede parecer extraño que un objeto que enlaza código y datos se pueda contemplar como una variable. Sin embargo, en OOP, este es precisamente el caso. Cada vez que se define un nuevo objeto, se está creando un nuevo tipo de dato. Cada instancia específica de este tipo de dato es una variable compuesta.

Diciéndolo de otra forma, la encapsulación es uno de los términos fundamentales en el paradigma de la orientación a objetos. Se refiere a las cualidades privadas de un objeto y exclusivas para ese objeto, otro objeto aún de la misma clase puede no tener acceso a esos atributos y/o métodos *privados* del primer objeto. De esta forma la comunicación dentro de un conjunto de objetos se da exclusivamente por medio de mensajes explícitos. La encapsulación es un mecanismo que permite ocultar los detalles de la representación interna de un componente. La encapsulación preserva la integridad de un objeto, las implementaciones internas pueden estar cambiando, pero la interfaz visible para los otros objetos permanece constante. En conjunto con la abstracción, la encapsulación permite definir interfaces entre los aspectos internos y externos del objeto. Proveen un poderoso mecanismo para especificar objetos que sean independientes unos de otros.

Polimorfismo.

Polimorfismo (del griego, cuyo significado es "muchas formas") es la cualidad que permite que un nombre se utilice para dos o más propósitos relacionados, pero técnicamente diferentes. El propósito del polimorfismo aplicado a la OOP es permitir usar un nombre para especificar una clase general de acciones. Dentro de una clase general de acciones, la acción específica a aplicar está determinada por el tipo de dato en concreto que la invoca.

De forma general, el concepto de polimorfismo es la idea de "una interfaz de múltiples métodos". Esto significa que es posible diseñar una interfaz genérica para un grupo de actividades

relacionadas. Sin embargo, la acción específica ejecutada depende de los datos. La ventaja del polimorfismo es que ayuda a reducir la complejidad permitiendo que la misma interfaz se utilice para especificar una *clase general de acción*. Es trabajo del compilador seleccionar la *acción específica* que se aplica a cada situación. El programador no necesita hacer esta selección manualmente. Sólo necesita recordar y utilizar la interfaz general. El polimorfismo se puede aplicar tanto a funciones como a operadores. Prácticamente todos los lenguajes de programación contienen una aplicación limitada de polimorfismo cuando se utiliza con los operadores aritméticos. El punto clave a recordar sobre el polimorfismo es que permite manejar complejidades más grandes a través de la creación de interfaces estándar para actividades relacionadas.

En sí, es la habilidad de una operación de tomar formas diferentes cuando es aplicado a objetos diferentes. La idea que subyace en el concepto de polimorfismo es que operaciones semánticamente equivalentes, deben presentar la misma interfaz aún cuando están siendo aplicadas a diferentes objetos.

Como se ha mencionado arriba, los objetos actúan en respuesta a los mensajes que reciben; sin embargo un mismo mensaje puede originar comportamientos completamente diferentes al ser recibido por objetos diferentes. A esto se le conoce como *polimorfismo*. El concepto de polimorfismo es tal vez el más difícil de entender pero en él radica la clave de la flexibilidad de los sistemas de software orientados a objetos. El polimorfismo establece que un objeto que envía mensajes a otro no necesita saber exactamente la clase a la que pertenece el segundo objeto, simplemente que el objeto puede entender el mensaje que está recibiendo.

Herencia.

La herencia es el proceso mediante el cual un objeto puede adquirir las propiedades de otro. Más en concreto, un objeto puede heredar un conjunto general de propiedades a las que pueden añadir aquellas características que son específicamente suyas. La herencia es importante porque permite que un objeto soporte el concepto de *clasificación jerárquica*. Mucha información se hace manejable gracias a la clasificación jerárquica. Por ejemplo, pensando en la descripción de una casa. Una casa es parte de una clase general llamada *edificio*. A su vez, *edificio* es una parte de la clase más general *estructura*, que es parte de la clase aún más general de objetos que se puede llamar *obra del hombre*. En cualquier caso, la clase hija hereda todas las cualidades asociadas con la clase padre y le añade sus propias características definitorias. Sin el uso de clasificaciones ordenadas, cada objeto tendría que definir todas las características que se relacionan con él explícitamente. Sin embargo, mediante el uso de la herencia, es posible describir un objeto estableciendo la clase general (o clases) a las que pertenece, junto con aquellas características específicas que le hacen único. Como ya se ve, la herencia juega un papel muy importante en la OOP.

Ahora bien, las clases de los objetos deben definirse de tal forma que forman una estructura *jerárquica*. Objetos definidos a un nivel más bajo de la jerarquía heredan los atributos de los

objetos de niveles superiores. Este es uno de los aspectos más importantes del modelo de orientación a objetos. Esta característica permite definir, en los objetos de más alto nivel de la jerarquía, comportamientos que se aplicarán en niveles más bajos. La clase de mayor nivel es la superclase mientras que la clase inferior es la subclase. Usualmente la subclase hereda los datos y los procedimientos de una superclase, estas características heredadas pueden modificarse al redefinir los datos o los métodos en las subclases.

Abstracción.

La *abstracción* es un proceso cognoscitivo por medio del cual se identifican las características relevantes para cierto fin. Las metodologías orientadas a objetos se caracterizan por asociar abstracciones del mundo real con componentes computacionales que reflejan las características de la abstracción. El hecho de que los lenguajes orientados a objetos permitan a los programadores usar objetos definidos previamente como *clases abstractas*

Para resumir los principios básicas de la Programación Orientada a Objetos:

Estructura de Programa.

- Los objetos son principalmente unidades estructurales.
- Los objetos son tareas o procesos.
- Un programa es un conjunto de objetos que se comunican.

Objetos.

- Los objetos agrupan datos y procedimientos. O sea, *Objeto = Código + Datos.*
- Los objetos recuerdan información (datos).
- Los objetos procesan información (procedimientos).
- Los objetos tienen un ciclo vital.
- Cada objeto de una clase tiene su propia copia de cada variable declarada dentro de la clase.

Clases.

- Las clases son plantillas para los objetos.
- Las clases son el mecanismo para crear objetos.
- Los objetos se crean a partir de especificaciones de clases.
- Las clases son tipos.
- Una declaración de clase es una abstracción logica que define un nuevo tipo que determina como será un objeto de ese tipo. Una declaración de *objeto* crea una entidad física de ese tipo. (Es decir, un *objeto* ocupa espacio de memoria, pero una definición de clase no).

Comunicación.

- Los objetos envían y reciben mensajes.
- Los mensajes llevan información a través del parámetro.
- Los mensajes invocan procesamiento.
- Los mensajes son llamadas de procedimiento o funciones.

Procesamiento.

- Los procedimientos contenidos en el interior de los objetos son métodos.
- Los métodos procesan información.
- Los métodos son código ejecutable.

1.4 ¿Cuál es la diferencia entre la Programación Basada en Objetos y la OOP?

La diferencia básica entre la programación basada en objetos y la orientada a objetos, es que la que está basada en objetos, no tiene todas las características para poder ser un lenguaje orientado a objetos (como el polimorfismo), mas sin embargo utiliza los objetos ya creados.

Las instrucciones que componen el código del programa sólo se ejecutan cuando ocurren ciertos sucesos en la computadora, denominados *eventos*. Los eventos son sucesos que pueden ser detectados por los distintos elementos u objetos que componen las aplicaciones (formularios, controles, etcétera) Asi, dependiendo del tipo de evento ocurrido y del objeto que lo ha detectado, se ejecutará una u otra sección del código.

La que está basada en objetos no crea a los objetos a utilizar, sino que "los clona" de un objeto ya establecido, sin tener que hacer construcción y destrucción de ellos.

Por decir otro ejemplo dentro de este caso, si se utilizara algo orientado a objetos, con decir $d(X, Y)$, se sabe que es la distancia de X a Y. Si son puntos, usa el estilo del teorema de Pitágoras calculando la raíz del cuadrado de la diferencia entre unos valores Y si Y es recta, *cambia la forma* de la fórmula y hace que se vuelva del estilo "distancia de un punto a una recta" Este polimorfismo, puesto que cambió la fórmula, hace que sea diferente el estilo de programación.

Capítulo 2

Generalidades del Lenguaje Visual Basic 4

El gran "Bang" que tienen ahora los lenguajes visuales se atribuye a que estos tipos de lenguajes tienen una gran facilidad de programabilidad, y podemos encontrar desde Visual C, Visual Java. Visual Fox, y por supuesto Visual Basic. El continuo desarrollo que ahora exigen los programas pueden ser creados con plataformas como Visual Basic, que permiten desarrollar programas desde vídeo juegos hasta potentes bases de datos, por lo que podemos decir que Visual Basic es de una interfaz amigable, fácil de aprender para los usuarios y fácil de manipular para los desarrolladores. Por el lado del mercado podemos ver que cada día las empresas buscan programadores expertos en Visual Basic por lo confiable y rápido que resulta este lenguaje y no es de extrañarnos que este lenguaje de programación poco a poco esté desplazando a otros visuales como C. Es increíble lo que se puede lograr con un programa Visual tal como Visual Basic que es, un lenguaje de programación orientado a eventos como ya antes lo hemos mencionado y creo que esta es la clave por la cuál ha tenido gran difusión en el medio tecnológico, aunque como ya sabemos Visual Basic fue creado de una amalgama de todos los lenguajes del viejo Basic y aunque esto sorprenda también de C.

