UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

DIVISIÓN DE INGENIERÍA MECÁNICA E INDUSTRIAL

"DESARROLLO DE UN SOFTWARE PARA EL ANÁLISIS DEL MODELO DE INVENTARIOS DE ARTÍCULOS MÚLTIPLES CON RESTRICCIÓN DE RECURSOS (PRESUPUESTO Y ESPACIO), E INTEGRACIÓN DE HERRAMIENTAS Y SOFTWARE PARA EL ANÁLISIS DE MODELOS GENERALES DE INVENTARIOS"

TESIS

PARA OBTENER EL TÍTULO DE:
INGENIERO INDUSTRIAL

PRESENTA:
FERNANDO MARIO AVILA CHAMORRO

DIRECTORA DE TESIS:
M. EN ING. SILVINA HERNÁNDEZ GARCÍA

MÉXICO, D.F. 2005

Dedicatorias y Agradecimientos

A Dios

Aunque no comparto un concepto clásico acerca de Dios siempre he tenido la sensación de que alguien más haya de mi familia me guía, alienta y cuida.

A papá y mamá

Sergio Avila Ramírez y *Yolanda Chamorro González*, por su cariño y apoyo incondicional. Son mi mayor bendición. Todo lo bueno que soy y pueda llegar a ser se los debo, lo malo, es mi entera responsabilidad.

¡Algún día.....!

A Miguel

Por haber compartido la experiencia de crecer juntos. ¡Ojalá continué!

A la Monse

Ha sido asombroso verte crecer. Eres un encanto.

A la memoria de mis abuelitos

Feliciano Avila y *Felipe Chamorro*, con cariño y añoranza. Se que les hubiese gustado compartir este momento conmigo.

A mis abuelitas

Gabriela Ramírez y *Ofelia González*, quienes son ejemplo de integridad, fortaleza y dedicación. Son excepcionales.

A mis amigos de la Facultad

Raúl, Legaspi, Jorge, Víctor, Charly, Luis, Eric, Ruth, Jenaro, Alejandro, Lety, Héctor, Adriana, Eynar, Paco, Areli, Xitlali, Armando, Talina, Noemí y *Margarita*. Gracias por su afecto, compañerismo y las innumerables experiencias compartidas.

A mi directora de tesis

M. en Ing. Silvina Hernández García, por todo el apoyo y paciencia brindados en la realización de este proyecto. Es usted excelente profesional y mejor ser humano.

A mis sinodales

M. A. José Gonzalo Guerrero Zepeda, M. I. Isabel Patricia Aguilar Juárez, M. I. Angel Leonardo Bañuelos Saucedo y *M. I. Andrés Mota Solórzano*, por su asesoría y recomendaciones. Gracias por el tiempo que dedicaron a la revisión de este trabajo.

A la Universidad Nacional Autónoma de México y su Facultad de Ingeniería

Gracias por permitirme ser universitario. En sus aulas recibí lecciones académicas y de vida invaluables, que me han permitido crecer y comprometerme con mi familia y la sociedad mexicana en su conjunto.

Índice

CAPÍTULO 6 **82**
Desarrollo del programa

Introducción

La presente obra fue escrita para obtener el título de Ingeniero Industrial tal como establece la legislación universitaria, por lo que el tema de tesis de la misma debía ser congruente con los conocimientos y habilidades adquiridas durante la carrera, es por esto que se decidió desarrollar un tema dentro del área de Análisis de Inventarios, la cual es vital para la Ingeniería Industrial.

Antes de presentar un panorama general de lo que el lector podrá encontrar en esta tesis, es conveniente definir lo que es un inventario:

> *"Inventario es una cantidad de bienes bajo el control de una empresa, guardados durante algún tiempo para satisfacer una demanda futura."*[1]

Ahora bien, ¿Qué es un inventario de artículos múltiples con restricción de recursos?, la siguiente definición puede responder esta pregunta:

> *Es un inventario en el cual se resguardan 2 o más clases de artículos, y se comparten recursos comunes limitados, como son presupuesto, capacidad de almacenaje y otros.*

En este escrito se presenta el desarrollo de un software para la resolución del modelo de inventarios de artículos múltiples con restricción de recursos (presupuesto y espacio), es decir, el lector podrá encontrar una descripción completa de cómo se creó el programa. El cual optimiza los recursos con los que se cuenta para administrar este tipo de inventarios, en otras palabras, *minimiza los costos cumpliendo con los requerimientos*, basándose en la teoría general de análisis de inventarios, teorías matemáticas de optimación, métodos numéricos y programación computacional.

También se incluyen muchas otras herramientas para el Análisis de Inventarios, todo esto con la intención de que el software le sea más útil al usuario final.

El inventario es uno de los activos más caros de muchas compañías, puede llegar a representar tanto como un 40% del capital total invertido. Los administradores de operaciones han reconocido desde hace mucho tiempo que el buen control del inventario es crucial en la organización. Por un lado, una empresa puede intentar la reducción de los costos mediante la reducción de los niveles de inventario. Por otro lado, los clientes se sienten insatisfechos cuando ocurren faltas frecuentes de inventario (llamado inventario agotado). Entonces, las compañías deben intentar un equilibrio entre la inversión en inventario y los niveles de servicio al cliente.

La teoría de Análisis de Inventarios se enfoca en la minimización del costo que es una importante función que se obtiene como resultado del detallado equilibrio entre la inversión

[1] Tomado de Daniel Sipper y Robert L. Bulfin Jr., *Planeación y Control de la Producción*, México, ed. McGraw-Hill, 2005, p. 219.

en inventario y los niveles de servicio al cliente. Es conveniente mencionar que la minimización del Costo de Inventario involucra más factores que los de la inversión en él y los niveles de servicio al cliente, por ejemplo, se puede mencionar el Costo de Ordenar, Costo de Mantener el Inventario, Costo de Faltante etc. Posteriormente se analizará cada uno de estos conceptos en profundidad.

Se ha indicado la importancia que tiene el Control de Inventarios, y por lo tanto también el análisis de estos para una compañía. Resultaría sumamente ineficiente analizar modelos de inventarios manualmente para casi todas las compañías existentes; de aquí que el uso de la computadora y softwares para estos fines se encuentre inmensamente difundido.

Existen gran variedad de programas para el Análisis de Inventarios, como son: *TORA*, *The Management Scientist*, *WinQSB*, etc. Pero ninguno de estos, al menos en versión estudiantil o freeware contienen un apartado o interfaz para resolver el modelo de inventarios de artículos múltiples con restricción de recursos. Observando este vacío y necesidad para la cátedra de nuestra facultad (especialmente la de los Ingenieros Industriales) y la factibilidad de crear un software, es como se decide desarrollar un programa de ordenador capaz de resolver el problema, por lo tanto se plantea el siguiente objetivo para la tesis.

> **Desarrollar un software capaz de analizar el problema del modelo de inventarios de artículos múltiples con restricción de recursos (Presupuesto y Espacio), acorde a las necesidades de la cátedra en la carrera de Ingeniería Industrial.**

Con la intención de que el programa sea lo más integral y útil posible para el Análisis de Inventarios, se establece el siguiente objetivo secundario.

> **Integrar software y herramientas para el análisis de modelos generales de inventarios al programa principal.**

Para alcanzar estos objetivos se decide utilizar la siguiente metodología, que es una variación del Método Científico (modelo general).

1. Delimitación del alcance de la solución.
2. Búsqueda de la información disponible para el análisis del modelo de inventarios de artículos múltiples con restricción de recursos.
3. Búsqueda de las posibles técnicas para el análisis matemático del modelo.
4. Elección de la técnica de análisis matemático más adecuada.
5. Análisis matemático.
6. Búsqueda de las alternativas de desarrollo computacional para los algoritmos de solución obtenidos.
7. Elección de la alternativa de desarrollo computacional más adecuada.
8. Programación del algoritmo de solución obtenido (creación del software).
9. Integración de software y herramientas para el análisis de modelos generales de inventarios.
10. Comprobación del software.

11. Conclusiones.

En el primer capítulo de este documento se describe la teoría clásica de análisis de inventarios y se contrasta con las teorías emergentes.

En el segundo capítulo se presenta la teoría existente con respecto al modelo de inventarios de artículos múltiples con restricción de recursos. Y se continúa por cuenta propia con el análisis y desarrollo matemático del modelo, con el objetivo de que la programación de los algoritmos de solución matemáticos del modelo sea viable.

En el tercero se exponen conceptos básicos de programación, desarrollo y clasificación de los lenguajes de programación y una completa semblanza de los lenguajes de programación más populares haciendo énfasis en *Visual Basic*.

El cuarto contiene un poco de historia acerca de los sistemas operativos (para computadoras) y su clasificación, así como una semblanza de los sistemas operativos más populares para PC.

En el quinto se describen herramientas para el análisis de inventarios divididas en dos grupos: software especialmente diseñado para el análisis de inventarios y programas de análisis matemático.

En el sexto se desarrolla el software después de justificar el sistema operativo bajo el que va a correr y el lenguaje de programación con el que se desarrollará.

En el séptimo se presenta un análisis de caso que permite comprobar el funcionamiento del software.

Y por último, las conclusiones contienen el alcance del software, su síntesis, especificaciones y el procedimiento de instalación. Mientras que los anexos el código fuente completo del software.

CAPÍTULO 1

Teoría de análisis de inventarios

La teoría de análisis de inventarios es una de las herramientas de las cuales se vale el Control de Inventarios para realizar su labor, pero, ¿Qué es el Control de Inventarios?

> *Es el conjunto de actividades, técnicas y herramientas utilizadas para mantener la cantidad de artículos (materiales, materias primas, producto en proceso y producto terminado) en el nivel deseado, tal que ni el costo ni la probabilidad de faltante sean de una magnitud significativa.*

Ahora bien, ya que se cuenta con una idea clara de lo que es el Control de Inventarios, habrá que definir lo que es la teoría de análisis de inventarios.

> *La teoría de análisis de inventarios es el conjunto de técnicas, modelos y herramientas matemáticas desarrolladas a partir de principios del siglo XX, que tienen como objetivo la minimización del costo de inventario.*

1 INVENTARIOS: DEFINICIÓN Y JUSTIFICACIÓN

Es recomendable dar un nuevo vistazo a la definición de inventario presentada en la introducción.

> *"Inventario es una cantidad de bienes bajo el control de una empresa, guardados durante algún tiempo para satisfacer una demanda futura."*[2]

Esta definición ayuda a inferir la característica fundamental de un inventario, el inventario es un "amortiguador" entre el abastecimiento y la demanda, ya que casi siempre existen diferencias entre los tiempos y tasas de estos 2 procesos.

Además de que los inventarios juegan el papel de amortiguadores entre el abastecimiento y la demanda, existen muchas otras razones que justifican su existencia, entre las cuales se pueden mencionar:

- Mejorar el nivel de servicio al cliente. Una parte fundamental dentro de cualquier empresa es "el cliente", ya que el es quien la mantiene viva y le da su razón de ser, es por esto que con el inventario se busca satisfacer la demanda y así mantener satisfecho al cliente.
- Descuentos por cantidad. Es una práctica común hoy en día aplicar descuentos de acuerdo a la cantidad de insumos adquiridos, esta beneficia tanto al vendedor como

[2] Tomado de Daniel Sipper y Robert L. Bulfin Jr., *Planeación y Control de la Producción*, México, ed. McGraw-Hill, 2005, p. 219.

al comprador, el vendedor recupera el abatimiento del costo del producto con el volumen vendido, y el comprador adquiere una ventaja competitiva al adquirir producto a un precio menor, aún cuando tenga que elevar sus niveles de inventario.

- Cantidades limitadas. Algunos sectores productivos requieren de insumos que por su naturaleza, ambiente macro y micro económico, condiciones sociales, etc. Son escasos, de aquí la necesidad de contar con inventarios capaces de satisfacer la demanda aún cuando exista escasez en los insumos.
- Proveedores no confiables. No obstante que se cuente con proveedores no confiables, se debe cumplir con el nivel de servicio que el consumidor espera si es que se quiere sobrevivir en un medio tan competido como el de hoy, y un medio para lograrlo es el inventario.
- Incertidumbre. Aunque se cuente con proveedores confiables (abastecimiento) y demandas uniformes y constantes siempre existe un cierto grado de incertidumbre, que puede ser evadido manteniendo un nivel de inventarios superior a la demanda pronosticada o bien pueda prever posibles contingencias en los reabastecimientos.
- Suavizamiento de la operación. Se usa cuando la demanda varía en el tiempo, lo cual es muy común en productos de temporada. En temporada baja se acumula inventario el cual será utilizado para satisfacer la demanda en temporada alta; esto permite que se mantenga una tasa de producción relativamente constante, lo cual es deseable en la manufactura.
- Economías de escala. Existen ciertos costos fijos como son los costos de preparación, de ordenar, etc. Que se pueden abatir a medida que se incrementa la cantidad de artículos comprados o producidos, es decir, el costo unitario promedio disminuye al ser prorrateado en un mayor número de unidades.
- Especulación. La fluctuación de los precios en el mercado puede ser la razón de mantener un inventario, ya que este puede constituirse en una ventaja competitiva que aumente las ganancias y por lo tanto la rentabilidad de la organización.

2 TIPOS DE INVENTARIOS

Existen gran variedad de clasificaciones para los inventarios, pero las principales son: de acuerdo a la certidumbre con la que se conoce la demanda, interdependencia entre las demandas de los artículos y según el valor agregado durante el proceso de manufactura.

2.1 Clasificación de acuerdo a la certidumbre con la que se conoce la demanda

Los inventarios se clasifican de acuerdo a la certidumbre con la que se conoce la demanda en:

- Inventario con demanda determinística. Aquí se conoce perfectamente la demanda futura del articulo en inventario.
- Inventario con demanda estocástica. En este tipo no se conoce con certeza la demanda futura del artículo, es decir, la demanda es aleatoria.

2.2 Clasificación de acuerdo a la interdependencia entre las demandas de los artículos

Se clasifican en:

- Inventarios con demanda dependiente. Se llaman así porqué la demanda de un artículo depende de la demanda de otro, esto es muy común en la manufactura. Imagine el caso de un televisor, la demanda de cada uno de sus componentes (transistores por ejemplo) depende de la demanda de televisores, pues cada televisor lleva a su vez varios transistores.
- Inventarios con demanda independiente. Aquí la demanda de un artículo no depende de la demanda de otro y está afectada principalmente por las condiciones del mercado, este tipo de inventario es común para los productos terminados.

2.3 Clasificación según el valor agregado durante la manufactura

En los sistemas de producción los inventarios se clasifican de la siguiente manera:

- Inventarios de materia prima. Incluye todos los materiales requeridos para los procesos de manufactura y ensamble. Normalmente son:
    1. Material que requiere de más procesamiento (madera, harina, barras de acero)
    2. Componentes que forman parte de un producto tal y como se adquirieron (chips, tuercas)
    3. Artículos de consumo (soldadura, pegamento, gasolina)
- Inventarios de producto en proceso. Es inventario que espera en el proceso de producción para ser procesado o ensamblado y puede incluir productos semiterminados o subensambles.
- Inventarios de productos terminados. Son las salidas de los procesos de producción (cualquier mercancía como llantas, refrescos, televisores. Los productos terminados de una organización pueden ser la materia prima de otra; por ejemplo, las llantas para los automóviles.

3 COSTOS DE INVENTARIO

Resulta obvio que un inventario incurra en costos, como son: el costo de compra, el costo de ordenar (de preparación), el costo de almacenaje y el costo por faltantes. Cada uno se explica a continuación.

El **costo de compra** es la cantidad total invertida en la compra de la mercancía, o el valor contable del producto cuando se trata de material en curso o productos terminados. Si c es el costo unitario de compra y Q es el número de unidades compradas (tamaño del lote). Entonces el costo total de compra es cQ.

Si se fabrica una unidad, c incluye tanto el costo del material como el costo variable para producirla. Por lo tanto, el costo total de manufactura para un lote de producción es cQ.

El **costo de ordenar** incluye todos los costos en que se incurre cuando se lanza una orden de compra. Los costos que se agrupan bajo esta rúbrica deben ser independientes de la cantidad que se compra y exclusivamente relacionados con el hecho de lanzar la orden. Sus componentes serían los siguientes:

- Costos implícitos del pedido: Costo de preparación de las máquinas cuando el pedido lo lanza producción.
- Costo de conseguir "LUGAR" en el almacén de recepción (movilización de mercancías o transporte a otras localizaciones, por ejemplo)
- Costos de transporte exclusivamente vinculados al pedido (la factura de un "courier" en el caso de una reposición urgente, por ejemplo)
- Costos de supervisión y seguimiento de la necesidad de lanzar un pedido, etc.
- Costos Administrativos vinculados al circuito del pedido.
- Costos de recepción e inspección.

Como se ve, este costo es independiente de la cantidad de artículos comprados, y por lo tanto es un costo fijo A. Para la fabricación el costo fijo de un lote es principalmente el costo de preparación, y también se denota con A.

De aquí que el costo total de comprar o producir un lote sea:

$$A + cQ$$

El inventario compromete capital, usa espacio y requiere mantenimiento. El costo de todo lo anterior es el **costo de almacenaje** o de mantener el inventario y puede incluir los siguientes rubros:

- Costos Financieros de las existencias
- Gastos del Almacén
- Seguros
- Costo de oportunidad
- Deterioros, perdidas y degradación de mercancía.

El costo de almacenar comienza con la inversión en el inventario, ya que este dinero no puede obtener rendimientos en otra parte (por ejemplo en una cuenta bancaria de ahorros) por estar comprometido en el inventario. Este costo es un costo de oportunidad que regularmente se expresa como un porcentaje de la inversión.

Muchas empresas tienen mejores oportunidades que las cuentas de ahorros y gran parte de ellas poseen una tasa mínima de retorno, que usan para evaluar inversiones, normalmente llamada costo de capital.

Otros costos en los que se incurre en el almacenaje, también se expresan como un porcentaje de la inversión en el, y se suman al costo de oportunidad (o de capital) para formar el costo total de mantener el inventario.

i = Costo total de mantener el inventario (en porcentaje)

Éste es el costo de mantener $1 de inventario durante una unidad de tiempo. Debido a que el inventario se mide normalmente en unidades y no en pesos, se tiene:

$$h = ic$$

donde h es el costo de mantener **una unidad** en inventario durante **una unidad** de tiempo. Normalmente i toma valores anuales de 25 a 40%, pero puede llegar hasta 60%.

El **costo por faltante** es la ganancia perdida y la pérdida de buena voluntad cuando no se tiene un producto que está siendo demandado. Si la demanda se surte atrasada existe un costo adicional al expedirla, costos de registro en libros y la reputación de un mal servicio. Cuando un material para producción se surte atrasado se puede parar la manufactura, se tardará en volver a arrancarla y tal vez haya una entrega tardía de producto al cliente. Es difícil estimar el costo por faltantes y puede ser una estimación subjetiva.

Hay 2 tipos de costos por faltantes. El primero resulta de que falte una unidad; y el otro toma en cuenta el tiempo que falta, se tiene:

π = Costo de faltante por unidad
$\hat{\pi}$ = Costo de faltante por unidad que falta por unidad de tiempo

$\hat{\pi}$ es para los faltantes lo que h es para el inventario.