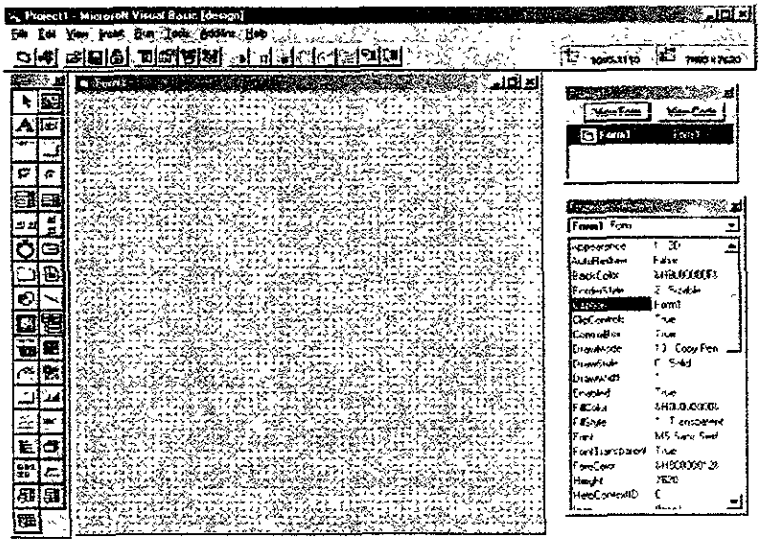
Visual Basic es una de las herramientas más modernas del mercado para el desarrollo de aplicaciones Windows. Permite realizar aplicaciones potentes y flexibles de forma rápida y sencilla, gracias a la facilidad de manejo de su entorno de desarrollo, que también es una aplicación para Windows. La versión 4.0 de Visual Basic está orientada al desarrollo de aplicaciones de 32 bits para Windows NT y Windows 95, Windows 98 o Windows 2000; pero también es posible crear aplicaciones de 16 bits para las versiones anteriores de Windows.

Este capítulo intenta mostrar una visión general del lenguaje, mostrando detalles de cómo es, y generando un ejemplo. Para esto el capítulo se divide en 4 partes básicas, en la sección 2.1 se muestra una forma general de cómo es el lenguaje y las partes que lo componen. En la sección 2.2 están la mayoría de las palabras reservadas, dependiendo del tipo de función que tenga, ya sean eventos, funciones, etcétera. En la sección 2.4 se muestra cómo generar un programa y sus pasos para hacerlo. De esta sección se muestra además una parte en especial, la 2.4.1, que es un ejemplo simple y el algoritmo para realizarlo. Finalmente, la sección 2.5 dice el porqué se prefirió utilizar este lenguaje a otro más complejo.

2.1 Introducción al Lenguaje de Programación Visual Basic para Windows, versión 4.0.

Para esta "introducción", se está suponiendo que ya se tiene instalado el compilador de esta versión en el disco duro y se conoce el manejo básico de Windows 95 o superior.

Cuando se ejecuta Visual Basic por primera vez, se despliega su Entorno de Desarrollo, obteniéndose una pantalla similar a la mostrada en la figura siguiente:



Las ventanas mostradas son:

- La *Ventana principal*: Es el soporte del sistema de menús de Visual Basic. Contiene la *Barra de menús* y la *Barra de Herramientas*.

- La *Barra de menús*: Contiene los diferentes menús desplegables con opciones que representan comandos para manejar los archivos, editar los programas, ejecutar y depurar las aplicaciones, suministrar ayuda, etcétera.

El acceso a dichas opciones o comandos puede realizarse tanto con el ratón como con el teclado. Cada uno de los distintos menús del entorno de desarrollo de *Visual Basic* agrupa los comandos relacionados con un determinado tema. La misión global de cada uno de estos menús se muestra en la siguiente tabla.

MENÚ	DESCRIPCIÓN
Archivo (File)	Contiene comandos relacionados con el manejo de los distintos archivos de disco que intervienen en la creación de las aplicaciones
Editar (Edit)	Contiene comandos relacionados con el manejo del editor de textos de Visual Basic, que permite escribir las instrucciones del código de las aplicaciones.
Ver (View)	Contiene opciones relativas a la visualización de los distintos elementos del entorno de desarrollo.
Insertar (Insert)	Permite ejecutar comandos para la inserción de nuevos elementos en el proyecto.
Ejecutar (Run)	Contiene comandos relativos a la ejecución y depuración de las aplicaciones desde dentro del entorno de desarrollo.
Herramientas (Tools)	Contiene comandos que permiten acceder a diversas utilidades y opciones del entorno de desarrollo.
Complementos (Add-Ins)	Contiene comandos que permiten añadir y acceder a una serie de herramientas externas, que aportan nuevas características y facilidades al entorno de desarrollo.
Ayuda (Help)	Contiene opciones para la obtención de ayudas interactivas sobre todos los aspectos de Visual Basic.

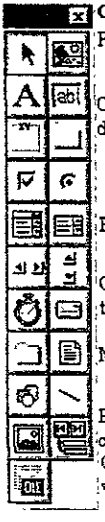
- La *Barra de herramientas*: Proporciona un acceso rápido a las operaciones que se realizan con mayor frecuencia mediante la pulsación de distintos botones con iconos descriptivos.

Estos botones permiten acceder de una forma rápida a los comandos más frecuentemente utilizados en los menús del entorno de desarrollo. El acceso a algunos de estos comandos, además de realizarse haciendo clic sobre el botón correspondiente, también puede realizarse con el teclado, ya que muchos de ellos disponen de teclas de método abreviado. La función de cada uno de estos botones y su tecla de método abreviado correspondiente es la siguiente.

- *Formulario*: Crea un nuevo módulo de formulario y lo añade al proyecto
- *Módulo*: Crea un nuevo módulo estándar y lo añade al proyecto.
- *Abrir Proyecto*: Carga un proyecto previamente salvado en el entorno de desarrollo ([*CTRL + O*]).
- *Salvar Proyecto*: Salva el proyecto actual y todos los archivos que lo componen.
- *Bloquear Controles*: Bloquea y desbloquea todos los controles de formularios en su posición actual.
- *Editor de menús*: Visualiza o activa la ventana del Editor de menús ([*CTRL + E*]).
- *Propiedades*: Visualiza o activa la ventana de propiedades ([*F4*]).

- *Examinador de objetos*: Visualiza o activa la ventana del Examinador de objetos ([*F2*]).
- *Proyecto*: Visualiza o activa la ventana de proyecto ([*CTRL + R*]).
- *Iniciar*: Ejecuta la aplicación correspondiente al proyecto actual, pasando a modo de ejecución. ([*F5*]).
- *Interrumpir*: Detiene la ejecución del proyecto temporalmente, pasando a modo de interrupción ([*CTRL + PAUSA*])
- *Terminar*: Finaliza la ejecución del proyecto, volviendo al modo de diseño.
- *Alternar puntos de ruptura*: Sitúa o elimina un punto de ruptura o *breakpoint* en la línea de código donde se encuentra actualmente el cursor ([*F9*]).
- *Inspección instantánea*: Visualiza el valor actual de la expresión de inspección seleccionada ([*SHIFT + F9*)].
- *Llamadas*: Visualiza la lista de las llamadas a procedimientos en curso, es decir, aquellas que no han concluido ([*CTRL + L*]).
- *Paso a paso por* instrucciones: Ejecuta la siguiente instrucción del código y vuelve al estado de interrupción ([*F8*]).
- *Paso a paso por procedimientos*: Realiza la misma operación que el anterior, aunque ejecutando los procedimientos como una sola unidad ([*SHIFT + F8*]).

- La *Caja de herramientas*: Contiene un conjunto de iconos representativos de los distintos controles (objetos) disponibles para ser utilizados en la aplicación. Haciendo alusión a cada uno de estos objetos, tomándolos de izquierda a derecha y de arriba hacia abajo:



CONTROL	NOMBRE	DESCRIPCION
Puntero	*Pointer*	Este icono no representa un control, sino que permite mover y altera el tamaño de los controles y formularios ya agregados
Cuadro de dibujo	*PictureBox*	Visualiza dibujos e imágenes gráficas procedentes de mapas de bits, iconos o meterchivos de Windows. También visualiza texto y gráficos generados con sus propios métodos. Además, puede actuar como contenedor de otros controles.
Etiqueta	*Label*	Visualiza textos que no pueden ser modificados por el usuario.
Cuadro de texto	*TextBox*	Visualiza textos que pueden ser modificados por el usuario. Permite al usuario introducir texto, editarlo, seleccionarlo, etcétera.
Marco	*Frame*	Actúa como contenedor de otros controles, creando agrupaciones de controles tanto visuales como funcionales.
Botón de comando	*CommandButton*	Permite al usuario ejecutar una determinada acción o comando.
Casilla de verificación	*CheckBox*	Permite al usuario indicar un estado del tipo verdadero/falso o *sí/no*. Cuando se utiliza en grupo, permite al usuario seleccionar múltiples opciones
Boton de Opcion	*OptionButton*	Cuando se utilizan
Cuadro combinado	*ComboBox*	Combina un cuadro de texto

Cuadro de lista	*ListBox*	Visualiza una lista de elementos
Barra de desplazamiento horizontal	*HscrollBar*	Permite al usuario desplazarse horizontalmente sobre documentos, listas y otros tipos de información que sean demasiado extensos para ser visualizados de una sola vez.
Barra de desplazamiento vertical	*VscrollBar*	Realiza la misma función que el control anterior, pero en sentido vertical.
Cronómetro	*Timer*	Genera eventos a intervalos fijos del reloj del sistema.
Cuadro de lista de unidades	*DriveListBox*	Visualiza una lista con las unidades de disco válidas, permitiendo al usuario seleccionar una.
Cuadro de lista de directorios	*DirListBox*	Visualiza una lista de directorios y rutas de acceso, permitiendo al usuario seleccionar uno.
Cuadro de lista de archivos	*FileListBox*	Visualiza una lista de archivos, permitiendo al usuario seleccionar uno
Forma	*Shape*	Dibuja una figura geométrica con una determinada forma (rectángulo, cuadrado, óvalo o círculo).
Línea	*Line*	Dibuja una línea con un determinado grosor y estilo de trazado.
Imagen	*Image*	Visualiza una imagen gráfica procedente de un mapa de bits, un icono o un metarchivo de Windows.
Datos	*Data*	Proporciona un método de acceso a la información almacenada en una base de datos
Contenedor OLE	*OLE Container*	Permite vincular e incrustar objetos procedentes de otras aplicaciones.

- La *Ventana de formulario*: Es el soporte para la creación del interfaz de la aplicación con el usuario. Sobre ella se irán situando los distintos elementos que formarán parte de la aplicación (controles, gráficos, textos, etcétera).

  Esta ventana es en sí la `forma` principal sobre la que se incrustan los objetos para programar con eventos. Al aparecer en el programa por primera vez o en un proyecto nuevo, su figura es así:

- La *Ventana de proyecto*: Contiene la lista de archivos o módulos que intervienen en la creación de la aplicación. A este conjunto de archivos se le denomina *Proyecto*.