Habiendo expuesto los costos de inventario, es posible hacer una primera aproximación al **costo total del inventario**, que es:

Costo de compra + C. de ordenar + C. de almacenaje + C. por faltante = $cQ + A + \pi + \hat{\pi}$

4 POLÍTICAS DE INVENTARIO

Existen un par de políticas de control de inventarios, llamadas de revisión periódica y de revisión continua.

- Política de revisión periódica. El nivel de inventario se verifica en intervalos de tiempo, puede ser una semana, un mes, o cualquier periodo T, llamado periodo de revisión, y se coloca una orden si el nivel de inventario es menor que cierto nivel predeterminado R, llamado punto de reorden. El tamaño de la orden Q es la cantidad requerida para aumentar el inventario al nivel S. Q varía de un periodo a otro. La Figura 1-1 presenta esta política suponiendo que la demanda es de una unidad a la vez y que las órdenes se entregan instantáneamente.
- Política de revisión continua. Aquí el nivel de inventario se monitorea y controla continuamente. Cuando el nivel llega al punto de reorden R, se ordena una cantidad fija Q. Esta es una política continua (Q, R), o política de cantidad fija de reorden.

Esta política se muestra en la Figura 1-2, y también se supone entrega instantánea y demanda de una unidad a la vez.
Antes de la era de las computadoras, era sumamente compleja y costosa la implementación de una política de revisión continua, pero ya que hoy en día hay computadoras disponibles casi en cualquier sitio, esta política se ha popularizado fuertemente.

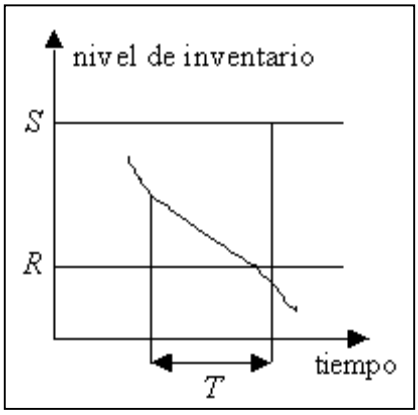


Figura 1-1
Política de revisión periódica

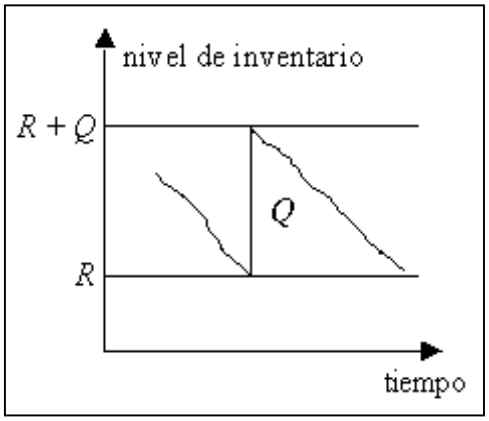


Figura 1-2
Política de revisión continua

5 MODELOS DE INVENTARIOS

Los modelos de inventarios son modelos matemáticos que tienen por objetivo minimizar el costo total de inventarios. Existe una gran variedad de modelos de inventarios, tantos como condiciones particulares tenga el tipo de inventario que analizan. El más importante y fundamental de ellos es el modelo EOQ (*Economic Order Quantity*; fórmula de Wilson) desarrollado por Harris en 1915.

5.1 Cantidad económica a ordenar (EOQ)

El modelo EOQ es el modelo fundamental en la teoría de análisis de inventarios, ya que sirve como base para la construcción de otros más elaborados, y todavía es uno de los modelos más utilizados en la industria.

Antes de desarrollar el modelo EOQ, se deben suponer ciertas condiciones:

- No se permiten faltantes.
- Hay una sola clase de artículo en el sistema.
- La demanda es uniforme y determinística (constante) de D unidades por unidad de tiempo.
- Toda la cantidad ordenada o pedida llega al mismo tiempo (tasa de reabastecimiento infinita).
- No hay tiempo de entrega.

Los parámetros a utilizar son los siguientes:

- Q = número de unidades a ordenar, debe ser un número entero positivo
- c = costo unitario ($ / unidad)
- i = costo total anual de mantener el inventario (% por año)
- $h = ic$ = costo total anual de mantener el inventario ($ por unidad por año)
- A = costo de ordenar ($ / orden)
- D = demanda por unidad de tiempo (unidades por año)
- T = longitud del ciclo, es el tiempo que transcurre entre colocación (recepción) de ordenes sucesivas de abastecimiento (años)
- $K(Q)$ = costo total anual promedio como una función del tamaño de lote Q.
- I_t = inventario disponible en el tiempo (inventario real en almacen)

Antes de proseguir con el desarrollo del modelo EOQ, resultará útil mostrar la geometría del inventario en la Figura 1-3 que es una descripción gráfica de I_t.

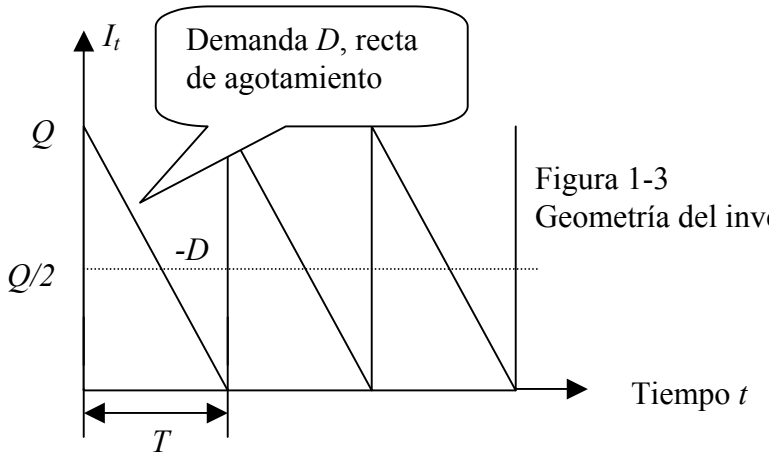


Figura 1-3
Geometría del inventario EOQ

En el tiempo cero el nivel de inventario es Q, a medida que pasa el tiempo el inventario se va agotando a una tasa D hasta llegar a agotarse cuando termina el periodo T, en ese momento se coloca una orden que será abastecida inmediata y completamente, es por eso que al comenzar el siguiente periodo el nivel de inventario se encuentra de nuevo instantáneamente en Q.

De la geometría del inventario es posible observar que:

$$T = \frac{Q}{D}$$

Y sea $\bar{I}$ el inventario promedio y A_i el área bajo la curva de inventario, se obtiene:

$$\bar{I} = \frac{A_i}{T} = \frac{1}{T}\frac{QT}{2} = \frac{Q}{2}$$

Ahora bien, se tienen los siguientes costos: costo de compra, costo de ordenar y costo de mantener inventario. No se menciona el costo por faltante ya que una de las condiciones del modelo EOQ es que no existen faltantes, y por lo tanto, tampoco su costo. Para cada ciclo, los costos son:

$$\text{Costo de compra} = cQ$$

$$\text{Costo de ordenar (o de preparación)} = A$$

$$\text{Costo promedio de mantener el inventario} = icT\frac{Q}{2} = hT\frac{Q}{2}$$

Es así, como el costo promedio por ciclo es:

$$cQ + A + hT\frac{Q}{2}$$

Para obtener el costo promedio anual *K(Q)*, sólo hay que multiplicar la ecuación anterior por el número de ciclos (*1/T*), y recordando que *1/T = D/Q*, se tiene:

$$K(Q) = \frac{cQ}{T} + \frac{A}{T} + h\frac{Q}{2}$$

$$K(Q) = cD + \frac{AD}{Q} + h\frac{Q}{2}$$

Si se quiere encontrar el valor de *Q* que minimiza *K(Q)* es posible aplicar el criterio de la primera y segunda derivada del cálculo diferencial para encontrar puntos de inflexión.

$$K'(Q) = \frac{dK(Q)}{dQ} = -\frac{AD}{Q^2} + \frac{h}{2} = 0$$

$$K''(Q) = \frac{d\left(\frac{dK(Q)}{dQ}\right)}{dQ} = \frac{2AD}{Q^3}$$

Como la Demanda *D*, el costo de ordenar *A*, y el número de unidades a ordenar *Q* deben ser positivos, $K''(Q)$ es positiva y *K(Q)* es una función cóncava hacía arriba que alcanza su mínimo en el punto donde la primera derivada de *K(Q)* es cero, resolviendo $K'(Q)$ se tiene:

$$Q^* = \sqrt{\frac{2AD}{h}}$$

Q^* es la cantidad económica a ordenar o lote económico o EOQ.

5.2 Otros modelos de inventarios

Antes de presentar los modelos más comunes desarrollados a lo largo de los años, resulta conveniente dividirlos de acuerdo a la certidumbre con la que se conoce la demanda.

- Modelos estáticos de tamaño de lote. Se usan cuando se tiene una demanda uniforme (constante), durante el tiempo de planeación.
- Modelos dinámicos de tamaño de lote. Son modelos que se usan cuando cambia la demanda durante el tiempo de planeación, es decir, se conoce la demanda con certidumbre aunque esta cambia a lo largo del tiempo.
- Modelos con demanda probabilística. Su nombre lo indica, la demanda no se conoce con certidumbre, es estocástica.

En la Figura 1-4 se presenta una organización general de los modelos de inventarios.

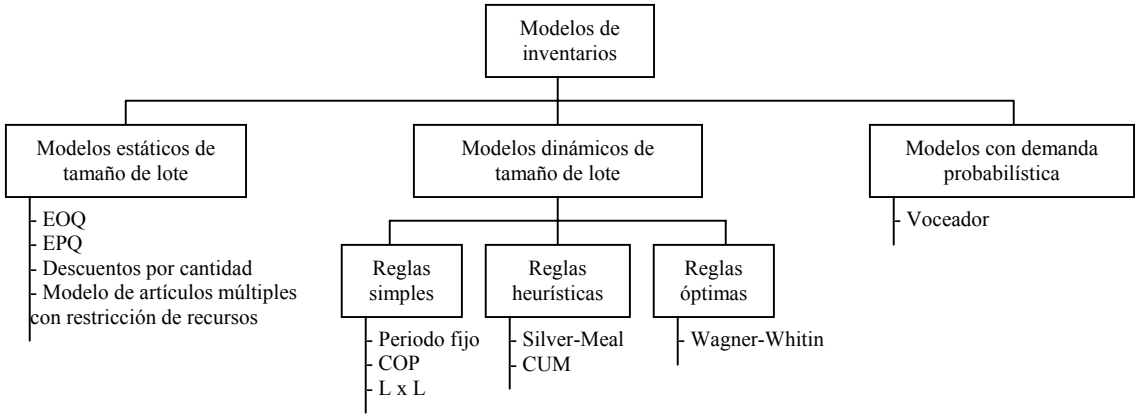


Figura 1-4
Organización general de los modelos de inventarios

5.2.1 Modelos estáticos de tamaño de lote

En realidad, todos los modelos estáticos de tamaño de lote son una extensión del modelo básico EOQ, con la única diferencia de que en cada modelo se van eliminando condiciones necesarias para el modelo EOQ, como las siguientes: no se permiten faltantes, no hay tiempo de entrega, la demanda es determinística y constante, hay una sola clase de artículo en el sistema y se tiene una tasa de reabastecimiento infinita.

- EOQ (Cantidad económica a ordenar). Este modelo ya fue descrito en un apartado anterior, y como se dijo, es la base para el desarrollo de modelos más elaborados.
- EPQ (Cantidad económica a producir). Es una extensión del EOQ, ya que se permite una tasa de reabastecimiento finita, es decir, no toda la cantidad ordenada llega al mismo tiempo, además de que se admiten faltantes. Esta situación es normal para artículos fabricados, en donde se entrega el lote a través del tiempo de acuerdo a la tasa de producción.

- Descuentos por cantidad. En el modelo EOQ se supone que el costo unitario es constante, independiente de la cantidad de unidades que se compre. En realidad, el proveedor puede ofrecer un plan de compras en el que varíe el precio unitario de acuerdo a la cantidad de unidades compradas, es decir, si la cantidad comprada es mayor que una cantidad específica, el costo por unidad se reduce.
  Existen dos tipos de planes de descuento. El **descuento en todas las unidades** aplica el descuento en el precio de todas ellas, desde la primera, si la cantidad excede el corte de descuento. El otro tipo aplica el descuento sólo al precio de las unidades que exceden la cantidad de corte, que es el plan de **descuento incremental**.
- Modelo de artículos múltiples con restricción de recursos. En este modelo se relaja la condición de que sólo puede haber una clase de artículo en el sistema, teniendo en cuenta que los artículos comparten recursos comunes, por ejemplo: presupuesto y espacio.
  En el siguiente capítulo se describe y analiza de manera profunda este modelo, ya que es una de las piedras angulares de la presente tesis.

5.2.2 Modelos dinámicos de tamaño de lote

Estos modelos surgen cuando la demanda es irregular, es decir, cuando no es uniforme durante el horizonte de planeación. El análisis de estos modelos se organiza en los siguientes grupos de técnicas de solución.

- **Reglas simples** son reglas de decisión para la cantidad económica a ordenar que no están basadas directamente en la optimización de la función de costo, sino que tienen otras características. Entre ellas se puede encontrar:
  1. Demanda de periodo fijo. Este enfoque es equivalente a la regla simple de ordenar "*m* meses de demanda futura".
  2. Cantidad a ordenar para el periodo (COP). Es una modificación de la regla anterior, en la que se usa la "estructura" para seleccionar el periodo fijo.
  3. Lote por lote (L x L). Es un caso especial de la regla de periodo fijo; la cantidad a ordenar es siempre la demanda para un periodo.
- **Reglas heurísticas** son las que están dirigidas al logro de una solución de bajo costo que se pueda considerar buena, es decir, cercana a la óptima.
  1. Método Silver-Meal (SM). El principio de este método es que considera ordenar para varios periodos futuros (*m*). Intenta lograr el costo promedio mínimo por periodo para el lapso de *m* periodos.
  2. Costo unitario mínimo (CUM). Este procedimiento es similar al Silver-Meal. La diferencia radica en que la decisión se basa en el costo variable promedio por unidad en lugar de por periodo.
- **Algoritmo de Wagner-Whitin (WW)** . Este algoritmo tiene el mismo objetivo que algunas de las técnicas anteriores, minimizar el costo variable de inventario, el costo de ordenar y el de mantener inventario durante el horizonte de planeación. La diferencia radica en que se genera una solución de costo mínimo que conduce a una cantidad "óptima" a ordenar. El procedimiento de optimación está basado en

programación dinámica; evalúa todas las maneras posibles de ordenar para cubrir la demanda en cada periodo del horizonte de planeación.

5.2.3 Modelos con demanda probabilística

El modelo con demanda probabilística más conocido es el del "voceador". Este tipo de modelo se adecua muy bien a las ventas de temporada, es decir, donde se pide una sola vez y la demanda es estocástica.

El modelo de inventario estocástico clásico también llamado del "voceador", considera un sólo periodo con demanda estocástica, definida a través de una función de densidad o de probabilidad conocida (distribución continua o discreta).

El enfoque de solución es un análisis económico marginal; se balancean los faltantes y los excedentes. La cantidad óptima a ordenar, Q^*, se encuentra optimizando el costo esperado, ya que se tiene un medio ambiente estocástico.

6 TEORÍAS CLÁSICAS VS TEORÍAS EMERGENTES

Gran parte de la información que se ha presentado en este capítulo, fue desarrollada en la época de las teorías clásicas de administración, en donde el principal objetivo residía en optimizar el inventario. Pero a la luz de las nuevas teorías de producción (teorías emergentes) el inventario no debe ser optimizado, sino reducido todo lo que se pueda. Así, existe un dilema al basar el presente documento en ideas pertenecientes a la era de las teorías clásicas de administración. Si a la luz de las nuevas teorías son obsoletas. Es así como se impone la siguiente pregunta: ¿La teoría clásica de análisis de inventarios y en especial sus modelos matemáticos siguen vigentes? Antes de responder esta pregunta es conveniente mostrar lo que una de las filosofías emergentes más difundidas y aceptadas a nivel mundial (JIT) maneja sobre la gestión de inventarios.

6.1 Justo a tiempo (JIT)

Justo a Tiempo ó Just in Time fue desarrollado por Toyota inicialmente para después trasladarse a muchas otras empresas de Japón y del mundo, ha sido el mayor factor de contribución al impresionante desarrollo de las empresas japonesas. El Justo a Tiempo mas que un sistema de producción es un sistema de inventarios, donde su meta es la de eliminar todo desperdicio. El desperdicio se define por lo general, como todo lo que no sea el mínimo absoluto de recursos de materiales, máquinas y mano de obra requeridos para añadir un valor al producto en proceso.

Los beneficios del JIT son que en la mayoría de los casos, el sistema justo a tiempo da como resultado importantes reducciones en todas las formas de inventario. Dichas formas abarcan los inventarios de piezas compradas, sub-ensambles, trabajos en proceso (WIP, por sus siglas en ingles) y los bienes terminados. Tales reducciones de inventario se logran por medio de métodos mejorados no solo de compras, sino también de programación de la producción.

El Justo a tiempo necesita que se hagan modificaciones importantes a los métodos tradicionales con los que se consiguen las piezas. Se eligen los proveedores preferentes para cada una de las piezas por conseguir. Se estructuran arreglos contractuales especiales para los pedidos pequeños. Estos pedidos se entregan en los momentos exactos en que los necesita el programa de producción del usuario y en las pequeñas cantidades que basten para periodos muy cortos.

Las entregas diarias o semanales de las piezas compradas no son algo inusuales en los sistemas Justo a tiempo. Los proveedores acuerdan, por contrato, entregar las piezas que se ajustan a los niveles de calidad preestablecidos, con lo que se elimina la necesidad de que el comprador inspeccione las piezas que ingresan. El tiempo de llegada de tales entregas es de extrema importancia. Si llegan demasiado pronto, el comprador debe llevar un inventario por separado, pero si llegan demasiado tarde, las existencias pueden agotarse y detener la producción programada.

A menudo quienes compran esas piezas pagan mayores costos unitarios para que se les entreguen de esta forma. Mientras que los costos de oportunidad de estructurar el contrato de compra pueden ser importantes, el costo subsiguiente de conseguir lotes de piezas individuales, diaria o semanalmente, puede reducirse a niveles cercanos a cero. Al no tener que inspeccionar las piezas de ingreso, el comprador puede lograr una mayor calidad en el producto y menores costos de inspección.

La producción de las piezas por fabricar se programa de tal forma que se minimice el inventario de trabajo en proceso (WIP), así como las reservas de bienes terminados.

Debido a que la incertidumbre ha sido eliminada, el control de calidad es esencial para el éxito de la instrumentación del "Justo a Tiempo". Además, ya que el sistema no funcionará si ocurren fallas frecuentes y largas, crea la ineludible necesidad de maximizar el tiempo efectivo y minimizar los defectos. A su vez, se requiere de un programa vigoroso de mantenimiento. La mayoría de las plantas japonesas operan con sólo dos turnos, lo que permite un mantenimiento completo durante el tiempo no productivo y tiene como resultado una tasa mucho más baja de fallas y deterioro de maquinaria que en Estados Unidos.