  El acceso a la ventana de proyecto puede realizarse seleccionando la opción *Proyecto* del menú *Ver*, haciendo clic sobre el botón *Proyecto* de la barra de herramientas o pulsando [*CTRL* + *R*].

Los proyectos pueden estar compuestos de diversos tipos de archivos: módulos de formulario, módulos de formularios MDI, módulos estándar, módulos de clases, controles personalizados, objetos insertables y archivos de recursos.

- La *Ventana de propiedades*: Contiene la lista de los atributos (propiedades) del formulario o control que se encuentra actualmente seleccionado. Visualiza también los valores actuales de dichos atributos y permite alterarlos.

  Permite acceder a las propiedades de todos los controles que se encuentran situados en el formulario actualmente seleccionado, incluidas también las del propio formulario. Se compone de dos elementos, la *Lista de objetos* y la *Lista de propiedades*, cuyo funcionamiento ya fue descrito anteriormente.

  El resultado de la alteración de los valores de las propiedades de los objetos se refleja automáticamente en el aspecto y el comportamiento de estos, tanto en la fase de diseño como en tiempo de ejecución. Esta figura muestra unas propiedades con una forma de ventana larga pero delgada.

- La *Ventana de código*: Contiene el código de programa de la aplicación, permitiendo editar las instrucciones que componen cada módulo. Esta ventana no se encuentra visible inicialmente. Cuando se desea comenzar a programar, haciendo doble clic en esa nueva forma, comienza por mostrar un editor para la forma, que se ejecutará cuando se haga el evento de cargar ese objeto mencionado (la *forma*), ya que *ipso facto* converge hacia el, pues el evento que está mencionado en el Procedimiento, es *Load* (Cargar), que es cuando se carga o llama a esa forma. Y se realizará lo que se indique en ese procedimiento (*Sub*).

- Los *Menús contextuales*: Se trata de menús emergentes que se despliegan al hacer clic con el botón derecho del ratón sobre un objeto del Entorno de Desarrollo, y contienen comandos relacionados con el manejo de dicho objeto.

  Cuando se realiza un clic con el botón secundario del ratón sobre un elemento del entorno de desarrollo, Visual Basic despliega un menú emergente con los comandos y acciones más frecuentemente utilizados en relación con dicho elemento. Para ejecutar la opción, bastará con hacer clic sobre ella, utilizando el botón izquierdo del ratón.

  Los menús contextuales suelen contener una opción resaltada en negrita. Cuando se selecciona ésta opción, su efecto es siempre el mismo que el de realizar un doble clic sobre el elemento con el botón izquierdo del ratón.

- El *Examinador de objetos*: Visualiza las propiedades y métodos de los objetos disponibles para ser utilizados en el proyecto actual, tanto procedentes de Visual Basic como de aplicaciones externas.

  Este es un cuadro de diálogo que permite visualizar las características de los objetos disponibles para ser utilizados en el proyecto. Muestra las clases disponibles tanto en las librerías de objetos como en los distintos módulos que forman parte del proyecto, mostrando

también sus propiedades y métodos. Para acceder al examinador de objetos debe seleccionarse la opción *Examinador de objetos* del menú *Ver*, hacer clic sobre el botón Examinador de objetos (Object Browser) de la Barra de herramientas, o bien pulsar [F2].

En la parte superior de la ventana aparece una lista desplegable *Libraries/Projects* (Bibliotecas/Proyectos). Esta lista permite seleccionar entre las distintas librerías de objetos disponibles para la aplicación, ya sean procedentes de Visual Basic o de otras aplicaciones externas. Entre ellas, se incluye el propio proyecto actual.

Cuando se selecciona el proyecto actual en la lista desplegable *Libraries/Projects*, la lista *Classes/Modules* mostrará todos los módulos que integran dicho proyecto (módulos de formulario, módulos estándar y módulos de clase). Cuando se selecciona uno de estos módulos en la lista *Classes/Modules*, la lista *Methods/Properties* muestra todos los procedimientos, propiedades y métodos que han sido definidos en él.

Al seleccionar un elemento en alguna de las listas anteriores y hacer clic sobre el botón ? de la parte inferior, se muestra de forma automática la ayuda interactiva correspondiente a dicho elemento

Además, a la derecha de este botón siempre se visualiza un texto descriptivo del elemento que se encuentra actualmente seleccionado.

Así, por ejemplo, si se trata de una constante intrínseca se muestra su valor real, si se trata de una propiedad se muestra su tipo de datos, etcétera.

2.2 Palabras reservadas.

Las palabras reservadas del lenguaje Visual Basic versión 4.0, están generalmente conjuntadas sobre la base de lo que son. Principalmente estos son los conjuntos principales a mostrar:

- *Eventos*
- *Funciones*
- *Métodos*
- *Objetos y Colecciones*
- *Propiedades*
- *Declaraciones*

En sí, aún para la versión 4.0 es una lista larga. por lo que solamente mostraré las más comunes, pues aún faltarían otras misceláneas palabras como *Me*, *Beep*, los diferentes tipos de variables (Byte, Boolean, Integer, Long, Single, Double, Currency, Date, Object, String, Variant, Type) y otras más. Para más referencia, consultar el apendice A

2.3 Formas, Objetos y Eventos.

Las aplicaciones realizadas con Visual Basic se componen de un conjunto de entidades denominadas objetos. Los objetos pueden definirse como combinaciones de código y datos que pueden ser tratadas como una sola unidad. Por lo tanto, su manejo se realiza como unidades independientes.

Los principales *objetos* utilizados para la creación de aplicaciones con Visual Basic son los siguientes:

- Formularios (Form): Representan las distintas ventanas de que constará la aplicación. Un objeto *Forma* es una ventana o caja de diálogo que hace parte de una aplicación de interface de usuario.
- Controles (Control): Son los elementos gráficos que deben incluirse en cada formulario para permitir la interacción entre la aplicación y el usuario.

Los formularios junto con los controles que contienen, forman el interfaz de la aplicación con el usuario. *Visual Basic* dispone de un amplio repertorio de controles predefinidos, que permiten realizar diversas funciones. Ejemplos de controles son los botones de comando, los cuadros de texto, las listas, etcétera. cada uno de ellos está representado por un icono diferente en la caja de herramientas.

Los objetos, ya sean formularios o controles, poseen una serie de atributos, denominados *propiedades*, que definen tanto su aspecto como su comportamiento. Cada una de estas propiedades posee un nombre y un valor que permite indicar características del objeto tales como su color, el tamaño, la posición, etcétera. Estos valores se establecen en la fase de diseño de la aplicación, aunque también pueden ser modificados durante la ejecución de la misma.

Como ya se mencionó anteriormente, cada tipo de objeto posee también un repertorio de procedimientos de eventos, que le permiten responder a los eventos que es capaz de detectar El programador debe seleccionar aquellos eventos a los que debe responder el objeto y escribir el código necesario en su procedimiento de evento correspondiente.

Una *colección de formas* es una colección cuyos elementos representan cada forma cargada en una aplicación. La colección incluye las formas MDI de aplicaciones, formas MDI hijo y formas que no son MDI. Las *colecciones de formas* tienen una particularidad simple, **Count**, que especifica el número de elementos en la colección

Finalmente, los objetos poseen una serie de procedimientos propios, llamados *métodos*, que permiten manipularlos. Existen métodos para mover el objeto, para dibujar en su interior, etcétera. Para invocar un método en una sentencia es necesario escribir el nombre del objeto y del método, separados por un punto.

Así, por ejemplo. para escribir un texto sobre un formulario puede utilizarse el método *Print*. Si el tiene por nombre *Form1*, la instrucción correspondiente quedará de la siguiente manera

Form1.Print "Texto visualizado sobre el formulario"

2.4 Como generar un programa.

A la hora de crear una aplicación *Windows* con *Visual Basic*, es necesario crear un *proyecto*. Este proyecto contendrá todas las especificaciones necesarias para el funcionamiento de la aplicación. En esencia, los proyectos de *Visual Basic* están constituidos por un conjunto de archivos, denominados también *módulos*, que contienen las especificaciones de cada uno de los elementos que intervienen en la creación de la aplicación.

Cada proyecto se almacena en un archivo con la extensión .VBP, que contiene la lista de los archivos individuales que componen la aplicación. El archivo de proyecto también guarda las condiciones del entorno de desarrollo la última vez que éste se salvó.

Cada formulario se almacena en un archivo individual con la extensión .FRM. Este tipo de archivos se denominan *módulos de formulario*, y contienen las propiedades, métodos y procedimientos de evento del formulario, así como de todos los controles que éste contiene.

Visual Basic también permite incluir en los proyectos archivos que solamente contienen declaraciones y procedimientos, y que pueden utilizarse de forma compartida por el resto de los módulos. A este tipo de archivos se les denomina *módulos estándar* y tienen la extensión .BAS Los procedimientos que contienen se denominan procedimientos *generales*, con el fin de diferenciarlos de los de evento.

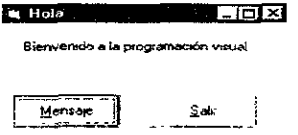
Los proyectos también pueden incluir otros tipos de archivos, como *módulos de clases*, archivos de controles personalizados y archivos de recursos, según se verá más adelante.

En resumen, el diseño de una aplicación con *Visual Basic 4* debe contemplar los siguientes pasos:

- Creación del proyecto asociado a la aplicación.
- Diseño de los formularios que componen la aplicación, y especificación de sus propiedades.
- Inserción de los controles de cada formulario, y especificación de sus propiedades.
- Escritura de los procedimientos de evento, tanto de los formularios como de los controles

2.4.1 Un ejemplo simple.

La aplicación consistirá en una ventana de dos botones Al hacer clic sobre el primero se visualiza un mensaje de bienvenida. El segundo simplemente finaliza la ejecución. Su aspecto final se muestra en la figura siguiente



Cuando se inicia Visual Basic por primera vez se crea un nuevo proyecto, denominado Project1, constituido por un único formulario, Form1 Para realizar el diseño de la interfaz será necesario dibujar los controles sobre dicho formulario, que inicialmente se encuentra vacío. Para esta aplicación se utilizará un control de una etiqueta y dos botones de comando. La siguiente figura muestra el aspecto de sus iconos representativos en la caja de herramientas, junto con una breve descripción de su misión

Para situar un control en el formulario deben realizarse los siguientes pasos.

1. Hacer clic sobre el control dentro de la caja de herramientas.
2. Mover el puntero del ratón (que pasará a tener forma de cruz) al formulario, situándolo en la posición donde se desea que quede la esquina superior derecha del control.
3. Manteniendo pulsado el botón izquierdo del ratón, desplazar el puntero hasta la posición donde se desea situar la esquina inferior izquierda del control. A esta operación se le denomina *Arrastrar*

Control	Icono	Nombre	Descripción
Etiqueta	A	**Label**	*Visualiza texto que el usuario no puede modificar*
Botón de comando		**Command Button**	*Desencadena una acción cuando se hace un clic sobre él.*

4. Liberar el botón del ratón.

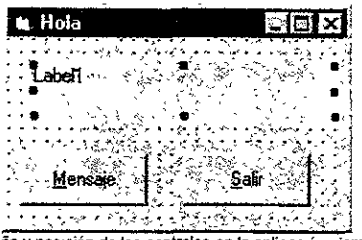
También puede realizarse esta operación con un doble clic sobre el control en la caja de herramientas. De esta forma, el objeto se sitúa en el centro del formulario con el tamaño por defecto.

Una vez situados los controles sobre el formulario es posible moverlos para cambiar su posición

Para ello, bastará con hacer clic sobre el control y arrastrarlo hasta la nueva posición, liberando entonces el botón. Igualmente, es posible cambiar el tamaño de un control mediante los siguientes pasos.

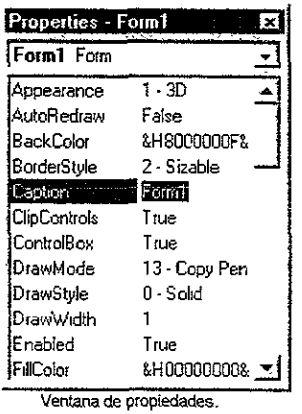
1. Seleccionar el control, haciendo clic sobre él Sobre sus lados y esquinas se mostrarán unos pequeños recuadros, denominados *Controladores de tamaño.*
2. Situar sobre un controlador el puntero del ratón, que pasará a tener una forma de flecha, indicando las direcciones posibles de movimiento.
3. Hacer clic sobre el controlador y arrastrar el lado o esquina hasta la posición deseada
4. Liberar el botón del ratón.

La figura siguiente muestra el tamaño y la posición de los controles para el caso de la aplicación ejemplo



Tamaño y posición de los controles en la aplicación ejemplo

La siguiente etapa en la creación de la aplicación consiste en dotar a los objetos de las propiedades deseadas. Esta operación se realiza sobre la Ventana de propiedades, cuyo aspecto se muestra en la siguiente figura, y está compuesta de los siguientes elementos:



Ventana de propiedades.

- La *Lista de objetos*: Contiene los nombres de todos los objetos situados en el formulario, incluido el propio formulario. El nombre mostrado es el del objeto a cuyas propiedades se está accediendo.
- La Lista de propiedades: Muestra los nombres y los valores actuales de todas las propiedades del objeto seleccionado.

Al hacer clic sobre el botón situado a la derecha de la Lista de objetos, se despliega dicha lista, permitiendo seleccionar el objeto a cuyas propiedades se desea acceder. Para establecer el valor de una propiedad de un control deben seguirse los siguientes pasos:

1. Acceder a la Ventana de propiedades, haciendo clic en el botón *Propiedades* de la Barra de herramientas, o pulsando *[F4]*.
2. Desplegar la Lista de objetos, haciendo clic en el botón derecho que aparece a su derecha, y seleccionar el nombre del control.
3. Seleccionar el nombre de una propiedad en la Lista de propiedades, haciendo clic sobre él

En la columna de la derecha se activará el *Cuadro de valores*, mostrando su valor actual.

4. Escribir el valor de la propiedad en el Cuadro de valores.

Al acceder a la Ventana de propiedades, se visualizan inicialmente las del control que se encuentra actualmente seleccionado. Algunas propiedades con un número limitado de valores permiten seleccionar el valor a partir de una lista. Esta lista de despliega haciendo clic en el botón que aparece a la derecha del Cuadro de valores. En estos casos, también es posible ir cambiando de valor cíclicamente, haciendo un doble clic sobre el nombre de la propiedad.

Otras propiedades permiten seleccionar su valor en cuadros de diálogo especializados, que se despliegan al hacer clic sobre el botón del Cuadro de valores. Así, las propiedades referentes al color despliegan una paleta de colores, las de tipo letra un cuadro de atributos de fuentes, las de nombres de archivos un cuadro de apertura de documentos, etcétera una vez seleccionado el valor éste pasa automáticamente al Cuadro de valores.

Para la aplicación ejemplo, será necesario especificar las propiedades que se muestran en la siguiente tabla.

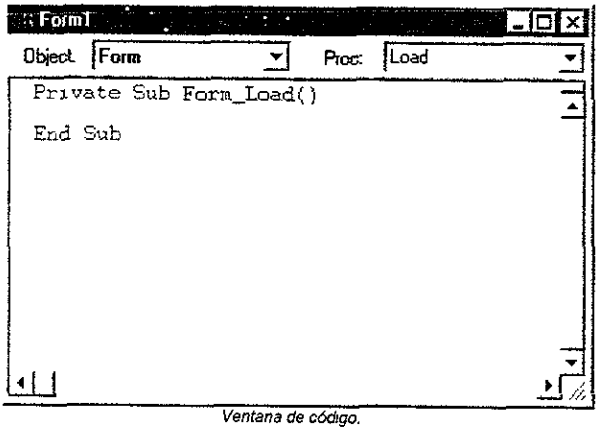
VALORES DE LAS PROPIEDADES DE LOS CONTROLES		
CONTROL	**PROPIEDAD**	**VALOR**
Form1	*Caption*	Hola
Command1	*Caption*	&Mensaje
Command2	*Caption*	&Salir
Label1	*Caption*	(Vacío)

La propiedad *Caption* de un control indica el texto que acompañará al mismo. En el caso de los formularios, será visualizado en su barra de títulos. En el caso de los botones de comando y de las etiquetas se mostrará en su interior.

Escritura del código

Para escribir el código de la aplicación es necesario acceder a la Ventana de código. Para ello, debe seleccionarse el módulo correspondiente en la ventana de proyecto y hacer clic en el botón Ver Código. También puede accederse a dicha ventana haciendo un doble clic sobre el formulario o control cuyo código se desea especificar

El aspecto de esta ventana se muestra en la figura siguiente, y consta de los siguientes elementos:



Ventana de código.

- La *Lista de objetos*: Contiene los nombres de todos los objetos del módulo actual, incluido el del propio formulario. El nombre mostrado es el del objeto a cuyo código se está accediendo
- La *Lista de procedimientos*: Contiene los nombres de todos los procedimientos de evento asociados al objeto seleccionado en la Lista de objetos. El nombre mostrado corresponde al evento cuyo código se está editando
- El *Área de edición*. Es la zona donde se muestran las líneas de código que componen el procedimiento de evento. Su funcionamiento es similar al de un editor de texto.

Para escribir el código de un procedimiento de evento deben seguirse los siguientes pasos:

-Desplegar la Lista de objetos de la Ventana de código, haciendo clic en el botón que aparece a su derecha, y seleccionar el nombre del objeto cuyo procedimiento de evento se desea editar.

-Desplegar igualmente la Lista de procedimientos, y seleccionar el nombre del evento.

-Escribir las instrucciones del código entre las líneas *Private Sub* y *End Sub* del Área de edición.

Para el caso del ejemplo de aplicación será necesario escribir el procedimiento de evento Click de los botones de comando. Estos procedimientos se ejecutarán cada vez que el usuario haga clic en el botón correspondiente.

```
Private Sub Command1_Click()
  Label1.Caption = "Bienvenido a la programación visual"
End Sub
Private Sub Command2_Click()
  End
End Sub
```

Como puede observarse, el primer procedimiento de evento se limita a asignar un texto a la

propiedad *Caption* del control etiqueta. El segundo simplemente ejecuta la sentencia *End*, que finaliza la ejecución de la aplicación.

Almacenamiento y ejecución.

Para salvar en disco el ejemplo de aplicación debe seleccionarse la opción *Guardar proyecto* (Save Project) del menú *Archivo* (File), o hacer clic en el botón *Guardar proyecto* (Save Project) de la Barra de herramientas. De esta forma se desplegará un cuadro de diálogo estándar de *Windows* para salvar documentos. En primer lugar debe indicarse el nombre del módulo de formulario, cuya extensión será FRM. Posteriormente se indicará el nombre del archivo de proyecto, que tendrá la extensión VBP.

Para ejecutar el ejemplo bastará con seleccionar la opción *Iniciar* (Start) del menú *Ejecutar* (Run), hacer clic en el botón *Iniciar* (Start) de la Barra de herramientas o pulsar *[F5]*. Para finalizarlo, además de haciendo clic en el botón *Salir* del formulario, puede hacerse clic en el botón *Terminar* (End) de la Barra de herramientas, o simplemente cerrar el formulario.

2.5 ¿Por qué se decidió utilizar el Lenguaje de Programación Visual Basic para Windows?

Por ser este lenguaje sencillo, fácil de aprender y además, es de los más comunes y solicitados en las áreas de trabajo puesto que está enlazado con las herramientas comunes de oficina (Word, Excel, etcétera).

Capítulo 3

Geometría Analítica para Windows.

Este programa es una evolución de otros programas ya antes pensados para la creación de gráficas en D.O.S., donde se ejecutaba el programa para realizar dibujos de puntos, rectas, circunferencias, gráficas de funciones y ecuaciones cuadráticas y paramétricas

La diferencia de esos viejos programas con el actual GAWin, es que ahora precisamente el nombre es por trabajar en Windows, pues es el actual sistema operativo que se usa. Ya no necesita intentar aprender tantas instrucciones del mismo programa, solamente usando el ratón y ciertas palabras

También el programa actual, GAWin, hace las gráficas de ecuaciones cuadráticas, en el menú está la opción de ver a los triángulos y sus distintas propiedades (baricentro, circuncentro, etcétera).