La presión para eliminar los defectos se hace sentir, no en la programación del mantenimiento, sino en las relaciones de los fabricantes con los proveedores y en el trabajo cotidiano en línea. La producción de justo a tiempo no permite una inspección minuciosa de las partes que arriban. Por ello, los proveedores deben mantener niveles de calidad altos y consistentes, y los trabajadores deben tener la autoridad para detener las operaciones si identifican defectos u otros problemas de producción.

La piedra angular de JIT es el sistema de producción *"Jalar"*, en donde se **hacen las cosas al principio del flujo solamente cuando se pide al final de éste**. En el sistema de producción jalar el flujo físico y el de información van en direcciones opuestas, como se muestra en la Figura 1-5 que se puede ver como una línea de producción en alguna empresa.

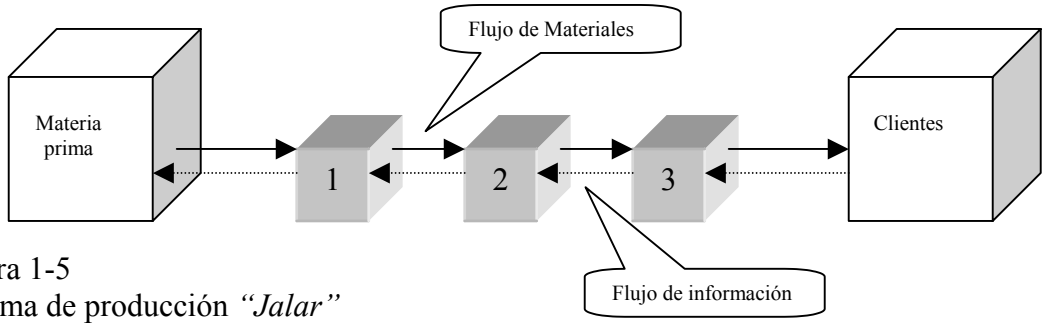


Figura 1-5
Sistema de producción *"Jalar"*

Una línea de producción clásica se compone de operaciones (de manufactura o ensamble), las cuales sufren de interdependencia, es decir, la salida de una operación depende de la entrada de una (o más) operaciones anteriores. Para reducir la interdependencia entre operaciones y mantener la salida de la línea de producción, es común introducir amortiguadores entre las operaciones (Figura 1-6), que no son otra cosa que "*inventarios"*. Estos amortiguadores separan las operaciones y eliminan la interdependencia a menos que el amortiguador se vacié cuando se detiene una máquina anterior.

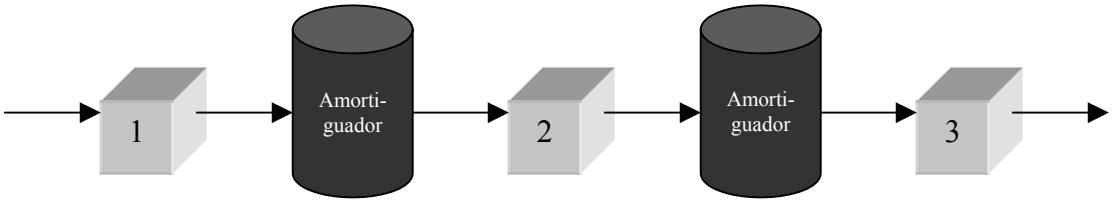


Figura 1-6
Separación de operaciones

6.2 Justificación de la teoría clásica de análisis de inventarios

En este momento ya es posible contestar a la pregunta de si la teoría clásica de análisis de inventarios y en especial sus modelos matemáticos siguen vigentes. La respuesta es que esa teoría y sus modelos "SÍ" siguen vigentes. A continuación se presentan tres justificaciones de la vigencia de las teorías clásicas.

1. Como se vio en el apartado anterior, incluso en un sistema JIT es necesaria la presencia de inventarios, manejados bajo la nomenclatura de *"amortiguadores"*. Además de que JIT no menciona que se deban de eliminar todos los inventarios, sino los inventarios que no son necesarios (y si existen inventarios, que mejor que optimizarlos).
2. La teoría clásica de análisis de inventarios y sus modelos proporcionan una amplia visión al Ingeniero Industrial y Administrador de operaciones. Un gran beneficio que se obtiene al usar diferentes modelos de sistemas de inventarios es la comprensión del comportamiento de estos sistemas, las relaciones entre los diferentes parámetros y variables y las sensibilidad respecto a las inexactitudes en los datos.

3. Por último, en su mayoría existen micro, pequeñas y medianas empresas en México. Es por esto que la implantación de teorías emergentes (como JIT) francamente se encuentre en sus primeras etapas en la generalidad del sistema productivo de nuestro país. De aquí que sea esencial que un Ingeniero Industrial conozca la teoría clásica de análisis de inventarios y sus modelos, porque en el campo laboral se encontrará que numerosas industrias manejan inventarios (muy amplios y repletos).

7 SISTEMAS DE CONTROL

Ya que se ha presentado todo este panorama para el análisis de inventario, debe de haber alguna herramienta para separar lo "importante" de lo "no importante" y así asignar el esfuerzo administrativo. Una de las herramientas más difundidas para alcanzar este fin es el análisis o clasificación ABC, que no es otra cosa que un análisis de Pareto.

Dickie (1951) de General Electric fue el primero en aplicar el principio de Pareto en el análisis de inventarios. Él le llamo análisis ABC; los artículos A son esos pocos artículos "importantes" y los C son los muchos artículos "no importantes". Los artículos B caen entre los A y los C.

7.1 Clasificación ABC

En cada empresa se utilizan diferentes productos, cada uno de ellos con sus propias características, por lo tanto, cada uno de ellos necesita de un manejo particular, dependiendo de su importancia en los procesos de la compañía y de las posibilidades de adquisición. El pensar que todos los productos se deben controlar de la misma manera, es una visión limitada de la realidad, que implica desgaste y sobre costos innecesarios.

El análisis ABC es una manera de clasificar los productos de acuerdo a criterios preestablecidos, la mayor parte de los textos que manejan este tema, toman como criterio el valor de los inventarios y dan porcentajes relativamente arbitrarios para hacer esta clasificación. Por ejemplo, el 10% de los productos representan el 60% de las compras de la empresa, por lo tanto, esta es la zona A, un 40% de los productos el 30%, que serian los que están ubicados en la zona B, el resto (50% de los productos y 10% de las compras) son productos C.

Los valores anteriores son arbitrarios, cada empresa tiene sus particularidades, si alguien decide utilizar este criterio debe ser consciente de las realidades de su empresa. Se debe pensar no solo en los costos, es importante ver otros criterios, lo que es sin duda la principal dificultad en este tipo de análisis. Es innegable, sin embargo que un pequeño porcentaje de productos, desde cualquier criterio, es indispensable para el funcionamiento de la empresa y/o para mejorar su rentabilidad, estos serian clasificados como productos A típicos, y de acuerdo a este punto de vista se van seleccionando los productos de las demás zonas; si uno considera oportuno podría pensarse en la posibilidad de agregar una zona D, para productos realmente intrascendentes y de costo muy bajo.

La siguiente gráfica (Figura 1-7) nos da una visión de la clasificación ABC, no se utilizaron porcentajes en forma explícita, para no caer en la tentación de dogmatizar sobre un valor en particular, la idea es que a los productos de la zona A se le busquen modelos que permitan un control muy fuerte sobre el criterio clave que se esté manejando y a medida que se alejen los productos de esta zona, los modelos puedan ser más flexibles.

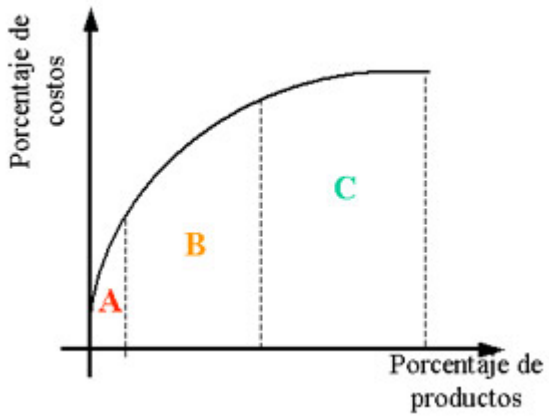


Figura 1-7
Curva ABC

CAPÍTULO 2

Modelo de inventarios de artículos múltiples con restricción de recursos

Este capítulo esta dividido en 2 partes principales; la primera parte se encarga de presentar la teoría existente con respecto al modelo de inventarios de artículos múltiples con restricción de recursos (presupuesto y espacio), y la segunda principia en donde la teoría termina, es decir, se parte de la teoría existente con respecto al modelo y se prosigue con el análisis y desarrollo matemático del mismo, de manera que la programación de los algoritmos de solución matemáticos del modelo sea viable.

1 TEORÍA EXISTENTE

El modelo fundamental EOQ maneja un solo artículo en el sistema, cosa que no es común en la mayoría de las compañías, de hecho, las grandes compañías manejan miles de artículos en sus sistemas de inventarios, de aquí la necesidad de contar con un modelo de inventarios que los optimice.

Una solución trivial al estudiar un sistema que cuenta con más de un artículo en inventario, sería la de analizar cada artículo de forma separada y de esa forma encontrar el lote óptimo de unidades para cada artículo, lo cual sería correcto si no existiese interacción entre los artículos, como compartir recursos comunes. Esos recursos comunes pueden ser presupuesto, capacidad de almacenaje o ambos, y por lo regular son limitados. Entonces el modelo EOQ clásico ya no es válido, ya que su utilización puede llevar a resultados que violen las restricciones de recursos.

Es por esto que se formula el problema como un modelo de optimización restringido y se resuelve usando multiplicadores de Lagrange.

1.1 Multiplicadores de Lagrange

El presente método lo creó uno de los matemáticos más grandes del siglo XVIII, Joseph Lagrange; él utilizó este método en un artículo de Mecánica, a la cual también era afecto, cuando tenía ¡19 años !

Es un extremo (máximo o mínimo) de la función F(x,y), cuando (x,y) vive sobre una curva del plano contenida en el dominio de F, cuya ecuación es $g(x,y) = K$; es decir, (x,y) satisface una condición o restricción.

Por ejemplo: si la gráfica de F es

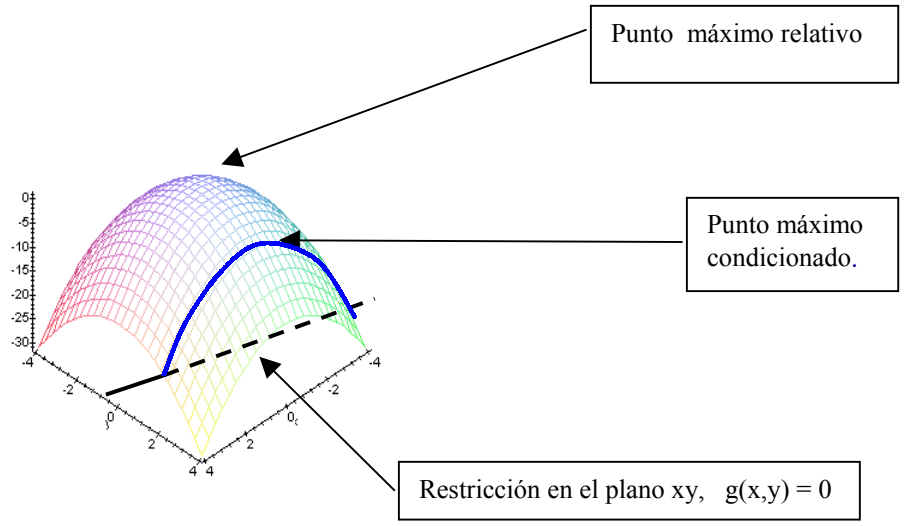


Figura 2-1
Máximo condicionado

Realizando una generalización del método con 3 variables, si se quieren encontrar los extremos relativos de una función:

$$f(x, y, z)$$

$$\textit{sujeta a } g(x, y, z) = A$$

se tiene que formular la Ecuación de Lagrange

$$F(x, y, z, \lambda) = f(x, y, z) + \lambda[g(x, y, z) - A]$$

donde λ es el multiplicador de Lagrange. Y resolver el sistema de ecuaciones:

$$F_x = 0$$
$$F_y = 0$$
$$F_z = 0$$
$$F_\lambda = 0$$

para λ, para después encontrar los valores críticos de x, y, z.

El multiplicador actúa como una penalización para reducir cada x, y, z críticos para minimizar la función f al mismo tiempo que satisfacer la restricción. El método de multiplicadores de Lagrange se puede generalizar para *n* incógnitas con *m* restricciones, por lo tanto puede haber *m* multiplicadores de Lagrange.

Para presentar este enfoque se considerará el caso de una restricción (presupuesto).

1.2 Modelo de inventarios de artículos múltiples con una restricción (presupuesto)

Es necesario que en cualquier momento la inversión total en inventario no sea mayor a C pesos, es decir,

$$\sum_{i=1}^{n} c_i Q_i \leq C$$

en donde n es el número de artículos.

La intención es todavía minimizar el costo total anual promedio,

$$K(Q) = \sum_{i=1}^{n} K_i(Q_i) = \sum_{i=1}^{n} (c_i D_i + \frac{A_i D_i}{Q_i} + h_i \frac{Q_i}{2})$$

Como la ecuación de Lagrange considera tanto el objetivo como la restricción, se tiene

$$K(Q,\lambda) = K(Q) + \lambda \left\{ \sum_{i=1}^{n} c_i Q_i - C \right\}$$

donde λ es el multiplicador de Lagrange. El valor mínimo de K se encuentra tomando derivadas parciales de la función $K(Q,\lambda)$. Para encontrar la solución óptima se requiere de los siguientes pasos:

1. Se resuelve el problema no restringido. Si se satisface la restricción, ésta es la solución óptima.
2. Si no ocurre así, se establece la ecuación para $K(Q,\lambda)$.
3. Se obtiene Q_i* resolviendo las $(n + 1)$ ecuaciones dadas por

$$\frac{\partial K(Q,\lambda)}{\partial Q_i} = 0$$
$$\frac{\partial K(Q,\lambda)}{\partial \lambda} = 0 \qquad \forall \quad i = 1,2,..., n$$

En el caso de que se cuente con la restricción de espacio en lugar de la de presupuesto, el procedimiento de solución es análogo.

1.3 Modelo de inventarios de artículos múltiples con dos restricciones (presupuesto y espacio)[3]

Las dos restricciones más comunes en los sistemas de inventarios son presupuesto y espacio. La formulación del problema general es

[3] Tomado de Daniel Sipper y Robert L. Bulfin Jr., *Planeación y Control de la Producción*, México, ed. McGraw-Hill, 2005, p. 251.

$$\text{minimizar} \quad K(Q) = \sum_{i=1}^{n} K_i(Q_i) = \sum_{i=1}^{n} (c_i D_i + \frac{A_i D_i}{Q_i} + h_i \frac{Q_i}{2})$$

$$\text{sujeta a} \quad \sum_{i=1}^{n} c_i Q_i \leq C \text{ .....restricción de presupuesto}$$

$$\sum_{i=1}^{n} f_i Q_i \leq F \text{ .....restricción de espacio}$$

$$Q_i \geq 0 \qquad \forall \qquad i = 1,2,....., n$$

f_i es el espacio requerido para una unidad del artículo tipo i y F es el espacio total disponible.

Evidentemente este problema es más complicado, ya que una o ambas restricciones pueden ser *activas*. El procedimiento para encontrar la solución óptima es el siguiente:

1. Se resuelve el problema no restringido. Si ambas restricciones se satisfacen, esta solución es la óptima.
2. De otra manera se incluye una de las restricciones (la de presupuesto), y se resuelve el problema de una restricción para encontrar Q_i (i = 1, 2, ..., n). Si la restricción de espacio se satisface, esta solución es la óptima.
3. De otra manera se repite el proceso sólo con la restricción de espacio.
4. Si las dos soluciones con una restricción no llevan a la solución óptima, entonces ambas restricciones son activas, y debe resolverse la ecuación de Lagrange con ambas restricciones:

$$K(Q, \lambda_1, \lambda_2) = K(Q) + \lambda_1 \left\{ \sum_{i=1}^{n} c_i Q_i - C \right\} + \lambda_2 \left\{ \sum_{i=1}^{n} f_i Q_i - F \right\}$$

Se obtiene Q_i* resolviendo las (n + 2) ecuaciones simultaneas siguientes:

$$\frac{\partial K(Q, \lambda_1, \lambda_2)}{\partial Q_i} = 0$$

$$\frac{\partial K(Q, \lambda_1, \lambda_2)}{\partial \lambda_1} = 0 \qquad \forall \quad i = 1,2,..., n$$

$$\frac{\partial K(Q, \lambda_1, \lambda_2)}{\partial \lambda_2} = 0$$

2 ANÁLISIS MATEMÁTICO

Una considerable cantidad de autores han dedicado su tiempo a escribir sobre el Análisis de Inventarios, sin embargo la información con respecto al modelo de artículos múltiples con restricción de recursos es más bien escasa. A nivel universitario el análisis del modelo se queda en la información presentada en el apartado anterior, y en libros muy especializados

de nivel de maestría y doctorado sólo se abunda un poco más en el tema, realizando algunos cambios al modelo y ciertas recomendaciones.

La programación de los algoritmos de solución presentados hasta el momento resultaría sumamente complicada y poco eficiente en el caso de que las restricciones sean activas. De aquí que en este escrito se haya decidido profundizar en el análisis matemático del modelo por cuenta propia.

2.1 Restricción de presupuesto

El objetivo continua siendo el siguiente:

$$\text{minimizar} \quad K(Q) = \sum_{i=1}^{n} K_i(Q_i) = \sum_{i=1}^{n} (c_i D_i + \frac{A_i D_i}{Q_i} + h_i \frac{Q_i}{2})$$

$$\text{sujeta a} \quad \sum_{i=1}^{n} c_i Q_i \leq C \text{ .....restricción de presupuesto}$$

$$Q_i \geq 0 \qquad \forall \qquad i = 1,2,....., n$$

Formulando la ecuación de Lagrange, se tiene

$$K(Q,\lambda) = K(Q) + \lambda \left\{ \sum_{i=1}^{n} c_i Q_i - C \right\}$$

La derivada parcial de la ecuación de Lagrange con respecto a Q_i es

$$\frac{\partial K(Q,\lambda)}{\partial Q_i} = \frac{1}{2} h_i - \frac{A_i D_i}{Q_i^2} + \lambda c_i = 0$$

Despejando a Q_i

$$Q_i = \sqrt{\frac{2 A_i D_i}{h_i + 2\lambda c_i}} \text{ .....Ec. A}$$

Derivando parcialmente la ecuación de Lagrange con respecto a λ

$$\frac{\partial K(Q,\lambda)}{\partial \lambda} = \sum_{i=1}^{n} c_i Q_i - C = 0$$

$$\sum_{i=1}^{n} c_i Q_i = C \text{ .....Ec. B}$$

Sustituyendo la Ec. A en la Ec. B

$$\sum_{i=1}^{n} c_i \sqrt{\frac{2A_i D_i}{h_i + 2\lambda c_i}} = C$$

Despejando el multiplicador de Lagrange λ

$$\lambda = \frac{1}{2C^2}\left(\sum_{i=1}^{n} \sqrt{2A_i D_i c_i}\right)^2 - \frac{i}{2} \text{ .....Fórmula 2-1}$$

Con la fórmula anterior se obtiene el multiplicador de Lagrange λ, que sólo hay que sustituir en la Ec. A para obtener Q_i*.