Además, este programa puede utilizar el ratón para graficar rectas entre 2 puntos, así como triángulos y circunferencias con solo hacer un clic en el lugar en el que se desee estén las coordenadas de los vértices en el caso del triángulo o el lugar en el que se quiere, esté el círculo. En D O.S., al no utilizarse el ratón no se podía realizar esto.

En la sección 3.1, se describirá brevemente a cada una de las secciones del programa

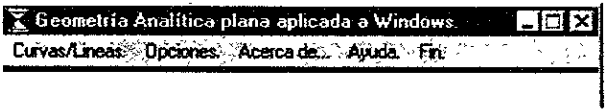
3.1 Descripción del programa.

El programa principalmente trata de hacer algo más cómodo para los alumnos de geometría analítica que requieren de gráficas de cónicas en un plano.

Está formado por cinco menús principales:

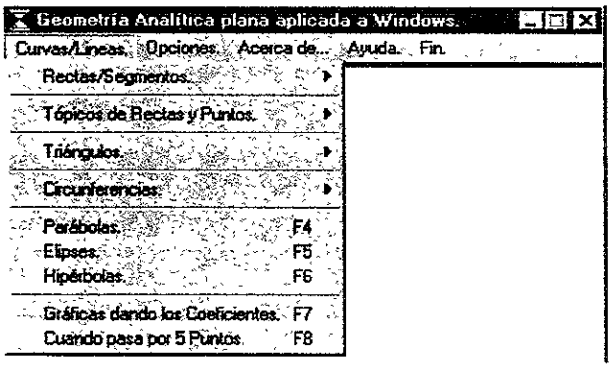
- el primero, es para las opciones de *Curvas* y *Líneas* a dibujar,
- el segundo es de las *Opciones* a utilizar, como lo son conocer fórmulas y gráficas de lo recientemente hecho,
- el tercer menú es un *Acerca de* quién lo realizó,
- el penúltimo es la *Ayuda* de temas a mostrar y con el
- quinto y último es para finalizar el programa.

La figura siguiente muestra la parte superior del programa, que expone las 5 opciones mencionadas:



Con respecto a la primera de Curvas/Líneas, se descompone en:

- *Rectas / Segmentos,*
- *Tópicos de Rectas y Puntos,*
- *Triángulos,*
- *Circunferencias,*
- *Parábolas,*
- *Elipses,*
- *Hipérbolas,*
- *Gráficas dando los Coeficientes* y
- La cónica que se genera *cuando pasa por 5 puntos.*

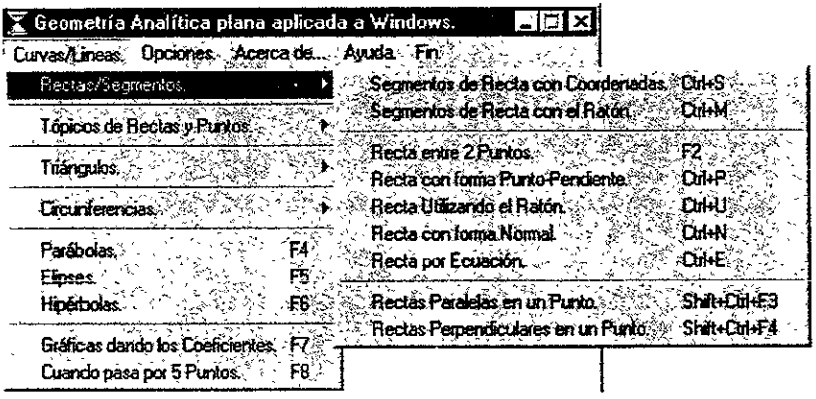


Los primeros 4 tienen submenús. El primero, Rectas/Segmentos, tiene las opciones de

- *Segmentos de Recta con Coordenadas,*
- *Segmentos de Recta con el Ratón,*
- *Recta entre 2 Puntos, Recta entre 2 Puntos,*
- *Recta con forma Punto-Pendiente,*
- *Recta utilizando el Ratón,*
- *Recta con forma Normal,*
- *Recta por Ecuación,*

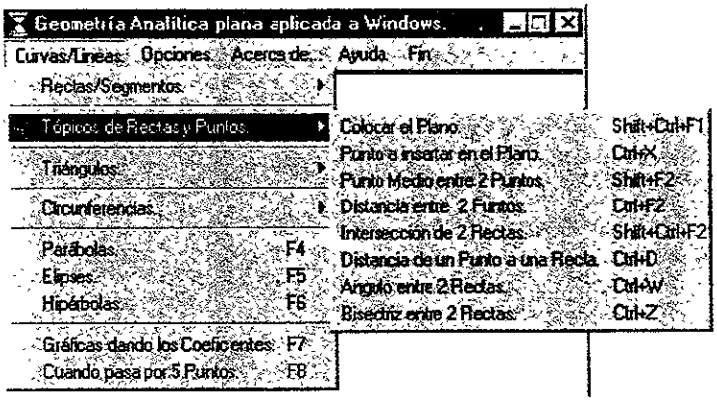
- *Rectas paralelas en un Punto* y
- *Perpendiculares en un Punto.*

La figura siguiente muestra el despliegue de la opción anterior correspondiente a las *Rectas* y los *Segmentos*.



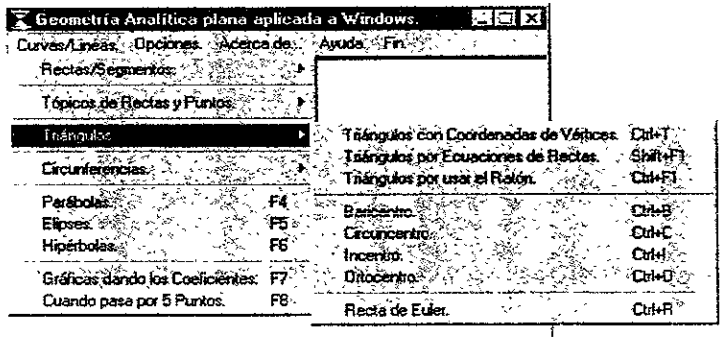
En el correspondiente a Tópicos de Rectas y Puntos, están las opciones:

- *Colocar el Plano,*
- *Punto a insertar en el Plano,*
- *Punto Medio entre 2 Puntos,*
- *Distancia entre 2 Puntos,*
- *Intersección de 2 Rectas,*
- *Distancia Punto-Recta,*
- *Angulo entre 2 Rectas,*
- *Bisectriz entre 2 Rectas.*



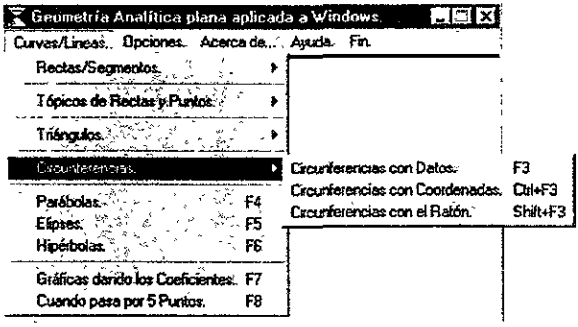
Para la tercer opción están unas formas de crear triángulos y ciertas propiedades, estas son:

Triángulos con Coordenadas,
Triángulos por ecuaciones de Rectas,
Triángulos por usar el Ratón,
Baricentro,
Circuncentro,
Incentro.
Ortocentro y
Recta de Euler



Para la cuarta opción del primer menú, tan solo son 3 opciones de cómo crear una circunferencia, las cuales son:

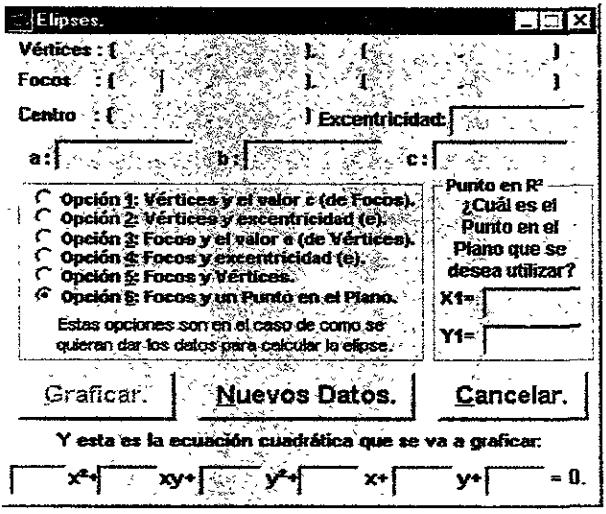
- *Circunferencias con Datos,*
- *Circunferencias con Coordenadas* y
- *Circunferencias con el Ratón.*



Las complementarias opciones del menú Curvas/Líneas son:

- *Parábolas,*
- *Elipses,*
- *Hipérbolas,*
- *Gráficas dando los Coeficientes y*
- *Cuando pasa por 5 Puntos.*

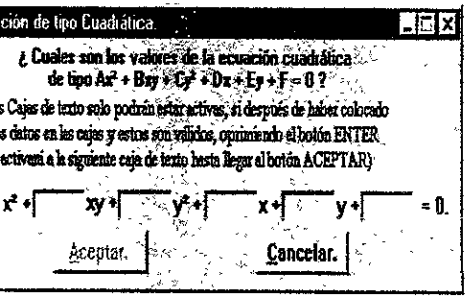
En los casos de las Parábolas, Elipses e Hipérbolas, cada una de ellas tiene opciones de con qué detalles se desea graficar a la cónica, por decir, con ciertos Vértices y Excentricidad o Focos y el valor de a o por su definición: conociendo los focos y un punto por el que pasa.



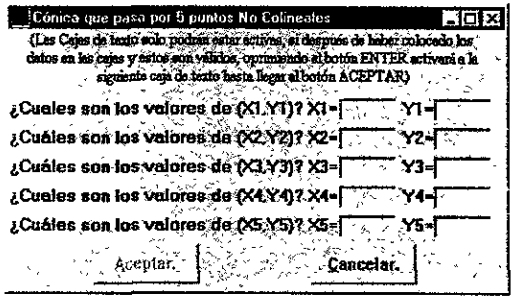
Esta figura es similar a la 6.19 mostrada en el archivo de ayuda correspondiente a las elipses, que solicita los valores de los focos y un punto a utilizar; lo mismo sucede para las hipérbolas y parábolas, solo que en estas últimas es con menos opciones.

En los otros 2 casos, se trata de detalles de ciertas cuadráticas, sin conocer sus valores, tan solo con conocer que pasa por 5 puntos, pues se sabe que ellos determinan una cónica. Estos resultados generan una ecuación cuadrática y esta ecuación cuadrática dibuja a la cónica y además da los detalles de sus elementos. Estas últimas 2 opciones fueron *Gráficas dando los Coeficientes* y *Cuando pasa por 5 puntos.*

Gráficas dando los Coeficientes

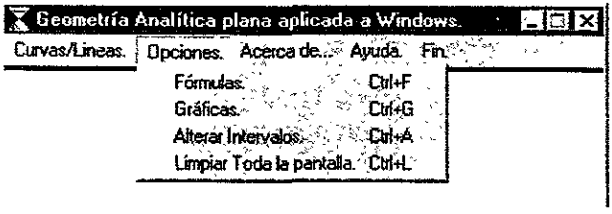


Cuando pasa por 5 puntos

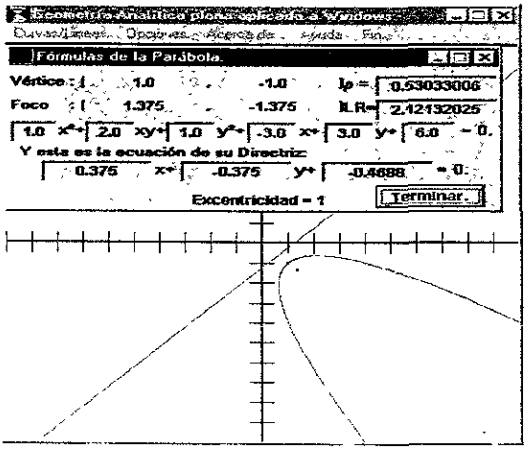


Con respecto al segundo *menú*, este tiene las *opciones* de:

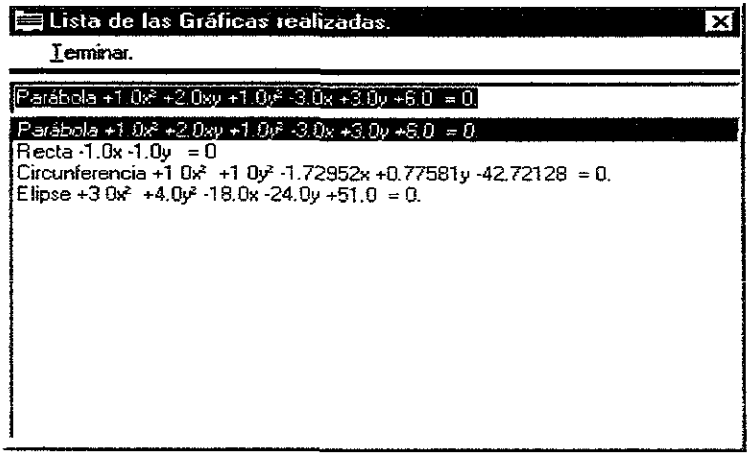
- *Fórmulas,*
- *Gráficas,*
- *Alterar Intervalos* y
- *Limpiar Toda la pantalla.*



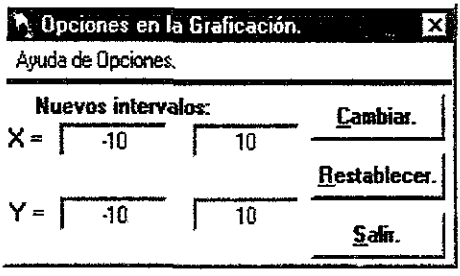
Con la opción *Fórmulas*, se muestran las fórmulas de alguna gráfica que se haya hecho de una forma reciente, si se desea volver a ver la ecuación de una cónica realizada, se deberá solicitar la figura de nuevo, para que la tome como gráfica reciente y vuelva a tener preparada la forma con sus ecuaciones. A continuación se muestra el ejemplo con una parábola.



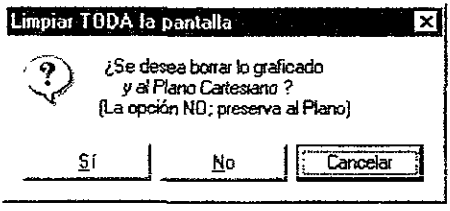
Las *gráficas* son las cónicas ya mostradas en la pantalla, cuyas fórmulas se pudieron haber visto alguna vez. Es una forma que muestra una lista de las cónicas que se hayan dibujado, aunque se haya deseado limpiar la pantalla. Usando el ejemplo anterior, lo que mostrará es lo siguiente:



El submenú de *Alterar Intervalos* es para cambiar el rango de que punto a que punto se puede ver la gráfica realizada; cuyo rango es en general de [-10,10] tanto en el eje ***X*** como en el eje ***Y***.

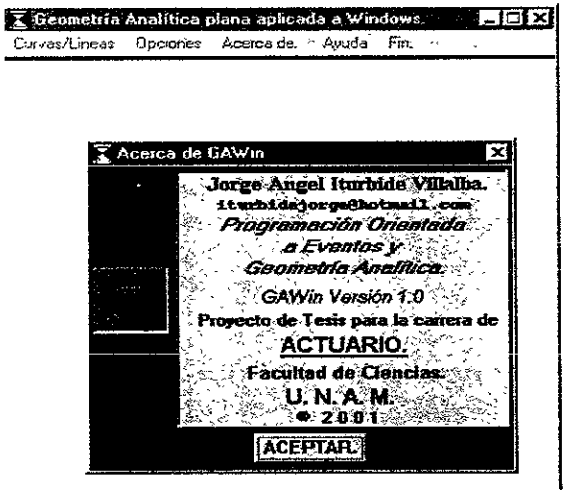


Y con respecto a la última opción, con el nombre se dice todo: es para borrar de la pantalla dibujos ya realizados, pudiéndose preservar el plano, o volverse a dibujar cuando se realice otra gráfica. El aviso siguiente se presenta en caso de que se desee *limpiar* la pantalla:

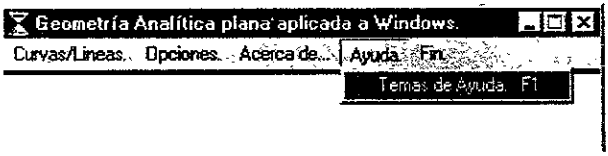


En el archivo de ayuda GAWin.HLP, en el capítulo 9 de ***Opciones y Complementos*** se hace más mención a lo referente a las opciones y todo el programa en sí.

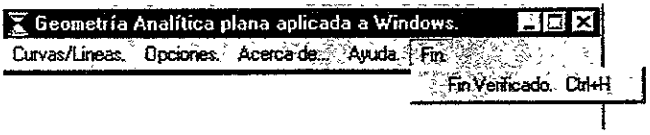
En el tercer menú, con pasar a esa opción, mostrará una forma diciendo el nombre de la tesis y de su autor.



El cuarto menú es para mostrar la ayuda del programa:



Con solo oprimir la tecla *F1* se despliega la ventana del archivo de ayuda (GAWin.HLP). Y la última opción es para salir del programa.



Después de oprimir las teclas *Ctrl + H*, se muestra un pequeño aviso para verificar si se está seguro de salir del programa.

Si se oprime el botón predeterminado, se concluye el programa, si se pasa al otro, se descarga el mensaje como si no se hubiera solicitado. Esta opción también se solicita con la tecla ESC.

Para más detalles de cómo funciona el programa, consultar el archivo GAWin.HLP.

Apéndice A

Palabras reservadas principales del lenguaje Visual Basic versión 4.

Palabras Reservadas: *Eventos.*

Dentro de los *eventos* dentro del lenguaje de esta versión 4.0:

Activate	AfterAddFile (Add-In)
AfterChangeFileName (Add-In)	AfterCloseFile (Add-In)
AfterColUpdate	AfterDelete
AfterInsert	AfterNewProject (Add-In)
AfterRemoveFile (Add-In)	AfterUpdate
AfterWriteFile (Add-In)	BeforeColUpdate
BeforeDelete	BeforeInsert
BeforeLoadFile (Add-In)	BeforeUpdate
Change	CheckIn (Add-In)
CheckOut (Add-In)	Click
ColResize	ConnectAddIn (Add-In)
DblClick	Deactivate
DisconnectAddIn (Add-In)	DoGetAddFileName (Add-In)
DoGetNewFileName (Add-In)	DoGetOpenProjectName (Add-In)
DragDrop	DragOver
DropDown	Error
Fetch (Add-In)	GotFocus
HeadClick	Initialize
KeyDown	KeyPress
KeyUp	LinkClose
LinkError	LinkExecute
LinkNotify	LinkOpen
Load	LostFocus
MouseDown	MouseMove
MouseUp	ObjectMove
Paint	PathChange

PatternChange	QueryUnload
Reposition	RequestChangeFileName (Add-In)
RequestWriteFile (Add-In)	Resize
RowColChange	RowLoaded
RowResize	Scroll
SelChange	Terminate
Timer	UnboundAddData
UnboundDeleteRow	UnboundReadData
UnboundWriteData	UnCheckOut (Add-In)
Unload	Updated
Validate	

Palabras Reservadas: *Funciones.*

Para las *funciones*, hay más palabras todavía, pues son mayores todas las funciones que puede hacer *un* objeto en base a *un* evento. Las distintas funciones son:

Abs	Array	Asc
AscB	Atn	CBool
CByte	CCur	CDate
CDbl	Choose	Chr
ChrB	CInt	CLng
Command	Cos	CreateDatabase
CreateObject	CSng	CStr
CurDir	CVar	CVErr
Date	DateAdd	DateDiff
DatePart	DateSerial	DateValue
Day	DDB	Dir
DoEvents	Environ	EOF
Error	Exp	FileAttr
FileDateTime	FileLen	Fix
Format	FreeFile	FV
GetAllSettings	GetAttr	GetObject
GetSetting	Hex	Hour
IIf	IMEStatus	Input
InStr	Int	IPmt
IRR	IsArray	IsDate
IsEmpty	IsError	IsMissing

IsNull	IsNumeric	IsObject
LBound	LCase	Left
Len	LoadPicture	LoadResData
LoadResPicture	LoadResString	Loc
LOF	Log	LTrim
Mid	Minute	MIRR
Month	MsgBox	Now
NPer	NPV	Oct
OpenDatabase	Partition	Pmt
PPmt	PV	QBColor
Rate	ReadProperty	RGB
Right	Rnd	RTrim
Second	Seek	Sgn
Shell	Sin	SLN
Space	Spc	Sqr
Str	StrComp	StrConv
String	Switch	SYD
Tab	Tan	Time
Timer	TimeSerial	TimeValue
Trim	TypeName	UBound
UCase	Val	VarType
Weekday	Year	

Palabras Reservadas: *Métodos*.