2.2 Restricción de espacio

El objetivo es:

$$\text{minimizar} \quad K(Q) = \sum_{i=1}^{n} K_i(Q_i) = \sum_{i=1}^{n} (c_i D_i + \frac{A_i D_i}{Q_i} + h_i \frac{Q_i}{2})$$

$$\text{sujeta a} \quad \sum_{i=1}^{n} f_i Q_i \le F \text{ .....restricción de espacio}$$

$$Q_i \ge 0 \qquad \forall \qquad i = 1,2,....., n$$

Formulando la ecuación de Lagrange, se tiene

$$K(Q,\lambda) = K(Q) + \lambda\left\{\sum_{i=1}^{n} f_i Q_i - F\right\}$$

La derivada parcial de la ecuación de Lagrange con respecto a Q_i es

$$\frac{\partial K(Q,\lambda)}{\partial Q_i} = \frac{1}{2} h_i - \frac{A_i D_i}{Q_i^2} + \lambda f_i = 0$$

Despejando a Q_i

$$Q_i = \sqrt{\frac{2A_i D_i}{h_i + 2\lambda f_i}} \text{ .....Ec. A}$$

Derivando parcialmente la ecuación de Lagrange con respecto a λ

$$\frac{\partial K(Q,\lambda)}{\partial \lambda} = \sum_{i=1}^{n} f_i Q_i - F = 0$$

$$\sum_{i=1}^{n} f_i Q_i = F \text{ .....Ec. B}$$

Sustituyendo la Ec. A en la Ec. B

$$\sum_{i=1}^{n} f_i \sqrt{\frac{2A_i D_i}{h_i + 2\lambda f_i}} = F$$

$$\sum_{i=1}^{n} f_i \sqrt{\frac{2A_i D_i}{ic_i + 2\lambda f_i}} - F = 0 \text{ .....Fórmula 2-2}$$

Esta última ecuación se puede resolver para λ mediante aproximaciones, y ya con λ obtener Q_i*.

Ahora bien, la maestra Isabel Patricia Aguilar Juárez sugiere una forma alterna de resolver el problema, consistente en eliminar de la función de costo total anual promedio el costo por mantener el inventario, ya que este se considera como una función lineal.[4]

Si no se toma en consideración el término correspondiente al costo de mantener el inventario, la última ecuación se simplificaría de la siguiente manera:

$$\sum_{i=1}^{n} f_i \sqrt{\frac{A_i D_i}{\lambda f_i}} - F = 0$$

y por tanto,

$$\sum_{i=1}^{n} \sqrt{\frac{A_i D_i f_i}{\lambda}} = F$$

Y despejando λ, se obtiene

$$\lambda = \frac{1}{F^2}\left(\sum_{i=1}^{n} \sqrt{A_i D_i f_i}\right)^2 \text{ .....Fórmula 2-3}$$

La ecuación que define a Q_i se simplificaría así:

[4] Adaptado de Isabel Patricia Aguilar Juárez, *El problema de inventario: con multiproducto (Tesis de Maestría)*, México, Facultad de Ingeniería UNAM, 1996, p. 43.

$$Q_i = \sqrt{\frac{A_i D_i}{\lambda f_i}}$$ .....Fórmula 2-4

Por lo tanto, bastaría con sustituir λ en la última ecuación para encontrar Q_i*.

Evidentemente el método alternativo de resolución propuesto por la profesora Aguilar sería bastante más sencillo que el de encontrar λ mediante aproximaciones, pero ¿Qué tan confiable será?

Para determinar cual de los 2 métodos es el más confiable, lo más acertado será emplearlos en la resolución de un problema de inventarios típico, en el que se aplique el modelo de artículos múltiples con restricción de recursos.

Ejemplo. HiEnd, una pequeña compañía de computadoras, compra dos tipos de lectoras de discos. La compañía no tiene mucho espacio para almacenar las lectoras de discos. Suponga que cada tipo de lectora requiere de 10 y 8 unidades de espacio respectivamente, y se cuenta con un total de 500 unidades de espacio en el almacén. El precio de estas dos lectoras es de $50 y $80, respectivamente, y su demanda anual es 250 y 484 unidades, respectivamente. La empresa tiene un gasto de $50 para procesar la orden de cualquiera de estas dos lectoras, y el gerente usa un 20% anual para las evaluaciones financieras. ¿Cuál será la cantidad óptima de lectoras a ordenar de cada tipo?[5]

Datos

- $A_1 = A_2 = \$50$
- $i = 20\ \%$ *anual*
- $F = 500$ *unidades de espacio*
- $c_1 = \$50$
- $f_1 = 10$ *unidades de espacio*
- $D_1 = 250$ *unidades por año*
- $c_2 = \$80$
- $f_2 = 8$ *unidades de espacio*
- $D_2 = 484$ *unidades por año*

Resolución

Está se llevará al cabo con la ayuda del programa de análisis matemático *Maple X*.

[5] Adaptado de Daniel Sipper y Robert L. Bulfin Jr., *Planeación y Control de la Producción*, México, ed. McGraw-Hill, 2005, p. 249 y 252.

Se va a empezar por introducir la fórmula 3-2, que es la que define el primer método para encontrar las cantidades óptimas de artículos.

```
> Ec1:=(f1*(sqrt((2*A1*D1)/((i*c1)+(2*lambda1*f1)))))+(f2*(sqrt((2*A2*D2)/((i*c2)+(2*lam
  bda1*f2)))))-F=0;
```

$$Ec1 := f1\sqrt{2}\sqrt{\frac{A1\ D1}{c1\ i + 2\,\lambda1\ f1}} + f2\sqrt{2}\sqrt{\frac{A2\ D2}{c2\ i + 2\,\lambda1\ f2}} - F = 0$$

Ahora se va introducir la fórmula 3-3, que define el segundo método para encontrar las cantidades óptimas de artículos.

```
> lambda2:=(1/(F^2))*((sqrt(A1*D1*f1)+sqrt(A2*D2*f2))^2);
```

$$\lambda2 := \frac{\left(\sqrt{A1\ D1\ f1} + \sqrt{A2\ D2\ f2}\right)^2}{F^2}$$

Para evaluar cual de las dos fórmulas arroja mejores cantidades óptimas de lectoras a ordenar de cada tipo, antes que nada hay que obtener los multiplicadores de lagrange que se consiguen con cada método, así que hay que sustiur el valor de cada parámetro.

```
> A1:=50;A2:=50;i:=0.2;F:=500;c1:=50;f1:=10;D1:=250;c2:=80;f2:=8;D2:=484;
```

$$A1 := 50$$
$$A2 := 50$$
$$i := .2$$
$$F := 500$$
$$c1 := 50$$
$$f1 := 10$$
$$D1 := 250$$
$$c2 := 80$$
$$f2 := 8$$
$$D2 := 484$$

El siguiente es el multiplicador de Lagrange que se obtiene con el primer método.

```
> lambda1:=solve(Ec1,lambda1);
```

$$\lambda1 := 1.760290933$$

Y el siguiente es el multiplicador de Lagrange que se obtiene con el método sugerido por la profesora Aguilar.

```
> lambda2:=lambda2;
```

$$\lambda2 := \frac{1}{250000}\left(\sqrt{125000} + \sqrt{193600}\right)^2$$

Al pasar el término anterior a decimales el multiplicador de Lagrange queda de la siguiente forma.

```
> lambda2:=2.518907935;
```

$$\lambda2 := 2.518907935$$

Para el primer método la cantidad óptima de lectoras a ordenar de cada tipo se encuentra a continuación.

```
> Q11:=sqrt((2*A1*D1)/((i*c1)+(2*lambda1*f1)));Q21:=sqrt((2*A2*D2)/((i*c2)+(2*lambda1*f2
  )));
```

$$Q11 := 23.51650810$$
$$Q21 := 33.10436488$$

Para el segundo método la cantidad óptima de lectoras a ordenar de cada tipo se encuentra a continuación.

```
> Q12:=sqrt((A1*D1)/(lambda2*f1));Q22:=sqrt((A2*D2)/(lambda2*f2));
```

$$Q12 := 22.27659757$$
$$Q22 := 34.65425304$$

Por último, se debe obtener el costo total anual promedio asociado con cada método.

```
> K1:=((c1*D1)+((A1*D1)/Q11)+((i*c1*Q11)/2))+((c2*D2)+((A2*D2)/Q21)+((i*c2*Q21)/2));K2:=
  ((c1*D1)+((A1*D1)/Q12)+((i*c1*Q12)/2))+((c2*D2)+((A2*D2)/Q22)+((i*c2*Q22)/2));
```

$$K1 := 52864.98039$$
$$K2 := 52868.07097$$

```
>
```

Page 1

Conclusión

Comparando los métodos, por el propuesto por la maestra Aguilar se llega a lotes óptimos que arrojan un costo total anual promedio *K(Q)*, $3.1 más alto que el del primer método. Esta cantidad se podría considerar como despreciable, pero en el caso de que en inventarios

se manejaran no pocas clases de artículos (lo que es común) la eficiencia del método alternativo decrecería considerablemente.

2.3 Ambas restricciones

El objetivo es:

$$\text{minimizar} \quad K(Q) = \sum_{i=1}^{n} K_i(Q_i) = \sum_{i=1}^{n} (c_i D_i + \frac{A_i D_i}{Q_i} + h_i \frac{Q_i}{2})$$

$$\text{sujeta a} \quad \sum_{i=1}^{n} c_i Q_i \le C \text{ .....restricción de presupuesto}$$

$$\sum_{i=1}^{n} f_i Q_i \le F \text{ .....restricción de espacio}$$

$$Q_i \ge 0 \qquad \forall \qquad i = 1,2,....., n$$

Formulando la ecuación de Lagrange, se tiene

$$K(Q, \lambda_1, \lambda_2) = K(Q) + \lambda_1 \left\{ \sum_{i=1}^{n} c_i Q_i - C \right\} + \lambda_2 \left\{ \sum_{i=1}^{n} f_i Q_i - F \right\}$$

La derivada parcial de la ecuación de Lagrange con respecto a Q_i es

$$\frac{\partial K(Q, \lambda_1, \lambda_2)}{\partial Q_i} = \frac{1}{2} h_i - \frac{A_i D_i}{Q_i^2} + \lambda_1 c_i + \lambda_2 f_i = 0$$

Despejando a Q_i

$$Q_i = \sqrt{\frac{2 A_i D_i}{h_i + 2(\lambda_1 c_i + \lambda_2 f_i)}} \text{ .....Ec. A}$$

Derivando parcialmente la ecuación de Lagrange con respecto a λ_1 y λ_2

$$\frac{\partial K(Q, \lambda_1, \lambda_2)}{\partial \lambda_1} = \sum_{i=1}^{n} c_i Q_i - C = 0$$

$$\sum_{i=1}^{n} c_i Q_i - C = 0 \text{ .....Ec. B}$$

$$\frac{\partial K(Q, \lambda_1, \lambda_2)}{\partial \lambda_2} = \sum_{i=1}^{n} f_i Q_i - F = 0$$

$$\sum_{i=1}^{n} f_i Q_i - F = 0 \text{ .....Ec. C}$$

Sustituyendo la Ec. A en la Ec. B y C

$$\left.\begin{aligned} \left(\sum_{i=1}^{n} c_i \sqrt{\frac{2A_i D_i}{h_i + 2(\lambda_1 c_i + \lambda_2 f_i)}}\right) - C = 0 \\ \left(\sum_{i=1}^{n} f_i \sqrt{\frac{2A_i D_i}{h_i + 2(\lambda_1 c_i + \lambda_2 f_i)}}\right) - F = 0 \end{aligned}\right\} \text{ Sist. de Ec. 2-1}$$

Este sistema de ecuaciones se puede resolver para λ_1 λ_2 mediante aproximaciones, y ya con λ_1 λ_2 obtener Q_i*.

CAPÍTULO 3

Programación

1 OBJETIVO DE LA PROGRAMACIÓN

El objetivo principal de la programación es utilizar la computadora como una herramienta para la resolución de problemas.

Antes de seguir con el objetivo de la programación resulta conveniente definir algunos términos comunes a ella:

- **Computadora**: Dispositivo que acepta **datos**, que permite **procesarlos** de acuerdo con un **programa** almacenado y que genera unos resultados.
- **Datos**: Representación formalizada de hechos, conceptos, características, etc., que son susceptibles de ser comunicados, interpretados o procesados por medios humanos o automáticos.
- **Información**: Significado que el hombre asigna a los datos por medio de las convenciones utilizadas en su representación. Información = datos procesados y organizados.
- **Procesamiento de datos (de información)**: Realización sistemática de operaciones (cálculos, ordenaciones, etc.)sobre los datos con el fin de obtener información organizada, significativa y útil.
- **Sistema** (de procesamiento de datos, SPD): Conjunto organizado de recursos humanos, de equipamiento, etc., que interaccionan con el fin de cumplir un conjunto de funciones específicas (procesamiento de datos).
- **Programa** (de computadora): Secuencia de instrucciones, cada una de las cuales especifica las operaciones que debe realizar la computadora para la resolución de un problema.

La **resolución de un problema** (mediante una computadora) se refiere al proceso consistente en partir de la descripción de un problema (habitualmente en lenguaje natural, y expresada en términos propios del dominio del problema) y desarrollar un programa de computadora que resuelva dicho problema. Este proceso exige los siguientes pasos: análisis del problema, diseño o desarrollo de un algoritmo, transformación del algoritmo en un programa (codificación), ejecución y validación del programa.

2 CONCEPTO DE ALGORITMO

> *"Conjunto de instrucciones que especifican la secuencia ordenada de operaciones a realizar para resolver un problema. En otras palabras, un algoritmo es un método o fórmula para la resolución de un problema."*

Conviene destacar que un algoritmo es independiente tanto del lenguaje de programación en que se exprese como de la computadora en la que se ejecute. De hecho, el término algoritmo es muy anterior a la era informática: proviene de *Mohammed al-Khowârizmî* (su apellido se tradujo al latín en la palabra *algoritmus*), matemático persa del siglo IX que enunció paso a paso las reglas para sumar, restar, multiplicar y dividir números decimales.

3 PRIMITIVAS DE CONTROL

¿Cómo es posible programar?

Porque se dispone de una notación algorítmica que permite:

- Describir las operaciones puestas en juego (acciones).
- Describir los objetos manipulados por el algoritmo (datos/informaciones).
- Controlar la realización de las acciones, indicando el modo de organización de estas acciones en el tiempo (mediante las primitivas de control).

Debe subrayarse que la notación algorítmica no es en ningún caso un lenguaje de programación. Lo esencial de la notación algorítmica es que las acciones elegidas sean las necesarias y suficientes para expresar todo algoritmo. En este sentido, autores como E. Dijkstra, C.A.R. Hoare y otros consolidaron a finales de los años 60 los fundamentos de la Programación Estructurada, mediante la proposición del uso de un conjunto de construcciones lógicas (secuencia, decisión e iteración) con las que se podría escribir cualquier programa.

A continuación se introducen las acciones disponibles en la notación algorítmica, indicándose tanto una posible forma de escribirlas como una interpretación informal de las mismas.

3.1 Acción elemental: ejecución de un acontecimiento elemental

Considerando que en última instancia es una computadora la máquina que ejecutará las instrucciones de un programa, las acciones elementales vendrán determinadas por las tareas que puede realizar una computadora. Básicamente estas tareas son las siguientes: realizar una serie de operaciones (aritméticas o lógicas) sobre unos operandos, posibilidad de obtener dichos operandos a través de un dispositivo de entrada o de la memoria de la computadora, y posibilidad de almacenar los resultados de dichas operaciones en memoria o enviarlos hacia un dispositivo de salida.

3.2 Composición secuencial: ejecución incondicional de una secuencia de acciones

Cuando se deben ejecutar sucesivamente distintas acciones, se escribirá la lista de dichas acciones en el orden en el que deban ser ejecutadas. Las acciones se separarán por un "punto y coma".

<ACCIÓN1>;
<ACCIÓN2>;
...
<ACCIÓNN>;

3.3 Composición condicional: ejecución condicional de una acción

La composición condicional permite expresar que una cierta acción sólo debe realizarse bajo una cierta condición. Se escribe como sigue:

SI <CONDICIÓN>
ENTONCES <ACCIÓN>
FINSI

La ejecución de esta acción provoca en primer lugar la observación del estado del sistema para detectar si la condición indicada se satisface o no. Si se satisface, es decir, si la condición tiene como resultado el valor verdadero (cierto), la acción indicada se ejecuta a continuación. Si por el contrario la condición tiene por resultado el valor falso, la ejecución de la acción termina tras la evaluación de dicha condición.

3.4 Composición alternativa: ejecución alternativa de una entre dos acciones

La composición alternativa permite expresar que debe ejecutarse una acción bajo cierta condición u otra acción bajo la condición contraria. Se escribe como sigue:

SI <CONDICIÓN>
ENTONCES <ACCIÓN1>
SINO <ACCIÓN2>
FINSI

La ejecución de esta acción provoca en primer lugar la evaluación de la condición indicada (es decir, la observación del estado del proceso). A la evaluación de la condición sigue la ejecución de <ACCIÓN1> si se satisface la condición (el resultado de evaluar la condición es verdadero) o bien la ejecución de <ACCIÓN2> si no se satisface la condición (el resultado de evaluar la condición es falso).

3.5 Composición selectiva (exclusiva): ejecución condicional de una entre varias acciones

La composición selectiva permite, como las composiciones condicional y alternativa, vincular la ejecución de una acción a la observación de una condición. En el primer caso se vinculaba una condición con una acción. En el segundo se asociaban dos condiciones complementarias con dos acciones. En la composición selectiva, se asocia un conjunto de acciones con un conjunto de condiciones que se excluyen mutuamente. Esta acción se escribirá en la notación algorítmica como sigue:

CASO <INDICADOR> **SEA**
<VALOR1>**:** <ACCIÓN1>**;**
<VALOR2>**:** <ACCIÓN2>**;**
...
<VALORN>**:** <ACCIÓNN>**;**
[**ENOTROCASO** <ACCIÓNX>]
FINCASO

Esta estructura se introduce para expresar de una forma cómoda las diversas condiciones y su relación con cada una de las acciones. La ejecución de esta primitiva de control consiste en la ejecución de aquella acción cuyo valor asociado coincida con el valor observado del indicador que aparece en la primitiva. Debe subrayarse también que para que esté bien construida esta estructura debe ejecutarse una y sólo una acción de entre todas las que se enumeran.

Observaciones:

- Se requiere que el indicador tenga, en el momento de su evaluación, un valor del conjunto enumerado en la composición.
- Opcionalmente, se puede indicar una acción por defecto que se realizará en el caso de que el indicador tome un valor distinto de los N valores explícitamente enumerados en la composición.

3.6 Composición iterativa: ejecución múltiple de una sola acción

La composición iterativa (en la forma MIENTRAS) permite expresar que se debe ejecutar una misma acción cero, una o más veces consecutivamente. Se escribe así:

MIENTRAS <CONDICIÓN> **HACER**
<ACCIÓN>
FINMIENTRAS

El número de ejecuciones de la acción no se expresa directamente. De hecho, la condición que aparece en la composición indica la observación que debe hacerse para que continúe la iteración (condición de permanencia).