Ahora con respecto a los *métodos*, la larga lista dentro de la versión 4.0 es:

Add	Add (Add-In)
Add (DBGrid)	AddFile (Add-In)
AddFormTemplate (Add-In)	AddItem
AddMDIFormTemplate (Add-In)	AddMenu (Add-In)
AddMenuTemplate (Add-In)	AddNew
AddReference (Add-In)	AddToolboxProgID (Add-In)
AddToolboxTypelib (Add-In)	AddToolboxVBX (Add-In)
Append	AppendChunk
Arrange	CancelUpdate
CellText	CellValue
Circle	Clear

Clear (Clipboard, ComboBox, ListBox)
Clone
Close
Close (OLE Container)
Cls
ColContaining
CompactDatabase
ConnectEvents (Add-In)
Copy
CopyQueryDef
CreateDatabase
CreateDynaset
CreateEmbed
CreateField
CreateGroup
CreateLink
CreateProperty
CreateQueryDef
CreateRelation
CreateSnapshot
CreateTableDef
CreateUser
CreateWorkspace
Delete
Delete (OLE Container)
DeleteQueryDef
DisconnectEvents (Add-In)
DoVerb
Drag
Edit
EndDoc
Execute
ExecuteSQL
FetchVerbs
FieldSize
FillCache
FindFirst
FindLast
FindNext
FindPrevious
GetBookmark
GetChunk
GetData
GetFormat
GetRows
GetText
Hide
Idle
InsertFile (Add-In)
InsertObjDlg
Item (Add-In)
Item
KillDoc
Line
LinkExecute
LinkPoke
LinkRequest
LinkSend
ListFields
ListIndexes
ListParameters
ListTables
LoadProject (Add-In)
MakeReplica
Move
Move (Data Access)
MoveFirst
MoveLast
MoveNext
MovePrevious
NewPage
NewPassword
OpenDatabase
OpenQueryDef

OpenRecordset	OpenTable
PaintPicture	Paste
PasteSpecialDlg	Point
PopupMenu	Print
PrintForm	PSet
Raise	ReadFromFile
Rebind	Refill
Refresh	Refresh (Data Access)
RefreshLink	RegisterDatabase
Reload (Add-In)	Remove
Remove (Add-In)	Remove (Data Access)
RemoveComponent (Add-In)	RemoveItem
RepairDatabase	Requery
RowBookmark	RowContaining
RowTop	SaveAs (Add-In)
SaveToFile	SaveToOle1File
Scale	ScaleX
ScaleY	Scroll
Seek	SetData
SetFocus	SetText
Show	ShowColor
ShowFont	ShowHelp
ShowOpen	ShowPrinter
ShowSave	ShowWhatsThis
Synchronize	TextHeight
TextWidth	Update (Data Access)
Update (OLE Container)	UpdateControls
UpdateRecord	WhatsThisMode
WriteProperty (Add-In)	ZOrder

Palabras Reservadas: *Objetos* y *Colecciones.*

Para los *objetos y colecciones*, la lista también es grande, éstos objetos son los que generan los eventos antes descritos y que tienen las funciones y métodos antes descritos. En cierta forma, los Controles son Objetos, aunque los clasifiquen por separado.

App Object	Application Object (Add-In)
CheckBox Control	ClassModule Object

Clipboard Object
Column Object
ComboBox Control
CommonDialog Control
Components Collection (Add-In)
Containers Collection
ControlTemplate Object (Add-In)
Data Control
Databases Collection
DBEngine Object
DBList Control
DirListBox Control
Documents Collection
Dynaset Object
Error Object
Field Object
FileControl Object (Add-In)
Font Object
Forms Collection
Frame Control
Group Object
HScrollBar Control
Index Object
Label Control
ListBox Control
Menu Control
MenuLine Object (Add-In)
OptionButton Control
Parameters Collection
PictureBox Control
Printers Collection
Properties Collection (Add-In)
Property Object (Add-In)
QueryDef Object
Recordset Object
Relation Object
RowBuffer Object

Collection Object
Columns Collection
CommandButton Control
Component Object (Add-In)
Container Object
Controls Collection
ControlTemplates Collection (Add-In)
Database Object
DBCombo Control
DBGrid Control
Debug Object
Document Object
DriveListBox Control
Err Object
Errors Collection
Fields Collection
FileListBox Control
Form Object
FormTemplate Object (Add-In)
Grid Control
Groups Collection
Image Control
Indexes Collection
Line Control
MDIForm Object
MenuItems Collection (Add-In)
OLE Container Control
Parameter Object
Picture Object
Printer Object
ProjectTemplate Object (Add-In)
Properties Collection (Data Access)
Property Object (Data Access)
QueryDefs Collection
Recordsets Collection
Relations Collection
Screen Object

SelBookmarks Collection	SelectedComponents Collection (Add-In)
SelectedControlTemplates Collection (Add-In)	Shape Control
Snapshot Object	SubMenu Object (Add-In)
Table Object	TableDef Object
TableDefs Collection	TextBox Control
Timer Control	User Object
Users Collection	VScrollBar Control
Workspace Object	Workspaces Collection

Palabras Reservadas: *Propiedades.*

Para las *Propiedades* de estos objetos (Controles), la lista parece ser también demasiado larga:

AbsolutePosition	Action (CommonDialog)
Action (OLE Container)	ActiveControl
ActiveForm	ActiveProject (Add-In)
AddInMenu (Add-In)	Align
Alignment	AllowAddNew
AllowDelete	AllowRowSizing
AllowSizing	AllowUpdate
AllowZeroLength	AllPermissions
Appearance	AppIsRunning
Application (Add-In)	Archive
Attributes	AutoActivate
AutoRedraw	AutoShowChildren
AutoSize	AutoVerbMenu
BackColor	BackStyle
BOF	BOFAction
Bold	Bookmark Property (Data Access)
Bookmark Property (DBGrid)	Bookmarkable
BorderColor	BorderStyle
BorderWidth	BoundColumn
BoundText	CacheSize
CacheStart	Cancel
CancelError	Caption
CellSelected	Checked
Class	ClassName (Add-In)
Clip	ClipControls

Clustered
ColAlignment
ColIsVisible
Color
ColPos
ColumnCount
ColumnName
Columns (ListBox)
Comments
ConflictTable
Container (Add-In)
Container
ControlTemplates (Add-In)
Count (Data Access)
CurrentX
Data
DatabaseName
DataField
DataSource
DataUpdatable
Default
DefaultPassword
DefaultValue
Description (ClassModule)
Description
DeviceName
DisplayType
DividerStyle
DragMode
DrawStyle
Drive
Duplex
Enabled
EOFAction
EXEName
FileControl (Add-In)
FileDescription

Col
ColIndex
CollatingOrder
ColorMode
Cols
ColumnHeaders
Columns (DBGrid)
ColWidth
CompanyName
Connect
Container (Data Access)
ControlBox
Copies
Count (VB Collections)
CurrentY
Database
DataChanged
DataMode
DataText
DateCreated
DefaultExt
DefaultUser
DefColWidth
Description (Data Access)
DesignMasterID
DialogTitle
DistinctCount
DragIcon
DrawMode
DrawWidth
DriverName
EditMode
EOF
Exclusive
Fields
FileCount (Add-In)
FileName

FileNames (Add-In)
FileTitle
FillStyle
Filter
FirstRow
FixedCols
Flags (Color Dialog)
Flags (Font Dialog)
Font
FontCount
FontName
FontSize
FontTransparent
ForeColor
ForeignName
Format
FullName (Add-In)
GridLineWidth
hDC
HeadFont
HeadLines
HelpCommand
HelpContext (Data Access)
HelpContextID
HelpFile (Data Access)
HelpKey
HideSelection
hInstance
hPal
Icon
IgnoreNulls
Index (Control Array)
IndexedValue (Add-In)
Inherited (Data Access)
InitDir
IntegralHeight
IsDirty (Add-In)

FileNumber
FillColor
Filter (CommonDialog)
FilterIndex
FixedAlignment
FixedRows
Flags (File Dialog)
Flags (Print Dialog)
FontBold
FontItalic
Fonts
FontStrikethru
FontUnderline
Foreign
ForeignTable
FromPage
GridLines
Handle
HeadBackColor
HeadForeColor
Height
HelpContext (CommonDialog)
HelpContext
HelpFile (App, CommonDialog, MenuLine)
HelpFile
Hidden
HighLight
HostName
HWnd
IconState (Add-In)
Image
Index (Data Access)
Inherit
IniPath
Instancing
Interval
IsolateODBCTrans

Italic
KeepLocal
LargeChange
LastModified
LastUsedPath (Add-In)
Left
LegalCopyright
LinkItem
LinkTimeout
List
ListField
Locked
LoginTimeout
LpOleObject
MatchedWithList
Max (CommonDialog)
MaxButton
MaxLength
MenuItems (Add-In)
Min (Scroll Bar)
Minor
MouseIcon
MultiLine
Name
NegotiateMenus
NegotiateToolbars
NewIndex
Normal
Number
NumIndices (Add-In)
Object
ObjectAcceptFormatsCount
ObjectGetFormatsCount
ObjectVerbs
ODBCTimeout
OLERequestPendingMsgText
OLERequestPendingTimeout
ItemData
KeyPreview
LastDLLError
LastUpdated
Lbound
LeftCol
LegalTrademarks
LinkMode
LinkTopic
ListCount
ListIndex
LockEdits
LogMessages
Major
MatchEntry
Max (Scroll Bar)
MaxFileSize
MDIChild
Min (CommonDialog)
MinButton
MiscFlags
MousePointer
MultiSelect
Negotiate
NegotiatePosition
NewEnum (Add-In)
NoMatch
Number (Data Access)
NumberFormat
Object (OLE Container)
ObjectAcceptFormats
ObjectGetFormats
ObjectVerbFlags
ObjectVerbsCount
OLEDropAllowed
OLERequestPendingMsgTitle
OLEServerBusyMsgText

OLEServerBusyMsgTitle
OLEServerBusyRaiseError
OLEServerBusyTimeout
OLEType
OLETypeAllowed
Options
OrdinalPosition
Orientation
Owner
Page
PaperBin
PaperSize
Parent
Password
PasswordChar
PasteOK
Path
Pattern
PercentPosition
Permissions
Picture
PID
Port
PrevInstance
Primary
PrinterDefault
PrintQuality
ProductName
Properties (Add-In)
Public
QueryTimeout
ReadOnly (Data Access)
ReadOnly
ReadOnlyMode (Add-In)
RecordCount
RecordsAffected
RecordSelectors
Recordset
RecordsetType
RecordSource
Replicable
ReplicaID
Required
Restartable
ReturnsRecords
Revision
Row
RowCount (DBGrid)
RowDividerStyle
RowHeight
RowIsVisible
RowPos
Rows
RowSource
ScaleHeight
ScaleLeft
ScaleMode
ScaleTop
ScaleWidth
ScrollBars
SelBookmarks
SelCount
Selected
SelectedComponents (Add-In)
SelectedControlTemplates (Add-In)
SelectedItem
SelEndCol
SelEndRow
SelLength
SelStart
SelStartCol
SelStartRow
SelText
Shape

Shortcut
Size (Data Access)
SizeMode
Sort
Source (Data Access)
SourceDoc
SourceItem
SourceTableName
StartMode
StrikeThrough
System
Table
Tag
Text
Top
TopIndex
TrackDefault
TwipsPerPixelX
Type (Data Access)
UBound
Unique
UpdateOptions
UserName
ValidateOnSet
ValidationText
Value (Data Access)
Value
Version (Add-In)
Visible
VisibleCount
VisibleRows
WhatsThisButton (Windows 95)
WhatsThisHelpID (Windows 95)
WindowList
WordWrap
X2
Y2

ShowInTaskbar (Windows 95)
Size (Font)
SmallChange
Sorted
Source
SourceField
SourceTable
SQL
Stretch
Style
TabIndex
TabStop
TaskVisible
Title
ToPage
TopRow
Transactions
TwipsPerPixelY
Type (Picture)
Underline
Updatable
UseMnemonic
V1xNullBehavior
ValidationRule
Value (Add-In)
Value (RowBuffer)
Verb
Version
VisibleCols
VisibleItems
Weight
WhatsThisHelp (Windows 95)
Width
WindowState
X1
Y1
Zoom

Palabras Reservadas: *Declaraciones.*

De las últimas palabras reservadas a citar, son las *declaraciones*:

AppActivate	Beep
BeginTrans	Call
ChDir	ChDrive
Close	CommitTrans
CompactDatabase	Const
Date	Declare
DefBool (Deftypes)	DefByte (Deftypes)
DefCur (Deftypes)	DefDate (Deftypes)
DefDbl (Deftypes)	DefInt (Deftypes)
DefLng (Deftypes)	DefObj (Deftypes)
DefSng (Deftypes)	DefStr (Deftypes)
DefVar (Deftypes)	DeleteSetting
Dim	Do...Loop
Each (For Each...Next)	Else (If...Then...Else)
End	Erase
Error	Exit
FileCopy	For Each...Next
For...Next	FreeLocks
Function	Get
GoSub...Return	GoTo
If...Then...Else	Input #
Kill	Let
Line Input #	Load
Lock...Unlock	Loop (Do...Loop)
Lset	Mid
MkDir	Next (For Each...Next)
Name	On Error
On GoSub	On Goto
Oper	Option Base
Opt'on Compare	Option Explicit
Option Private	Print #
Private	Property Get
Property Let	Property Set

Public
Randomize
RegisterDatabase
RepairDatabase
Resume
RmDir
Rset
SaveSetting
Select Case
Set
SetDataAccessOption
Static
Sub
Time
Unload
While...Wend
With

Put
ReDim
Rem
Reset
Return (GoSub...Return)
Rollback
SavePicture
Seek
SendKeys
SetAttr
SetDefaultWorkspace
Stop
Then (If...Then...Else)
Type
Unlock (Lock...Unlock)
Width #
Write #

Conclusiones

En conclusión, se ha dicho desde el principio que el más cómodo de los estilos de programación (en lo particular para mí), es el que ya está orientado a eventos, pues al estar éste basado en los objetos creados por un ambiente, ya es más fácil crear programas fuertes, pero sin tener que realizar un mayor esfuerzo que el de estar generando las clases que deben de hacerse para poder construir al objeto. Aunque el lenguaje Visual Basic 4.0 en particular tiene ciertas características que hacen que parezca casi orientado a objetos, le faltan unas cuantas (como el polimorfismo) para ser orientado a objetos. De todos modos esto carece de importancia, pues lo realmente importante es la facilidad de uso, además de ser solicitado en el área laboral en estos tiempos. Tal vez versiones más recientes (como la 6.0) ya puedan tener el detalle de ser orientadas a objetos, pero en lo particular me agradó esta versión por su comodidad sin tener que estar aprendiendo y saltando de versión a versión.

Con la introducción que se hizo, se mostró lo sencillo que es aprender este lenguaje y ver cómo se pueden ir haciendo programas fáciles, hasta llegar a potentes administradores de bases de datos. Las mismas *palabras clave* de Visual Basic, desde sus eventos, métodos. etcétera; van mencionando que son de tipo *Data Access* o *Add-In*, que significa que son para acceso de datos o para agregar algo en un control. En la bibliografía se están mostrando libros fáciles de entender para autodidacticamente aprender tanto geometría analítica, como programación en este lenguaje tan solicitado.

Es por eso que este programa facilitará a muchos que no conocen de cómputo ni de geometría, el poder aprender el uso de un sencillo paquete de computadora; y a los que sí conocen por lo menos les permitirá comprobar sus resultados de manera visual, pues aunque existen paquetes tan potentes como el Scientific WorkPlace, que permite dibujar una cónica si se da su ecuación paramétrica, por ejemplo (a cos (t), b sin (t)) o se expresan pedazos de la curva como gráficas de funciones de una variable, como $f(x)= (a\ x^{2} + b\ y^{2})^{(1/2)}$, no tienen herramientas para dibujar una cónica si se dan algunos detalles de ella, como vértices y excentricidad, o focos y un semieje, etcétera. Así como tampoco permiten, dada una ecuación general de segundo grado en dos variables, determinar qué curva es (casos degenerados) y cuáles son sus elementos.

Aunque el programa describe lo que se está mencionando en el archivo de ayuda del programa, que es lo que está mostrado generalmente en cualquier libro de geometría analítica, en el estilo de la geometría analítica *plana*, no muestra lo relacionado a coordenadas polares ni ecuaciones paramétricas. Esto tal vez se realizaría para una futura versión.

Bibliografía

ALONSO, Ma. Dolores y RUMEAU, Silvia, Metodología de la Programación: Programación Estructurada, España, Paraninfo, 1992, 436 pp.

CEBALLOS SIERRA, Fco. Javier, Enciclopedia de Microsoft Visual Basic, España, RA-MA, 1994, 731 pp.

CORNELL, Gary, Manual de Visual Basic 3 para Windows, México, McGraw-Hill, 1994, 792 pp.

Como Programar En Visual Basic, Prensa Técnica, España, 1997, 256 pp.

DE OTEYZA, Elena, et al., Geometría Analítica, México, Prentice-Hall, 1995, 329 pp.

EISENHART, Luther Pfahler, Coordinate Geometry, E.U.A., GINN AND COMPANY, 1939,298 pp.

FULLER, Gordon y TARWATER, Dalton, Geometría Analítica, E.U.A., Addison-Wesley, 1988,382 pp.

HEYMAN, Mark Steven, La esencia de Visual Basic 4, México, Prentice-Hall, 1996, 457 pp.

KINDLE, Joseph H., Geometría Analítica (serie SHAUM), México, McGraw-Hill, 1988, 494 pp.

LEHMANN, Charles H., Geometría Analítica, México, Limusa, 1988, 494 pp.

Microsoft Corporation, Microsoft Visual Basic version 3.0 Language Reference, E.U.A., 1993,674 pp.

_____, Microsoft Visual Basic version 3.0 Programmer's Guide, E.U.A , 1993,713 pp.

NELSON, Ross, GUÍA COMPLETA DE VISUAL BASIC PARA WINDOWS, España, McGraw-Hill, 1994, 396 pp.

SMITH, Curtis y AMUNDSEN, Michael, Teach Yourself Database Programming with Visual Basic 4 in 21 days, E.U.A., SAMS Publishing, 1996, 855 pp.

STEEN, Frederick H. Y BALLOU, Donald H., Geometría Analítica, México, Publicaciones Cultural S.A., 1975, 353 pp.

SWOKOWSKI, Earl W., Algebra y Trigonometría con Geometría Analítica, México, Grupo Editorial Iberoamérica, 1989, 644 pp.

VOSS, Greg, Programación orientada a objetos: una introducción, México, McGraw-Hill, 1994, 597 pp.

WEXLER, Charles, Geometría Analítica: un enfoque vectorial, España, Montaner y Simón, 1968, 303 pp.

WOOTON, William, BECKENBACH. Edwin F. y FLEMING, Frank J., Geometría Analítica Moderna, México, Publicaciones Cultural S.A., 1985, 440 pp.