La ejecución de esta primitiva de control provoca la evaluación de dicha condición, que en caso de satisfacerse (dar como resultado el valor verdadero) es seguida por la ejecución de la acción, y así sucesivamente hasta que el resultado de la evaluación de la condición de permanencia sea falso.

Existe una segunda forma de composición iterativa, la forma REPETIR, que permite expresar la ejecución de una acción una o más veces consecutivamente.

REPETIR
<ACCIÓN>
HASTA <CONDICIÓN>

Al igual que en la estructura MIENTRAS, el número de ejecuciones de la acción no se expresa directamente, pero a diferencia de aquélla, la condición que aparece en la composición REPETIR indica la observación que debe hacerse para que termine la iteración (condición de salida).

La ejecución de esta primitiva de control provoca la ejecución de la acción, para a continuación evaluar la condición de salida, y así sucesivamente hasta que el resultado de la condición sea verdadero. Conviene recalcar que la acción indicada se ejecuta como mínimo una vez.

Por último, existe una tercera forma de composición iterativa, la forma DESDE, que permite expresar la ejecución de una acción un número determinado de veces.

4 DESARROLLO DE LOS LENGUAJES

Un **lenguaje de programación** es un lenguaje artificial que se utiliza para expresar programas de computadora. Cada computadora, según su diseño, "entiende" un cierto conjunto de instrucciones elementales (lenguaje máquina). No obstante, para facilitar la tarea del programador, se dispone también de lenguajes de alto nivel más fáciles de manejar y que no dependen del diseño específico de cada computadora. Los programas escritos en un lenguaje de alto nivel no podrán ser ejecutados por una computadora mientras no sean traducidos al lenguaje propio de ésta.

4.1 Lenguajes de bajo nivel

Existen diferentes generaciones de lenguajes de programación. La primera generación se denomina **lenguaje de máquina o de primera generación,** en donde cada instrucción se escribía en lenguaje binario, es decir, utilizando exclusivamente unos y ceros (1,0), evidentemente este nivel de programación no permitía seguir fácilmente la secuencia de instrucciones de un programa (se elaboraba una instrucción de programa en términos de ceros y unos por cada operación que se iba a realizar).

El siguiente nivel o **lenguaje ensamblador** utilizó símbolos mnemotécnicos que representan series de unos y ceros (instrucciones). En esta etapa se logró una mejoría, pero persiste la dificultad de programación. Estos lenguajes de máquina y ensamblador se utilizaban hasta antes de 1970 para el desarrollo de programas de aplicaciones y software de sistemas con la idea de utilizar más eficientemente la computadora mediante **lenguajes de bajo nivel,** aunque fuese muy tardada la programación.

4.2 Lenguajes de alto nivel

En este tercer nivel el programador pudo escribir cierto número de operaciones en una sola instrucción. Estos lenguajes marcan su aparición en 1955 con el FORTRAN (Formula Traslation) que tuvo como propósito facilitar a científicos e ingenieros la programación de la computadora. Para 1959 la Dra. Grace Hopper presentó su lenguaje COBOL (Common Business Oriented Language) que se ha aplicado a problemas administrativos y de negocios. La tendencia siempre ha sido reducir el número de instrucciones. Hasta la última fecha dada, casi todo el software que se producía era específico de cada computadora (o hardware) que se fabricaba y difícilmente podía utilizarse en otro equipo.

La memoria de las computadoras era muy costosa y limitada hasta la tercera generación, por lo que los programadores y analistas se preocupaban por la eficiencia y escribían una y otra vez su software para optimizar el uso de dicha memoria; sin embargo, al abaratarse el costo del hardware ahora lo más importante y costoso es el tiempo del programador o analista y por consecuencia en algunos casos se ha perdido la necesidad de hacer uso eficiente de la memoria de la máquina, produciéndose desarrollos de software no óptimos desde ese punto de vista.

Dentro del grado de sofisticación de los lenguajes de alto nivel se tiene la siguiente clasificación por orden de complejidad:

- Lenguajes orientados a procedimientos.
- Lenguajes orientados a problemas (áreas de trabajo específicas).
- Lenguajes de consulta.
- Generadores de aplicaciones.

El nivel de un lenguaje es inversamente proporcional al número de instrucciones necesarias que se requieren para resolver un problema dado, es decir, en una sola instrucción se indica de manera implícita toda una serie de acciones a resolver.

4.2.1 Lenguajes orientados a procedimientos

Se clasifican como:

- Científicos (manejo de matrices, solución de problemas recursivos, análisis estadísticos, etc.).
- Para negocios (nóminas, balances, estados financieros, etc.).
- De aplicaciones múltiples (tanto científicas como de negocios).

Los **lenguajes científicos** son:

FORTRAN, desarrollado por un grupo dirigido por John Backus. La primera versión data de 1955 y se considera como el primer lenguaje de alto nivel.; de él se han escrito diversas versiones, se le ha empleado en la solución de múltiples problemas de ingeniería. Cuenta

con uno de los mejores manejos de punto flotante y opera con números complejos del tipo (a + bi).

LabVIEW, es un ambiente de desarrollo de programas mucho más moderno que C o BASIC. Es diferente a la mayoría de los lenguajes de programación en un aspecto; mientras otros sistemas de programación usan lenguaje de texto para crear líneas de código, LabVIEW usa un lenguaje de programación gráfico llamado G, para crear programas en forma de diagramas de bloque.

Los **lenguajes para negocios** incluyen una gran capacidad para almacenar, recuperar y manipular información alfanumérica; los más conocidos son:

COBOL, desarrollado por Grace Hopper de la Sperry Rand y R. W. Berner de la IBM. Aparece en 1959 la primera versión que emplea una sintaxis que se parece mucho a órdenes dadas en idioma inglés; continúa mejorándose hasta la fecha y es el lenguaje en el que están escritos la mayoría de programas de negocios.

RPG (desarrollado a partir de 1964 con la finalidad de generar programas de informes). Se maneja por medio de menús desplegables, de forma interactiva y puede cubrir ampliamente todas las necesidades de procesamiento de información.

Los **lenguajes de aplicaciones múltiples** se han desarrollado con la idea de simplificar el medio ambiente de programación y cubrir la necesidades de programación de cada tipo. Como casos más representativos están:

BASIC (*Beginner´s all purpose symbolic instruction code*, data de 1965) fue desarrollado para las grandes computadoras por John Kemeny y Kenneth Kurz del colegio de Darmouth y posteriormente adoptado como el lenguaje más popular entre las microcomputadoras; se puede decir que, casi sin lugar a dudas, que es el lenguaje más fácil de aprender y utilizar para todo tipo de aplicaciones, tanto científicas como de negocios. Tiene gran poder y flexibilidad para todo tipo de proceso de información o solución de problemas con la computadora.

Visual-Basic, VB es una herramienta de diseño de aplicaciones para Windows, en la que estas se desarrollan en una gran parte a partir del diseño de una interface gráfica. En una aplicación Visual - Basic el programa está formado por una parte de código puro, y otras partes asociadas a los objetos que forman la interface gráfica. Es por tanto un termino medio entre la programación tradicional, formada por una sucesión lineal de código estructurado, y la programación orientada a objetos. Combina ambas tendencias. Ya que no se puede decir que VB pertenezca por completo a uno de esos dos tipos de programación, se debe inventar una palabra que la defina : PROGRAMACION VISUAL.

C, Martin Richards desarrolló el lenguaje BCPL que a su vez influyó en Ken Thompson para desarrollar el lenguaje B; derivado de ello Dennis Ritchie desarrolla en una DEC PDP-11 el lenguaje C al iniciarse la década de los años setenta. Este lenguaje pretende dar un substituto eficiente al lenguaje de bajo nivel conocido como ensamblador, ubicándose en un nivel intermedio entre un lenguaje bajo y uno de alto nivel. Su característica más

importante es permitir al programador manipular bits, bytes y direcciones de memoria; es por ello un lenguaje de gran aceptación en las áreas de computación y se ha constituido como poderosa herramienta para el desarrollo de software. En 1980 Bjarne Stroustrup de los laboratorios Bell, inspirado en el lenguaje Simula67 adicionó las características de la programación orientada a objetos (incluyendo las ventajas de una biblioteca de funciones orientada a objetos) y lo denominó C con clases. Para 1983 dicha denominación cambió a la de C++.

4.2.2 Lenguajes orientados a áreas de trabajo específicas

Estos lenguajes se conocen comúnmente como *paquetes* y están diseñados para resolver los problemas de un área particular de aplicación. Generalmente sólo demandan la información de entrada al problema y con ella presentan el esquema de solución. Estos lenguajes se han producido para casi todas las aplicaciones posibles, como muestra se mencionan:

- COGO (análisis de esfuerzos en edificios y puentes).
- SPSS (análisis estadístico).
- Microsoft Word (procesamiento de palabras).

4.2.3 Lenguajes de consulta

Los lenguajes de consulta generan automáticamente el procedimiento de solución de un problema, es decir, el programador sólo indica qué hacer y no cómo hacerlo instrucción por instrucción. Las instrucciones básicas de estos lenguajes manipulan matemáticamente los datos, dan formato a resultados o informes y producen la salida en el orden deseado. Como ejemplo de este tipo de lenguajes:

- DATA EASE
- EASYTRIEVE

4.2.4 Generadores de aplicaciones

En este tipo de lenguaje, las necesidades de un sistema se indican seleccionando funciones previamente programadas, sin necesidad de dar instrucciones a nivel procedimiento. Por ejemplo:

- ORACLE

5 LENGUAJES DE PROGRAMACIÓN MÁS POPULARES

5.1 Visual Basic

5.1.1 Antecedentes históricos

El lenguaje de programación BASIC (*Beginner's All purpose Symbolic Instruction Code*) nació en el año 1965 como una herramienta destinada a principiantes, buscando una forma sencilla de realizar programas, empleando un lenguaje casi igual al usado en la vida ordinaria (en inglés), y con instrucciones muy sencillas y escasas. Teniendo en cuenta el año de su nacimiento, este lenguaje cubría casi todas las necesidades para la ejecución de programas. Téngase en cuenta que las máquinas existentes en aquella época estaban estrenando los transistores como elementos de conmutación, los ciclos de trabajo llegaban a la impensable cifra de 10.000 por segundo y la memoria no pasaba de unos pocos k´s en toroides de ferrita.

La evolución del BASIC por los años 70´s fue escasa, dado el auge que tomaron en aquella época lenguajes de alto nivel como FORTRAN y el COBOL. En 1978 se definió una norma para unificar los Basics existentes creándose la normativa BASIC STANDARD.

Con la aparición de las primeras computadoras personales, dedicadas comercialmente al usuario particular, allá por la primera mitad de los ochentas, el BASIC resurgió como lenguaje de programación pensado para principiantes, y muchas de estas pequeñas computadoras domésticas lo usaban como único sistema operativo (Sinclair, Spectrum, Amstrad).

Con la popularización de la PC, salieron varias versiones del BASIC que funcionaban en este tipo de ordenadores (Versiones BASICA, GW-BASIC), pero todas estas versiones del BASIC no hicieron otra cosa que terminar de rematar este lenguaje. Los programadores profesionales no llegaron a utilizarlo, debido a las desventajas de este lenguaje respecto a otras herramientas (PASCAL, C, CLIPPER). El BASIC con estas versiones para PC llegó incluso a perder crédito entre los profesionales de la informática.

Las razones para ello eran obvias:

- No era un lenguaje estructurado.
- No existían herramientas de compilación fiables.
- No disponía de herramientas de intercambio de información.
- No tenía librerías.
- No se podía acceder al interior de la máquina.
- Un largo etcétera de desventajas respecto a otros lenguajes de programación.

Tal fue ese abandono por parte de los usuarios, que la aparición del Quick-BASIC de Microsoft, una versión ya potente del BASIC, que corregía casi todos los defectos de las versiones anteriores pasó prácticamente inadvertida, a no ser porque las últimas versiones del sistema operativo MS-DOS incluían una versión de Quick-BASIC algo recortada (Q-

Basic) como un producto más dentro de la amplia gama de ficheros ejecutables que acompañan al sistema operativo, y aprovecha de él el editor de textos (Cada vez que se llama al EDIT se esta corriendo el editor del Q-Basic).

Esta versión del popular BASIC ya es un lenguaje estructurado, lo que permite crear programas modularmente, mediante subrutinas y módulos, capaz de crear programas ya competitivos con otros lenguajes de alto nivel. Sin embargo llegaba tarde, pues los entornos MS-DOS estaban ya superados por el entorno gráfico Windows.

Sin embargo algo había en el BASIC que tentaba a superarse: su gran sencillez de manejo. Si a esto se le añade el entorno gráfico Windows, el aprovechamiento al máximo de las posibilidades de Windows en cuanto a intercambio de información, de sus librerías, de sus drivers y controladores, manejo de bases de datos, etc. el producto resultante puede ser algo que satisfaga todas las necesidades de programación en el entorno Windows. La suma de todas estas cosas es Visual Basic. Esta herramienta conserva del BASIC de los años 80 únicamente su nombre y su sencillez, y tras su lanzamiento al mercado, la aceptación a nivel profesional hizo borrar por fin el "mal nombre" asociado a la palabra BASIC.

Actualmente se está comercializando la versión 7.0 de este producto. Desde su salida al mercado, cada versión supera y mejora la anterior. Dados los buenos resultados a nivel profesional de este producto, y el apoyo prestado por el fabricante para la formación de programadores, Visual Basic se ha convertido en la primera herramienta de desarrollo de aplicaciones en entorno Windows.

Es obligado decir sin embargo, que sigue siendo BASIC. No se pueden comparar sus prestaciones con otros lenguajes cuando se desea llegar al fondo de la máquina y controlar uno a uno sus registros. No es ese el fin perseguido con VB y si es necesario llegar a esas precisiones será necesario utilizar otro lenguaje que permita bajar el nivel de programación. (Visual-C). o realizar librerías (DLLs) que lo hagan. En la mayor parte de las aplicaciones, las herramientas aportadas por VB son mas que suficiente para lograr un programa fácil de realizar y de altas prestaciones.

5.1.2 Visual Basic y el entorno Windows

Las interfaces gráficas de usuario (GUI) han revolucionado el mundo informático. Quizá más importante a largo plazo es el hecho de que las aplicaciones Windows **tienen una interfaz de usuario consistente**, es decir, los usuarios tienen más tiempo para dominar la aplicación sin tener que preocuparse de las teclas que deben pulsarse dentro de los menús y cuadros de diálogo.

Todo esto tiene un precio; antes de la existencia de Visual Basic, el desarrollo de aplicaciones para Windows era mucho más complicado que desarrollar aplicaciones para DOS. Los programadores tenían que preocuparse más de lo que estaba haciendo el ratón, de dónde estaba el usuario dentro de un menú y de si estaba realizando un clic o un doble clic en un punto determinado. Desarrollar aplicaciones para Windows requería expertos programadores en C, e incluso éstos tenían problemas.

Visual Basic ha cambiado esta situación. Se pueden desarrollar aplicaciones para Windows más rápidamente. Los errores de programación no se generan tan frecuentemente y, si lo hacen, son más sencillos de depurar. Además incluye dos conceptos importantes:

- Un método visual de creación de aplicaciones, incluyendo formularios (ventanas), controles y, componentes del formulario.
- La habilidad de asociar código directamente a cada evento de cada elemento del diseño visual.

Windows ha sido diseñado para tomar las ventajas de las nuevas tecnologías de hardware y software. Entre las innovaciones más importantes de la programación con VB se tiene:

- API (*Application Programming Interface*) de 32 bit, compatible con la API soportada en Windows NT.
- Un modelo de memoria sin muchas restricciones, pudiendo alcanzar casi los 2 billones de caracteres de longitud.
- La capacidad de incrustar objetos OLE entre la inmensa mayoría de aplicaciones Windows con solo arrastrar y soltar.
- La adaptación de los controles OLE (OCXs) en lugar de los controles VBX para utilizarlos en programación.
- Optimización del registro central para almacenar información de las aplicaciones.
- Optimización de la implementación de multimedia, incluyendo sonidos, gráficos y animaciones.
- Modo de presentación de las aplicaciones, incluyendo la barra de tareas, controles de ficha y, ficheros de ayuda.
- Etc.

5.1.3 Características generales de Visual Basic

Visual Basic es una herramienta de diseño de aplicaciones para Windows, en la que estas se desarrollan en una gran parte a partir del diseño de una interface gráfica. En una aplicación Visual Basic, el programa está formado por una parte de código puro, y otras partes asociadas a los objetos que forman la interface gráfica.

Es por tanto un termino medio entre la programación tradicional, formada por una sucesión lineal de código estructurado, y la programación orientada a objetos. Combina ambas tendencias. Ya que no se puede decir que VB pertenezca por completo a uno de esos dos tipos de programación, se debe inventar una palabra que la defina : PROGRAMACION VISUAL.

La creación de un programa bajo Visual Basic lleva los siguientes pasos:

- Creación de un interface de usuario. Este interface será la principal vía de comunicación hombre máquina, tanto para salida de datos como para entrada. Será necesario partir de una ventana (Formulario) a la que se le irán añadiendo los controles necesarios.

- Definición de las propiedades de los controles (Objetos) que se hayan colocado en ese formulario. Estas propiedades determinarán la forma estática de los controles, es decir, como son los controles y para qué sirven.
- Generación del código asociado a los eventos que ocurran a estos objetos. A la respuesta a estos eventos (click, doble click, una tecla pulsada, etc.) se le llama Procedimiento, y deberá generarse de acuerdo a las necesidades del programa.
- Generación del código del programa. Un programa puede hacerse solamente con la programación de los distintos procedimientos que acompañan a cada objeto. Sin embargo, VB ofrece la posibilidad de establecer un código de programa separado de estos eventos. Este código puede introducirse en unos bloques llamados Módulos, en otros bloques llamados Funciones, y otros llamados Procedimientos. Estos Procedimientos no responden a un evento acaecido a un objeto, sino que responden a un evento producido durante la ejecución del programa.

5.1.4 La programación visual y guiada por eventos

Un programa realizado en DOS es un conjunto de sentencias que se ejecutan de arriba a abajo más o menos, en el orden que el programador ha diseñado. Una aplicación en Windows presenta todas las opciones posibles en uno o más formularios para que el usuario elija entre ellas. La secuencia en la que se ejecutarán las sentencias no puede ser prevista por el programador. Esto da lugar a la *Programación Orientada a Eventos*. Para programar una aplicación en VB hay que escribir código separado para cada objeto en general, quedando la aplicación dividida en pequeños procedimientos, conducido cada uno de ellos por un suceso. Un ***suceso*** es una acción reconocida por un objeto (formulario o control) el suceso puede ser causado por el usuario o, indirectamente por el código. En Visual Basic cada formulario y cada control tienen predefinidos un conjunto de sucesos. Cuando ocurren estos sucesos, Visual Basic invoca al procedimiento asociado con el objeto para ese suceso.

5.1.5 El entorno Visual Basic

Los elementos que componen la pantalla de Visual Basic son:

Barra de título

Muestra el nombre del proyecto y del formulario que se encuentra activo.

Barra de menús

Agrupa los menús desplegables que contienen todas las operaciones que pueden llevarse a cabo en VB.

Barra de herramientas

Contiene los botones que se utilizan con mayor frecuencia al trabajar con un proyecto. Cambia a medida que se le utiliza. El programa cuenta con un total de cuatro barras de herramientas:

- Depuración (Debug). Esta barra de herramientas aparece cuando se utiliza las herramientas de depuración interactivas para rastrear y corregir problemas.
- Edición (Edit). Esta barra de herramientas auxilia en la edición del código de VB.
- Editor de formularios. (Form Editor). Esta barra de herramientas ayuda a ajustar objetos en el formulario.
- Estándar (Standard). Esta es la barra de herramientas predeterminada, que aparece debajo de la barra de menús.

Caja de herramientas

Provee de un conjunto de herramientas que permiten colocar los controles en el formulario durante el diseño del proyecto.

Ventana de proyecto

En esta ventana están especificados los ficheros (formularios, módulos, etc.) que forman la aplicación y, dónde se seleccionarán para crearlos o modificarlos. Esto se debe a que hay ficheros que pueden utilizarse en más de una aplicación. Además contiene dos botones: *Ver Formulario* que visualiza el formulario seleccionado y; *Ver Código* que visualiza el código del fichero seleccionado.

Ventana del formulario

Es la ventana que da lugar a la interfaz de usuario. Es la ventana que se personalizará. Los puntos que aparecen sobre el formulario, forman una rejilla que ayuda a la hora de alinear los controles que se sitúan sobre el mismo. Esta rejilla desaparece en tiempo de ejecución. Para eliminarla en tiempo de diseño se accederá a la opción *Herramientas/Opciones/Ficha Entorno/Mostrar Cuadrícula.*

Ventana de propiedades

Especifica las propiedades de cada uno de los objetos. En cada momento mostrará las propiedades del objeto seleccionado en el formulario. Está formada por dos partes: la lista desplegable de objetos que visualiza el nombre del objeto seleccionado y, la lista de propiedades del objeto seleccionado.

A continuación se presenta la pantalla de Visual Basic (Figura 3-1).

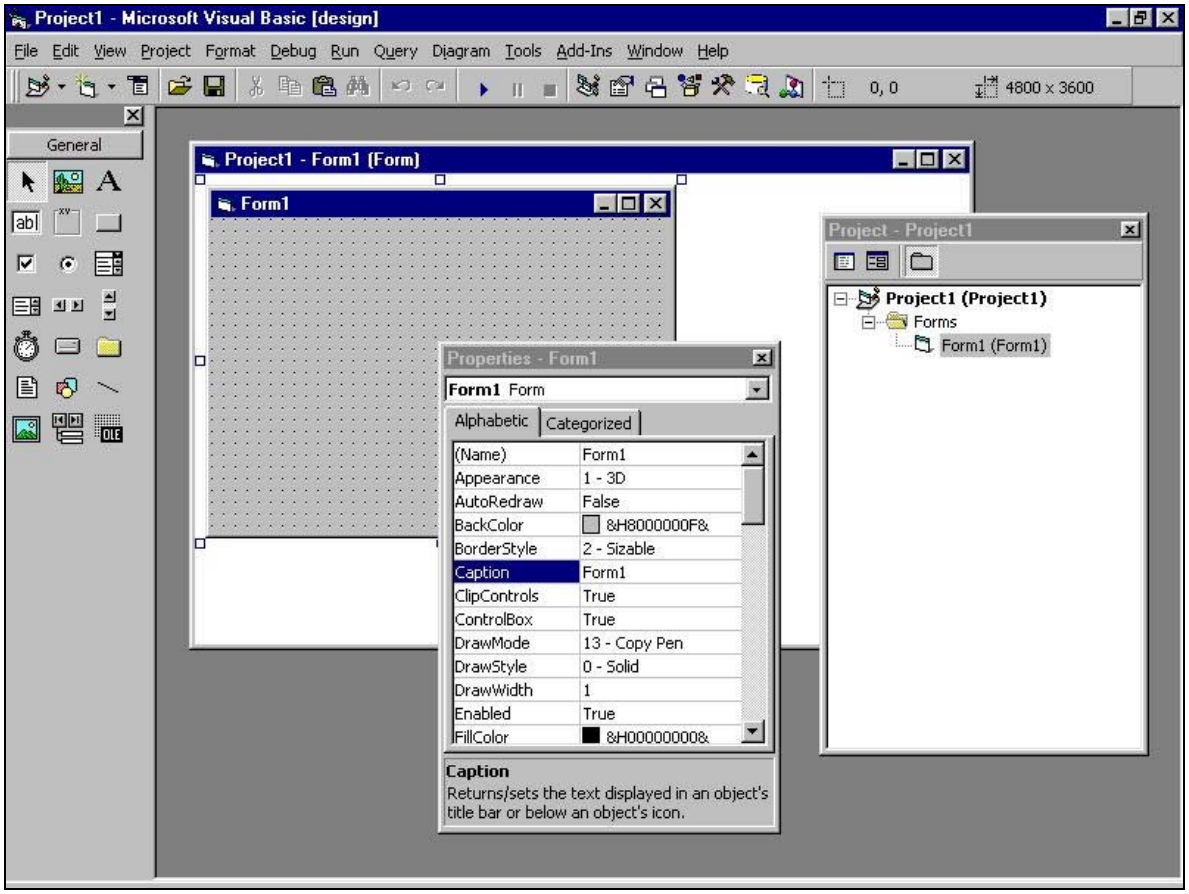


Figura 3-1
Pantalla de Visual Basic

¿Cómo funciona una aplicación Visual Basic?

Normalmente, se escribe código a los eventos a los que se quiera dar respuesta. Si el evento no tiene respuesta o no se produce, Visual Basic no hará nada, es decir no se ejecutará código alguno. Para poder ejecutar código, se tendrá que haber dado respuesta a algún evento de algún control y, que dicho evento tenga lugar.

¿Está Visual Basic realmente orientado a objetos?

Desde VB 4.0 se puede decir que este está realmente orientado a objetos, ya que cumple con los tres conceptos clave:

- **Encapsulación.** La utiliza de manera diferente a C++. Los objetos Visual Basic tienen propiedades como color, tamaño, tipo de letra y, además incorporan unos métodos que responden a mensaje como clics de ratón, pulsación de teclas, etc. En los que se puede integrar código.
- **Polimorfismo.** Capacidad de reaccionar de manera distinta ante un mensaje idéntico, dependiendo del objeto que lo reciba, por ejemplo un clic del ratón recibido por un botón de comando o por una caja de texto.

- **Herencia.** Capacidad de derivarse a partir de otro objeto.

5.1.6 Organización de aplicaciones en VB

Dado que es muy común en aplicaciones Visual Basic compartir código o formularios personalizados, Visual Basic organiza las aplicaciones en lo que denomina ***proyectos***. Cada proyecto puede tener varios formularios y, el código que activa los controles de un formulario es archivado con el formulario en archivos separados. El código general compartido por todos los formularios de una aplicación puede ser dividido en varios módulos, que también se archivan separadamente. En VB un proyecto puede tener, además, módulos de clase y ficheros de recursos.

Aunque Visual Basic almacena separadamente los archivos que forman un proyecto, hace un seguimiento de dónde están los archivos. Crea un archivo con la extensión .VBP de Visual Basic Program/Project. Visual Basic permite tener un solo proyecto abierto en un momento determinado.

Los ***formularios*** se archivan con la extensión .FRM y contienen una imagen del formulario y, de todos los controles que pertenecen a él, incluidas sus propiedades. También pueden contener subrutinas de manejo de eventos, procedimientos generales, declaraciones de variables y de constantes a nivel de formulario y, procedimientos externos.

Un ***módulo estándar*** contiene código Visual Basic que no está asociado a ningún formulario en particular. Los procedimientos que se encuentran en el módulo, pueden ser accedidos desde cualquier otro procedimiento de la aplicación. Se almacenan con la extensión .BAS. Los ***módulos de clase*** tienen la extensión .CLS y contienen código, incluido subrutinas, funciones, métodos y procedimientos para crear descripciones genéricas de objetos. Estos módulos contienen propiedades que describen el comportamiento de una clase, así como el código que define las propiedades y los métodos de la clase.

Los ***archivos de recursos*** se almacenan con la extensión .RES y contienen bitmaps, cadenas de texto, o cualquier otra información que pueda ser cambiada sin tener que reeditar el código de la aplicación. Un proyecto solo puede contener un archivo de recursos.

5.1.7 Procedimientos y Funciones

Como ya se ha mencionado, en Visual Basic el código es almacenado en ***módulos***. Hay tres clases de módulos: formularios (**.frm*), módulos estándar (**.bas*) y clases (**.cls*). Cada módulo puede contener:

- Declaraciones de constantes, de tipos, de variables y de procedimientos externos (DLL´s).
- Procedimientos conducidos por eventos.
- Procedimientos y funciones generales (estándar).
- Procedimientos ***Property***.

La base de una aplicación en Visual Basic la forman sus procedimientos conducidos por eventos. Un *procedimiento conducido por un evento* es el código que es invocado cuando un objeto reconoce que ha ocurrido un determinado evento.

Un *procedimiento estándar* es el código que se ejecuta de forma independiente a cualquier evento, y por lo tanto, también a cualquier objeto de la aplicación (formulario, control, etc.). Es invocado cuando se hace una llamada explícita del mismo.

Un procedimiento estándar puede escribirse como procedimiento ***Sub*** o como función ***Function***. Un procedimiento conducido por un evento siempre es un procedimiento ***Sub***.

Un procedimiento ***Property*** permite crear propiedades para una clase y manipularlas. Los módulos de clases son la base de la programación orientada a objetos en Visual Basic. Los objetos de estas clases pueden incluir sus propias propiedades y métodos, pero no sus propios eventos.

5.1.7.1 Ámbito de un procedimiento

Cuando un procedimiento no se califica explícitamente con las palabras reservadas ***Public*** o ***Private*** es, por omisión, *público* en todos los módulos. Lógicamente el carácter *público* de un procedimiento puede ser cambiado a *privado*, modificando así su accesibilidad. Por ejemplo, el siguiente procedimiento escrito en cualquier módulo es público:

```
Sub MiProc()
    ´Declaraciones y sentencias
End Sub
```

Consecuentemente, un procedimiento público (***Public***) puede invocarse desde cualquier parte de la aplicación, pero un procedimiento privado (***Private***) sólo puede ser llamado desde otros procedimientos que estén en el mismo módulo.

Cuando un procedimiento es llamado para su ejecución, Visual Basic busca ese procedimiento en el módulo activo. Si no lo encuentra, entonces continúa la búsqueda en el resto de los módulos de la aplicación.

5.1.7.2 Crear un procedimiento

Para crear un procedimiento general, hay que dirigirse a la ventana de proyecto, elegir el módulo donde se quiere definir el procedimiento y abrir la ventana de código correspondiente haciendo clic en el botón *Ver código*. A continuación se ejecuta la orden *Agregar procedimiento...* del menú *Herramientas*. También, puede escribirse ***Sub*** o ***Function*** seguido del nombre del procedimiento directamente sobre la ventana de código; en este último caso, al pulsar la tecla *Entrar* Visual Basic completará el esqueleto del procedimiento.

Para editar un procedimiento general existente, se selecciona "(*General*)" en la lista *objeto* de la ventana de código y a continuación se selecciona el procedimiento en la lista *procedimiento*.

No se crean procedimientos conducidos por eventos (ya existen). Para editar un procedimiento conducido por un evento, se selecciona el objeto (formulario, control, etc. en el cual se dispara el evento que conduce el procedimiento) en la lista *objeto* de la ventana de código, y a continuación se selecciona el procedimiento en la lista *procedimiento* (el nombre del procedimiento que aparece en la lista *procedimiento* es el mismo que el del evento que conduce al procedimiento).

5.1.7.3 Funciones (Function)

Una función es un procedimiento que cuando se ejecuta devuelve un resultado. La sintaxis correspondiente a una función es la siguiente:

[***Private/Public***] [***Static***] ***Function*** *nombre* [(*parámetros*)] [***As*** *tipo*]
[*sentencias*]
[*nombre = expresión*]
[***Exit Function***]
[*sentencias*]
[*nombre = expresión*]
End Function

- *nombre.* Es el nombre de la función; su tipo determina el tipo de datos que devuelve la función.
- *parámetros.* Son una lista de variables separadas por comas que se corresponden con los argumentos que son pasados cuando es invocada la función. Cuando se llama a una función, Visual Basic asigna el valor de cada argumento en la llamada al parámetro que ocupa su misma posición en la lista de parámetros de la función.
- *sentencias.* Son líneas de texto que indican una o más operaciones a realizar.
- *expresión.* Define el valor devuelto por la función. Este valor es almacenado en el propio *nombre* de la función, que actúa como variable dentro del cuerpo de la misma. Si no se efectúa esta asignación el resultado devuelto será 0 si ésta es numérica, o nulo ("") si la función es de caracteres.
- ***Exit Function.*** Permite salir de una función. ***Exit Function*** no es necesaria a no ser que se necesite retornar a la sentencia situada inmediatamente a continuación de la que efectúo la llamada antes de que la función finalice.
- ***End Function.*** Esta sentencia, al igual que ***Exit Function***, devuelve el control a la sentencia que se encuentra inmediatamente a continuación de la efectuó la llamada, continuando de esta forma la ejecución del programa.

La llamada a una función es de la forma:

[*variable =*] *nombre* [(*argumentos*)]

- *argumentos*. Son una lista de constantes, variables o expresiones separadas por comas. El número de argumentos debe ser igual al número de parámetros de la función. Los tipos de argumentos deben coincidir con los tipos de sus correspondientes parámetros.

Por compatibilidad con otros lenguajes, es aconsejable escribir los paréntesis cada vez que se llama a una función, aunque ésta no tenga argumentos. Una llamada a una función puede formar parte de una expresión.

5.1.7.4 Procedimientos (Sub)

La sintaxis que define un procedimiento es la siguiente:

[***Private/Public***] [***Static***] ***Sub*** *nombre* [(*parámetros*)]
  [*sentencias*]
  [***Exit Sub***]
  [*sentencias*]
End Sub

La explicación es análoga a la dada para las funciones (***Function***).

La llamada a un procedimiento puede ser de alguna de las dos formas siguientes:

Call *nombre* [(*argumentos*)]
nombre [*argumentos*]

A diferencia de una función, un procedimiento no puede ser utilizado en una expresión, ya que un procedimiento no retorna un valor a través de su nombre.

Para declarar procedimientos externos, esto es, procedimientos contenidos en una biblioteca dinámica (*Dynamic Link Library*, abreviadamente *DLL*), se utiliza la sentencia ***Declare***. Una sentencia ***Declare*** debe aparecer en la sección de declaraciones del módulo.

5.1.7.5 Llamar a procedimientos en otros módulos

Para llamar a un procedimiento público (***Sub*** o ***Function***) de un formulario o de una clase desde cualquier otro módulo hay que utilizar la siguiente sintaxis:

form.procedimiento(*args*)

donde *form* representa el formulario al cual pertenece el procedimiento llamado.

Para llamar a un procedimiento público de un módulo estándar desde cualquier otro módulo hay que hacerlo de la manera siguiente:

módulo.procedimiento(*args*)

donde *módulo* se refiere al nombre del módulo estándar al que pertenece el procedimiento.

5.1.7.6 Declarar un procedimiento privado

Para hacer que un procedimiento (***Sub*** o ***Function***) sólo sea accesible desde los procedimientos del módulo al cual pertenece, hay que colocar al principio de la cabecera del procedimiento la palabra clave ***Private***.

Si no se especifica la palabra clave ***Private*** se supone que el procedimiento es ***Public***, lo que significa que puede ser invocado desde otros módulos.

5.1.7.7 Argumentos por referencia y por valor

En los procedimientos (***Sub*** o ***Function***), los argumentos se pasan por *referencia*; de este modo, cualquier cambio de valor que sufra un parámetro en el cuerpo del procedimiento, también se produce en el argumento correspondiente de la llamada al procedimiento.

Cuando se llama a un procedimiento (***Sub*** o ***Function***), se puede especificar que el valor de un argumento no sea cambiado por ese procedimiento, poniendo dicho argumento entre paréntesis en la llamada. Un argumento entre paréntesis en la llamada es un argumento pasado por *valor*. Por ejemplo:

Factorial (*Num*), *Fact*

Observando la llamada al procedimiento *Factorial*; el argumento *Num* es pasado por valor, lo cual significa que se pasa una copia de *Num*. Si el procedimiento cambia ese valor, el cambio afecta sólo a la copia y no a la propia variable *Num*.

Otra forma de especificar que un argumento será pasado por valor es anteponiendo la palabra ***ByVal*** a la declaración del parámetro en la cabecera del procedimiento (***Sub*** o ***Function***). Análogamente, ***ByRef*** especifica que el parámetro será pasado por referencia; por omisión se supone ***ByRef***.

Por ejemplo:

Sub Factorial (*ByVal N, F*)

5.2 Lenguaje C

5.2.1 Antecedentes históricos

C es un lenguaje de programación de propósito general que ofrece economía sintáctica, control de flujo y estructuras sencillas y un buen conjunto de operadores. No es un lenguaje de muy alto nivel y más bien un lenguaje pequeño, sencillo y no está especializado en ningún tipo de aplicación. Esto lo hace un lenguaje potente, con un campo de aplicación ilimitado y sobre todo, se aprende rápidamente. En poco tiempo, un programador puede utilizar la totalidad del lenguaje.

Este lenguaje ha sido estrechamente ligado al sistema operativo UNIX, puesto que fueron desarrollados conjuntamente. Sin embargo, este lenguaje no está ligado a ningún sistema operativo ni a ninguna máquina concreta. Se le suele llamar *lenguaje de programación de sistemas* debido a su utilidad para escribir compiladores y sistemas operativos, aunque de igual forma se pueden desarrollar cualquier tipo de aplicación.

La base del C proviene del BCPL, escrito por Martin Richards, y del B escrito por Ken Thompson en 1970 para el primer sistema UNIX en un DEC PDP-7. Estos son lenguajes sin tipos, al contrario que el C que proporciona varios tipos de datos. Los tipos son caracteres, números enteros y en coma flotante, de varios tamaños. Además se pueden crear tipos derivados mediante la utilización de punteros, vectores, registros y uniones. El primer compilador de C fue escrito por Dennis Ritchie para un DEC PDP-11 y escribió el propio sistema operativo en C.

C trabaja con tipos de datos que son directamente tratables por el hardware de la mayoría de computadoras actuales, como son los caracteres, números y direcciones. Estos tipos de datos pueden ser manipulados por las operaciones aritméticas que proporcionan las computadoras. No proporciona mecanismos para tratar tipos de datos que no sean los básicos, debiendo ser el programador el que los desarrolle. Esto permite que el código generado sea muy eficiente y de ahí el éxito que ha tenido como lenguaje de desarrollo de sistemas. No proporciona otros mecanismos de almacenamiento de datos que no sea el estático y no proporciona mecanismos de entrada ni salida. Ello permite que el lenguaje sea reducido y los compiladores de fácil implementación en distintos sistemas. Por contra, estas carencias se compensan mediante la inclusión de funciones de librería para realizar todas estas tareas, que normalmente dependen del sistema operativo.

Originalmente, el manual de referencia del lenguaje para el gran público fue el libro [1] de Kernighan y Ritchie, escrito en 1977. Es un libro que explica y justifica totalmente el desarrollo de aplicaciones en C, aunque en él se utilizaban construcciones, en la definición de funciones, que podían provocar confusión y errores de programación que no eran detectados por el compilador. Como los tiempos cambian y las necesidades también, en 1983 ANSI establece el comité X3J11 para que desarrolle una definición moderna y comprensible del C. El estándar está basado en el manual de referencia original de 1972 y se desarrolla con el mismo espíritu de sus creadores originales. La primera versión de estándar se publicó en 1988 y actualmente todos los compiladores utilizan la nueva definición. Una aportación muy importante de ANSI consiste en la definición de un conjunto de librerías que acompañan al compilador y de las funciones contenidas en ellas. Muchas de las operaciones comunes con el sistema operativo se realizan a través de estas funciones. Una colección de ficheros de encabezamiento, *headers*, en los que se definen los tipos de datos y funciones incluidas en cada librería. Los programas que utilizan estas bibliotecas para interactuar con el sistema operativo obtendrán un comportamiento equivalente en otro sistema.

¿Que es el C++?

El C++ es en realidad una mejora del propio C, en cuanto velocidad en su código final, así como en las funciones, datos, y gestión de memoria y recursos del propio ordenador.

Aunque en sus instrucciones, puede haber cierta diferencia y además la aparición de comandos nuevos con respecto al C, no es otro lenguaje distinto del C, sino un descendiente directo y mejorado del propio C.

5.2.2 El desarrollo de un programa

Un programa C puede estar formado por diferentes módulos o fuentes. Es conveniente mantener los fuentes de un tamaño no muy grande, para que la compilación sea rápida. También, al dividirse un programa en partes, puede facilitar la legibilidad del programa y su estructuración. Los diferentes fuentes son compilados de forma separada, únicamente los fuentes que han sido modificados desde la última compilación, y después combinados con las librerías necesarias para formar el programa en su versión ejecutable.

Los comandos necesarios para compilar, *linkar* y ejecutar un programa dependen del sistema operativo y se debe dirigir a los manuales correspondientes para conocer la sintaxis exacta.

Todos los programas que se hagan en C contendrán unos de los siguientes elementos:

- Contendrá conexiones con ficheros.
- Contendrá conexiones con el teclado, para entrar datos.
- Contendrá operaciones gráficas o relacionadas con la pantalla.
- Bucles.
- Rutinas.
- Datos ya preparados e introducidos por el propio programador.

Uno de los programas más sencillos que se puede desarrollar en C es:

```
#include <stdio.h>

main()

{

        printf("Hola amigos!\n");

}
```

Con el se visualiza el mensaje Hola amigos! en la terminal. En la primera línea se indica que se tengan en cuenta las funciones y tipos definidos en la librería stdio (*standard input/output*). Estas definiciones se encuentran en el fichero *header* stdio.h. Ahora, en la función main se incluye una única sentencia que llama a la función printf. Esta toma como argumento una cadena de caracteres que se imprimen y van encerrados entre dobles comillas " ". El símbolo \n indica un cambio de línea. Hay un grupo de símbolos, que son tratados como caracteres individuales, que especifican algunos caracteres especiales del código ASCII. Los más importantes son:

\a	*Alert*
\b	*backspace*
\f	*formfeed*
\n	*newline*
\r	*carriage return*
\t	*horizontal tab*
\v	*vertical tab*
\\	*backslash*
\'	*single quote*
\"	*double quote*
\OOO	visualiza un carácter cuyo código ASCII es OOO en octal.
\xHHH	visualiza un carácter cuyo código ASCII es HHH en hexadecimal.

5.2.3 Los datos en C

Los tipos de datos básicos definidos por C son caracteres, números enteros y números en coma flotante. Los caracteres son representados por char, los enteros por short, int, long y los números en coma flotante por float y double. Los tipos básicos disponibles y su tamaño son:

char	Carácter	(normalmente 8 bits)
short	Entero corto con signo	(normalmente 16 bits)
int	Entero con signo	(depende de la implementación)
unsigned	Entero sin signo	(depende de la implementación)
long	Entero largo con signo	(normalmente 32 bits)
float	Flotante simple	(normalmente 32 bits)
double	Flotante doble	(normalmente 64 bits)

La palabra unsigned en realidad es un modificador aplicable a tipos enteros, aunque si no se especifica un tipo se supone int. Un modificador es una palabra clave de C que indica que una variable, o función, no se comporta de la forma normal. Hay también un modificador signed, pero como los tipos son por defecto con signo, casi no se utiliza.

5.2.4 Funciones

Un programa C está formado por un conjunto de funciones que al menos contiene la función main. Una función se declara con el nombre de la función precedido del tipo de valor que retorna y una lista de argumentos encerrados entre paréntesis. El cuerpo de la

función está formado por un conjunto de declaraciones y de sentencias comprendidas entre llaves.

5.2.5 Expresiones y operadores

Los distintos operadores permiten formar expresiones tanto aritméticas como lógicas. Los operadores aritméticos y lógicos son:

+, -	suma, resta
++, --	incremento, decremento
*, /, %	multiplicación, división, módulo
>>, <<	rotación de bits a la derecha, izquierda.
&	AND *booleano*
|	OR *booleano*
^	EXOR *booleano*
~	complemento a 1
!	complemento a 2, NOT lógico
==, !=	igualdad, desigualdad
&&, ||	AND, OR lógico
<, <=	menor, menor o igual
>, >=	mayor, mayor o igual

En estos operadores deben tenerse en cuenta la precedencia de operadores y las reglas de asociatividad, que son las normales en la mayoría de lenguajes. Además hay toda una serie de operadores aritméticos con asignación, como pueden ser += y ^=.

En la evaluación de expresiones lógicas, los compiladores normalmente utilizan técnicas de evaluación rápida. Para decidir si una expresión lógica es cierta o falsa muchas veces no es necesario evaluarla completamente. Por ejemplo una expresión formada <exp1> || <exp2>, el compilador evalúa primero <exp1> y si es cierta, no evalúa <exp2>. Por ello se deben evitar construcciones en las que se modifiquen valores de datos en la propia expresión, pues su comportamiento puede depender de la implementación del compilador o de la optimización utilizada en una compilación o en otra. Estos son errores que se pueden cometer fácilmente en C ya que una asignación es también una expresión.

5.2.6 Conversión de tipos

Cuando se escribe una expresión aritmética a+b, en la cual hay variables o valores de distintos tipos, el compilador realiza determinadas conversiones antes de que evalúe la expresión. Estas conversiones pueden ser para 'aumentar' o 'disminuir' la precisión del tipo al que se convierten los elementos de la expresión. Un ejemplo claro, es la comparación de una variable de tipo int con una variable de tipo double. En este caso, la de tipo int es convertida a double para poder realizar la comparación.

Los tipos pequeños son convertidos de la forma siguiente: un tipo char se convierte a int, con el modificador signed si los caracteres son con signo, o unsigned si los caracteres son sin signo. Un unsigned char es convertido a int con los bits más altos puestos a cero. Un signed char es convertido a int con los bits más altos puestos a uno o cero, dependiendo del valor de la variable.

Para los tipos de mayor tamaño: si un operando es de tipo double, el otro es convertido a double. Si un operando es de tipo float, el otro es convertido a float. Si un operando es de tipo unsigned long, el otro es convertido a unsigned long. Si un operando es de tipo long, el otro es convertido a long. Si un operando es de tipo unsigned, el otro es convertido a unsigned. Si no, los operandos son te tipo int.

5.2.7 Control de flujo

La sentencia de control básica es if (<e>) then <s> else <t>. En ella se evalúa una expresión condicional y si se cumple, se ejecuta la sentencia *s*; si no, se ejecuta la sentencia *t*. La segunda parte de la condición, else <t>, es opcional.

En el caso que <e> no sea una expresión condicional y sea aritmética, se considera falso si vale 0; y si no, verdadero. Hay casos en los que se deben evaluar múltiples condiciones. Se puede programar con un grupo de sentencias if then else anidadas, aunque ello puede ser de complicada lectura. Para evitarlo se puede utilizar la sentencia switch. Cuando se encuentra una sentencia case que concuerda con el valor del switch se ejecutan las sentencias que le siguen y todas las demás a partir de ahí, a no ser que se introduzca una sentencia break para salir de la sentencia switch.

Otras sentencias de control de flujo son las que permiten realizar iteraciones sobre un conjunto de sentencias. En C se tienen tres formas principales de realizar iteraciones. La sentencia while (<e>) <s> es seguramente la más utilizada. La sentencia, o grupo de sentencias <s> se ejecuta mientras la evaluación de la expresión <e> sea verdadera. Otra sentencia iterativa, que permite inicializar los controles del bucle es la sentencia for (<i>; <e>; <p>) <s>.

Una variación de la sentencia while es: do <s> while (<e>); En ella la sentencia se ejecuta al menos una vez, antes de que se evalúe la expresión condicional.

Otras sentencias interesantes, aunque menos utilizadas son break y continue. La sentencia break provoca que se termine la ejecución de una iteración o para salir de la sentencia switch.

5.2.8 Definiciones y prototipos de funciones

Los programas sencillos normalmente no necesitan un nivel de estructuración elevado. Pero cuando éstos crecen un poco se necesita estructurarlos adecuadamente para mantenerlos legibles, facilitar su mantenimiento y para poder reutilizar ciertas porciones de código. El mecanismo C que permite esto son las funciones. Con los compiladores, los fabricantes

proporcionan un conjunto importante de funciones de librería. A veces, puede interesar el construir nuevas librerías. Las funciones se definen de la siguiente manera:

Los prototipos de funciones son una característica clave de la recomendación ANSI del C.

Un prototipo es una declaración que toma la forma:

tipo_resultado nombre_función (*tipo_parámetro nombre_parámetro ...*);

Aquí hay varios ejemplos:

- int fact_i (int v);
- int mayor (int a, int b);
- int cero (double a);
- long raiz (long valor);
- void final_countdown (void);
- int main (int argc, char **argv);

Observando el prototipo de una función se puede decir exactamente que tipo de parámetros necesita y que resultado devuelve. Si una función tiene como argumento void, quiere decir que no tiene argumentos, al igual que si el resultado es void, no devuelve ningún valor.

Las funciones al viejo estilo se compilan correctamente en muchos compiladores actuales. Por contra, proporcionan menos información sobre sus parámetros y errores que afecten al tipo de parámetros de llamada a las funciones no pueden ser detectados automáticamente. Por tanto, la declaración de una función debe escribirse igual que su prototipo pero sin el punto y coma final. El cuerpo de la función le sigue encerrado entre llaves.

En un programa que esté formado por distintas partes bien diferenciadas es conveniente utilizar múltiples ficheros fuente. Cada fuente agrupa las funciones semejantes, como por ejemplo en un compilador se podría tener un fuente para el análisis léxico, otro para el sintáctico y otro para la generación de código. Pero en un fuente necesitaremos funciones que se han definido en otro. Para ello, se escribirá un fichero de cabecera (*header*), que contendrá las declaraciones que se pueden necesitar en otros fuente. Así, en el fuente que implementa el analizador sintáctico se pondrá una línea #include "lexic.h". De esta forma al compilar el módulo sintáctico se tendrán todos los prototipos de las funciones del léxico y el compilador podrá detectar malas utilizaciones de las funciones allí definidas.

5.2.9 Construcción de tipos

Los datos del mundo real, normalmente no están formados por variables escalares que se encuentren dentro de los tipos básicos. Por ejemplo, puede interesar cuántos módulos en C se han escrito cada semana, a lo largo del año. O también, tener los datos de cada planeta

del Sistema Solar, masa, posición, velocidad y aceleración, para un programa de simulación de la ley de gravitación de Newton. Para resolver el primer caso, C permite declarar una variable que sea de tipo vector. Para el segundo, se puede definir un registro para cada elemento.

Un vector es una porción de memoria que es utilizada para almacenar un grupo de elementos del mismo tipo Un vector se declara: *tipo nombre* [*tamaño*];. Por ejemplo, int modulo[52];. Aquí 'modulo' es un vector de 52 elementos enteros.

Cada elemento de un vector es accedido mediante un número de índice y se comporta como una variable del tipo base del vector. Los elementos de un vector son accedidos por índices que van desde 0 hasta N-1 para un vector de N elementos. Los elementos de un vector pueden ser inicializados en la misma declaración.

También se pueden definir vectores multidimensionales. C no impone ninguna limitación al número de dimensiones de un vector. Existe, en cambio, la limitación del tamaño de memoria que se pueda utilizar en el ordenador.

5.2.10 Ámbito de funciones y variables

El ámbito, o visibilidad, de una variable indica en que lugares del programa está activa esa variable. Hasta ahora, se han utilizado variables definidas en el cuerpo de funciones. Estas variables se crean en la memoria del ordenador cuando se llama a la función y se destruyen cuando se sale. Es necesario a veces, que una variable tenga un valor que pueda ser accesible desde todas las funciones de un mismo fuente, e incluso desde otros fuentes.

En C, el ámbito de las variables depende de dónde han sido declaradas y si se les ha aplicado algún modificador. Una variable definida en una función es, por defecto, una variable local. Esto es, que sólo existe y puede ser accedida dentro de la función. Para que una variable sea visible desde una función cualquiera del mismo fuente debe declararse fuera de cualquier función. Esta variable sólo será visible en las funciones definidas después de su declaración. Por esto, el lugar más común para definir las variables globales es antes de la definición de ninguna función. Por defecto, una variable global es visible desde otro fuente. Para definir que existe una variable global que está definida en otro fuente tenemos que anteponer la palabra extern a su declaración. Esta declaración únicamente indica al compilador que se hará referencia a una variable externa al módulo que se compila.

Las variables locales llevan implícito el modificador auto. Esto es que se crean al inicio de la ejecución de la función y se destruyen al final. En un programa sería muy ineficiente en términos de almacenamiento que se crearan todas las variables al inicio de la ejecución. Por contra, en algunos casos es deseable. Esto se consigue anteponiendo el modificador static a una variable local. Si una función necesita una variable que únicamente sea accedida por la misma función y que conserve su valor a través de sucesivas llamadas, es el caso adecuado para que sea declarada local a la función con el modificador static. El modificador static se puede aplicar también a variables globales. Una variable global es por defecto accesible desde cualquier fuente del programa. Si, por cualquier motivo, se desea

que una de estas variables no sea visible desde otro fuente se le debe aplicar el modificador static. Lo mismo ocurre con las funciones. Las funciones definidas en un fuente son utilizables desde cualquier otro. En este caso conviene incluir los prototipos de las funciones del otro fuente. Si no se desea que alguna función pueda ser llamada desde fuera del fuente en la que está definida se le debe anteponer el modificador static.

Otro modificador muy importante es const. Con él se pueden definir variables cuyo valor debe permanecer constante durante toda la ejecución del programa. También se puede utilizar con argumentos de funciones. En esta caso se indica que el argumento en cuestión es un parámetro y su valor no debe ser modificado. En el caso que por error se modifique ese parámetro, el compilador indicará el error.

Otro modificador utilizado algunas veces es el register. Este modificador es aplicable únicamente a variables locales e indica al compilador que esta debe ser almacenada permanentemente en un registro del procesador de la computadora. Este modificador es herencia de los viejos tiempos, cuando las tecnologías de optimización de código no estaban muy desarrolladas y se debía decir qué variable era muy utilizada en la función. Hoy en día casi todos los compiladores realizan un estudio de qué variables locales son las más adecuadas para ser almacenadas en registros, y las asignan automáticamente. Con los compiladores modernos se puede dar el caso de que una declaración register inadecuada disminuya la velocidad de ejecución de la función, en lugar de aumentarla. Por ello, hoy en día, la utilización de este modificador está en desuso, hasta el punto de que algunos compiladores lo ignoran. Se debe tener en cuenta que de una variable declarada como register no se puede obtener su dirección, ya que está almacenada en un registro y no en memoria.

5.2.11 Apuntadores

Cada variable de un programa tiene una dirección en la memoria del ordenador. Esta dirección indica la posición del primer byte que la variable ocupa. En el caso de una estructura es la suma del tamaño de cada uno de sus campos. Como en cualquier caso las variables son almacenadas ordenadamente y de una forma predecible, es posible acceder a estas y manipularlas mediante otra variables que contenga su dirección. A este tipo de variables se les denomina apuntadores.

Los apuntadores en C son el tipo más potente y seguramente la otra clave del éxito del lenguaje. La primera ventaja que se obtiene de los apuntadores es la posibilidad que dan de poder tratar con datos de un tamaño arbitrario sin tener que moverlos por la memoria. Esto puede ahorrar un tiempo de computación muy importante en algunos tipos de aplicaciones. También permiten que una función reciba y cambie el valor de una variable. Es conveniente recordar que todas las funciones C únicamente aceptan parámetros por valor. Mediante un puntero a una variable se puede modificarla indirectamente desde una función cualquiera.

Para manipular un apuntador, como variable que es, se utiliza su nombre; pero para acceder a la variable a la que apunta se le debe preceder de *. A este proceso se le llama indirección. Se accede indirectamente a una variable. Para trabajar con apuntadores existe

un operador, &, que indica 'dirección de'. Con él se puede asignar a un apuntador la dirección de una variable, o pasar como parámetro a una función.

Los apuntadores también se pueden utilizar con los registros. Para ello se utiliza la notación -> en lugar del punto que se utilizaba anteriormente.

5.2.12 Funciones matemáticas

La utilización de las funciones matemáticas definidas en el ANSI C requieren la inclusión del fichero math.h. Todas ellas trabajan con el tipo double, por lo que si los argumentos o resultados son del tipo float el propio compilador se encarga de convertirlos al formato adecuado. En ANSI se está trabajando para proporcionar funciones con argumentos de tipo float e introducir el tipo long float. Casi todas la funciones tienen la forma double *nombre* (double x);.

atan2	toma dos argumentos x e y y devuelve la arcotangente de y/x en radianes.
exp	devuelve el valor e elevado a x.
acos	retorna el arco coseno del parámetro x.
asin	retorna el arco seno del parámetro x.
atan	retorna el valor de la arco tangente del parámetro x.
cos	retorna el coseno del ángulo x.
cosh	retorna el coseno hiperbólico del parámetro x.
sin	retorna el seno del ángulo x.
sinh	retorna el seno hiperbólico del parámetro x.
tan	retorna la tangente del ángulo x.
tanh	retorna la tangente hiperbólica del parámetro x.
log	retorna el logaritmo natural del parámetro x.
log10	retorna el logaritmo en base 10 del parámetro x.
pow	toma dos parámetros *x* e *y* y devuelve el valor *xy*
sqrt	retorna la raiz cuadrada del parámetro x.

6 PERSPECTIVAS DE LOS LENGUAJES DE ALTO NIVEL

Hasta antes de 1969 el software era prácticamente exclusivo de la computadora para el cual estaba escrito, y los programas para una marca de computadora no podían correr en otra; sin embargo, a partir de ese año la IBM desarrolló el software portátil inaugurando la era de los productos de software (los programas pueden procesarse en cualquier máquina). Con este enfoque, los lenguajes se continúan desarrollando y se obtiene un software de aplicaciones cuyo objetivo es convertir a la computadora en una herramienta para todas las personas; aparecen procesadores de texto, las hojas de cálculo electrónico, bases de datos,

lenguajes de consulta y generadores de aplicaciones cuyo uso no requiere saber programar la computadora, lo único necesario es aprender a usar eficientemente las órdenes o comandos del software de aplicaciones que se seleccione, para realizar tareas diversas. Por último, la tendencia en el futuro, con la pronta aparición de sistemas de inteligencia artificial, es que se llegará a dialogar de forma natural con la computadora para la solución de múltiples problemas. Los lenguajes orientados hacia el desarrollo de los sistemas expertos y de la inteligencia artificial se denominan como cuarta generación.

CAPÍTULO 4

Sistemas operativos

Para empezar hay que definir lo que es un Sistema Operativo:

> *"Sistema operativo es un programa o conjunto de programas que permiten administrar los recursos de hardware y software de una computadora".*

1 UN POCO DE HISTORIA

A finales de los cuarentas el uso de computadoras estaba restringido a aquellas empresas o instituciones que podían pagar su alto precio, y no existían los sistemas operativos. En su lugar, el programador debía tener un conocimiento y contacto profundo con el hardware, y en el infortunado caso de que su programa fallara, debía examinar los valores de los registros y páneles de luces indicadoras del estado de la computadora para determinar la causa del fallo y poder corregir su programa, además de enfrentarse nuevamente a los procedimientos de apartar tiempo del sistema y poner a punto los compiladores, ligadores, etc; para volver a correr su programa, es decir, enfrentaba el problema del procesamiento serial (serial processing).

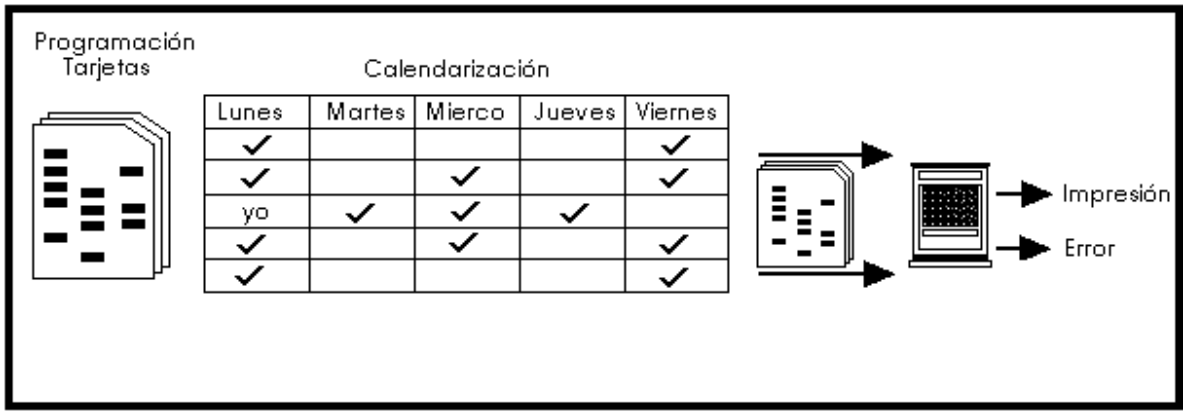


Figura 4-1
Ciclo de programación primera generación de computadoras

La importancia de los sistemas operativos nace históricamente desde los cincuentas, cuando se hizo evidente que el operar una computadora por medio de tableros enchufables en la primera generación y luego por medio del trabajo en lote en la segunda generación se podía mejorar notoriamente, pues el operador realizaba siempre una secuencia de pasos repetitivos, lo cual es la característica fundamental de un programa. Es decir, se comenzó a ver que las tareas mismas del operador podían plasmarse en un programa, el cual a través del tiempo y por su enorme complejidad se le llamó "Sistema Operativo". Así, se tiene entre los primeros sistemas operativos al Fortran Monitor System (FMS) e IBSYS.

Posteriormente, en la tercera generación de computadoras nace uno de los primeros sistemas operativos con la filosofía de administrar una familia de computadoras: el OS/360 de IBM. Fue este un proyecto tan novedoso y ambicioso que enfrentó por primera vez una serie de problemas conflictivos debido a que anteriormente las computadoras eran creadas para dos propósitos en general: el comercial y el científico. Así, al tratar de crear un solo sistema operativo para computadoras que podían dedicarse a un propósito, al otro o ambos, puso en evidencia la problemática del trabajo en equipos de análisis, diseño e implantación de sistemas grandes. El resultado fue un sistema del cual uno de sus mismos diseñadores patentizó su opinión en la portada de un libro: una horda de bestias prehistóricas atascadas en un foso de brea.

Surge también en la tercera generación de computadoras el concepto de la multiprogramación, porque debido al alto costo de las computadoras era necesario idear un esquema de trabajo que mantuviese a la unidad central de procesamiento más tiempo ocupada, así como el encolado (spooling) de trabajos para su lectura hacia los lugares libres de memoria o la escritura de resultados. Sin embargo, se puede afirmar que los sistemas durante la tercera generación siguieron siendo básicamente sistemas de lote.

En la cuarta generación la electrónica avanza hacia la integración a gran escala, pudiendo crear circuitos con miles de transistores en un centímetro cuadrado de silicón y ya es posible hablar de las computadoras personales y las estaciones de trabajo. Surgen los conceptos de interfaces amigables intentando así atraer al público en general al uso de las computadoras como herramientas cotidianas. Se hacen populares el MS-DOS y UNIX en estas máquinas. También es común encontrar clones de computadoras personales y una multitud de empresas pequeñas ensamblándolas por todo el mundo.

Para mediados de los ochentas, comienza el auge de las redes de computadoras y la necesidad de sistemas operativos en red y sistemas operativos distribuidos. La red mundial Internet se va haciendo accesible a toda clase de instituciones y se comienzan a dar muchas soluciones (y problemas) al querer hacer convivir recursos residentes en computadoras con sistemas operativos diferentes. Para los 90's el paradigma de la programación orientada a objetos cobra auge, así como el manejo de objetos desde los sistemas operativos. Las aplicaciones intentan crearse para ser ejecutadas en una plataforma específica y poder ver sus resultados en la pantalla o monitor de otra diferente (por ejemplo, ejecutar una simulación en una máquina con UNIX y ver los resultados en otra con DOS). Los niveles de interacción se van haciendo cada vez más profundos.

2 SISTEMAS OPERATIVOS PARA MICROCOMPUTADORAS

Las primeras microcomputadoras carecían de sistema operativo, por lo que su uso era complejo. Gary Kidall desarrolló el primer sistema operativo para microcomputadoras, el cual se denominó CP/M; su finalidad fue controlar la entrada/salida de información del recién diseñado microprocesador, así como el almacenamiento de datos y proceso general. A continuación Bill Gates y Paul Allen también desarrollaron este tipo de software para microcomputadoras cuando escribieron el sistema operativo más conocido hasta ese

entonces: el DOS para la IBM y su equivalente el MSDOS para las computadoras compatibles.

Actualmente toda las microcomputadoras están diseñadas para funcionar y correr los programas con ayuda de un sistema operativo, es decir, para utilizar algún software de aplicaciones, es requisito primero carga en la memoria primaria la parte del sistema operativo, cuya función fundamental es administrar los recursos de la computadora, dirigiendo el tránsito electrónico de la información que entra/sale, minimizando su tiempo de estancia; maximizando la productividad del proceso y optimizando los recursos del sistema de cómputo al trabajar en la obtención de algún resultado para un proceso o problema dado. En resumen, un sistema operativo está constituido por una serie de programas que controlan toda la actividad del sistema de cómputo y del software de aplicaciones que se utilice.

El sistema operativo está constituido por tres partes esencialmente:

- Un administrador de entrada/salida que coordina toda las comunicaciones de la computadora con sus equipos periféricos y por consiguiente el flujo de información entre ellos, por ejemplo de pantalla a impresora.
- El procesador de comandos que interpreta todo lo que se teclea, permitiendo con ello su ejecución.
- Por último, una serie de programas denominados utilerías que permite la administración de los archivos y tareas diversas como el formateo de discos, copiado, etc.

De esta manera , el sistema operativo indica a la computadora cómo recibir, almacenar, desplegar, recuperar o imprimir información o prepararla para su envío a lugares distantes. En forma complementaria ofrece la posibilidad de administra información. Dado que el sistema operativo toma parte de la memoria RAM, el resto de la memoria está destinada para los programas de aplicaciones y archivos de datos.

Al principio los sistemas operativos fueron exclusivos para ciertos modelos y marcas de computadoras; de igual manera gran cantidad del software que se escribía, sólo corría con el sistema operativo para el que fue escrito. Si el sistema operativo puede correr en computadoras diversas, se dice que es de tipo portátil. Actualmente la tendencia es crear este último tipo de software.

3 CLASIFICACIÓN DE LOS SISTEMAS OPERATIVOS

Básicamente se cubrirán tres clasificaciones: sistemas operativos por su estructura (visión interna), sistemas operativos por los servicios que ofrecen y, finalmente, sistemas operativos por la forma en que ofrecen sus servicios (visión externa).

3.1 Sistemas operativos por su estructura

Se deben observar dos tipos de requisitos cuando se construye un sistema operativo, los cuales son:

- Requisitos de usuario: Sistema fácil de usar y de aprender, seguro, rápido y adecuado al uso al que se le quiere destinar.
- Requisitos del software: Donde se engloban aspectos como el mantenimiento, forma de operación, restricciones de uso, eficiencia, tolerancia frente a los errores y flexibilidad.

A continuación se describen las distintas estructuras que presentan los actuales sistemas operativos para satisfacer las necesidades que de ellos se quieren obtener.

3.1.1 Estructura monolítica

Es la estructura de los primeros sistemas operativos constituidos fundamentalmente por un solo programa compuesto de un conjunto de rutinas entrelazadas de tal forma que cada una puede llamar a cualquier otra (Ver Fig. 4-2). Las características fundamentales de este tipo de estructura son:

- Construcción del programa final a base de módulos compilados separadamente que se unen a través del ligador.
- Buena definición de parámetros de enlace entre las distintas rutinas existentes, que puede provocar mucho acoplamiento.
- Carecen de protecciones y privilegios al entrar a rutinas que manejan diferentes aspectos de los recursos de la computadora, como memoria, disco, etc.

Generalmente están hechos a medida, por lo que son eficientes y rápidos en su ejecución y gestión, pero por lo mismo carecen de flexibilidad para soportar diferentes ambientes de trabajo o tipos de aplicaciones.

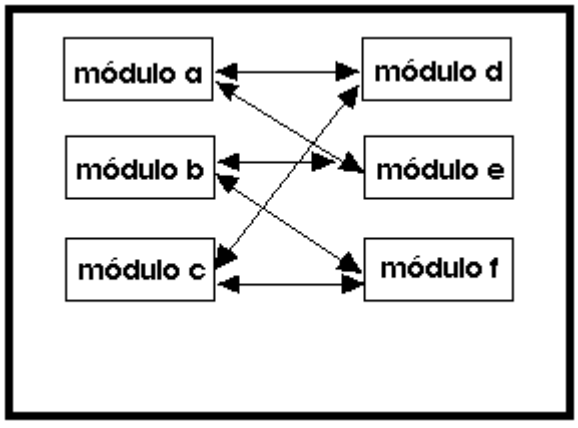


Figura 4-2
Estructura monolítica

3.1.2 Estructura jerárquica

A medida que fueron creciendo las necesidades de los usuarios y se perfeccionaron los sistemas, se hizo necesaria una mayor organización del software, del sistema operativo, donde una parte del sistema contenía subpartes y esto organizado en forma de niveles.

Se dividió el sistema operativo en pequeñas partes, de tal forma que cada una de ellas estuviera perfectamente definida y con un claro interface con el resto de elementos.

Se constituyó una estructura jerárquica o de niveles en los sistemas operativos, el primero de los cuales fue denominado THE (Technische Hogeschool, Eindhoven), de Dijkstra, que se utilizó con fines didácticos (Ver Fig. 4-3). Se puede pensar también en estos sistemas como si fueran `multicapa'. Multics y Unix caen en esa categoría.

Figura 4-3
Sistema jerárquico THE

En la estructura anterior se basan prácticamente la mayoría de los sistemas operativos actuales. Otra forma de ver este tipo de sistema es la denominada de anillos concéntricos o "rings" (Ver Fig. 4-4).

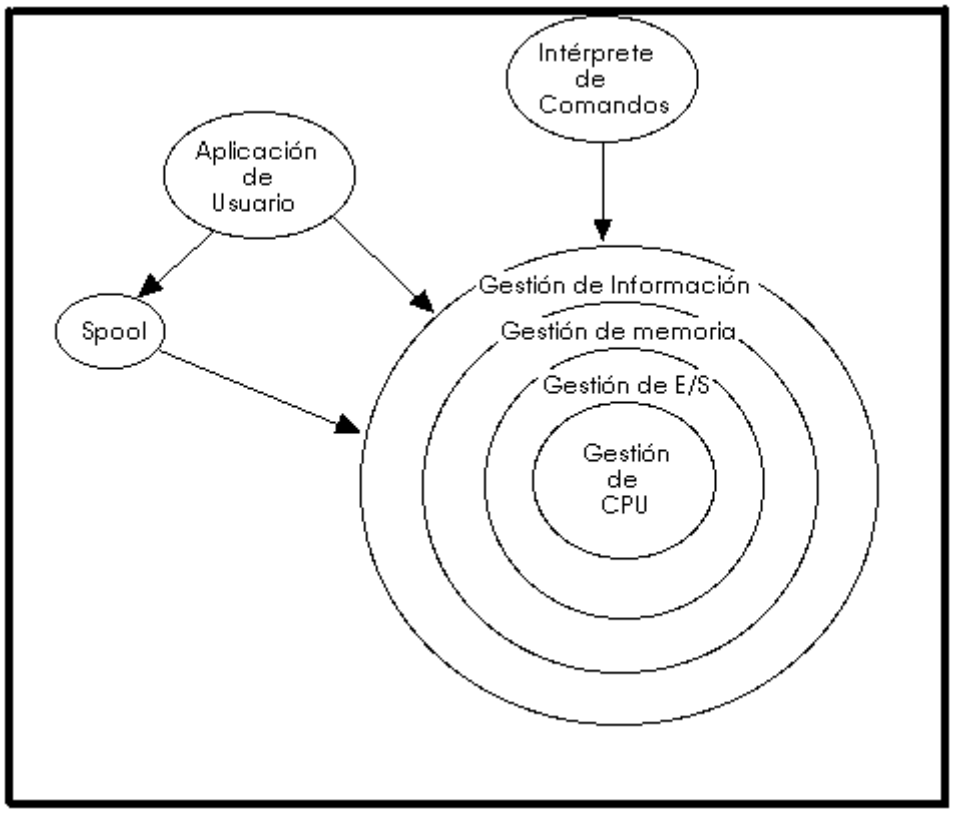


Figura 4-4
Organización
jerárquica
(anillos)

En el sistema de anillos, cada uno tiene una apertura, conocida como puerta o trampa (trap), por donde pueden entrar las llamadas de las capas inferiores. De esta forma, las zonas más internas del sistema operativo o núcleo del sistema estarán más protegidas de accesos indeseados desde las capas más externas. Las capas más internas serán, por tanto, más privilegiadas que las externas.

3.1.3 Máquina virtual

Se trata de un tipo de sistemas operativos que presentan una interface a cada proceso, mostrando una máquina que parece idéntica a la máquina real subyacente. Estos sistemas operativos separan dos conceptos que suelen estar unidos en el resto de sistemas: la multiprogramación y la máquina extendida. El objetivo de los sistemas operativos de máquina virtual es el de integrar distintos sistemas operativos dando la sensación de ser varias máquinas diferentes.

El núcleo de estos sistemas operativos se denomina monitor virtual y tiene como misión llevar a cabo la multiprogramación, presentando a los niveles superiores tantas máquinas virtuales como se soliciten. Estas máquinas virtuales no son máquinas extendidas, sino una réplica de la máquina real, de manera que en cada una de ellas se pueda ejecutar un sistema operativo diferente, que será el que ofrezca la máquina extendida al usuario (Ver Fig. 4-5).

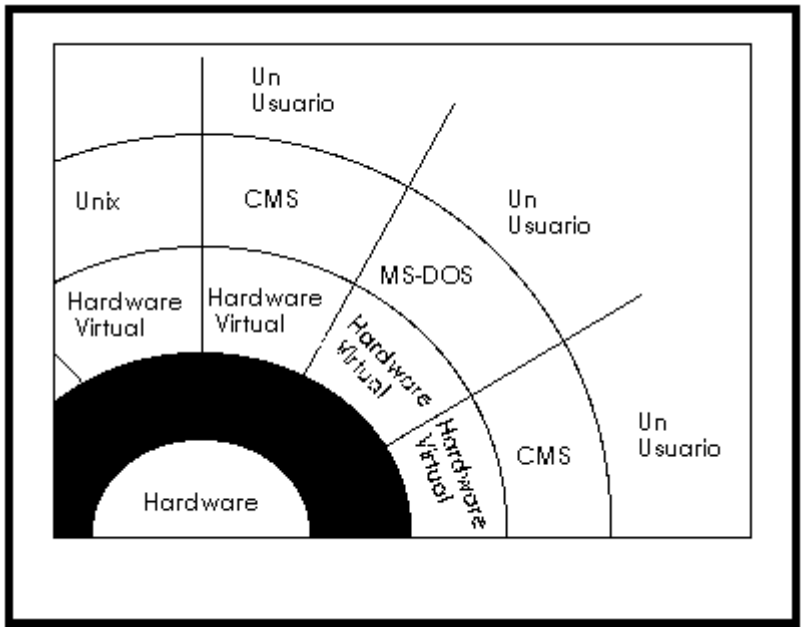


Figura 4-5
Máquina virtual

3.1.4 Cliente-servidor (Microkernel)

El tipo más reciente de sistemas operativos es el denominado Cliente-servidor, que puede ser ejecutado en la mayoría de las computadoras, ya sean grandes o pequeñas.

Este sistema sirve para toda clase de aplicaciones, por tanto, es de propósito general y cumple con las mismas actividades que los sistemas operativos convencionales.

El núcleo tiene como misión establecer la comunicación entre los clientes y los servidores. Los procesos pueden ser tanto servidores como clientes. Por ejemplo, un programa de aplicación normal es un cliente que llama al servidor correspondiente para acceder a un archivo o realizar una operación de entrada/salida sobre un dispositivo concreto. A su vez, un proceso cliente puede actuar como servidor para otro. Este paradigma ofrece gran flexibilidad en cuanto a los servicios posibles en el sistema final, ya que el núcleo provee solamente funciones muy básicas de memoria, entrada/salida, archivos y procesos, dejando a los servidores proveer la mayoría que el usuario final o programador puede usar. Estos servidores deben tener mecanismos de seguridad y protección que, a su vez, serán filtrados por el núcleo que controla el hardware.

3.2 Sistemas operativos por servicios

Esta clasificación es la más comúnmente usada y conocida desde el punto de vista del usuario final. Esta clasificación se comprende fácilmente con el cuadro sinóptico que a continuación se muestra en la Fig. 4-6.

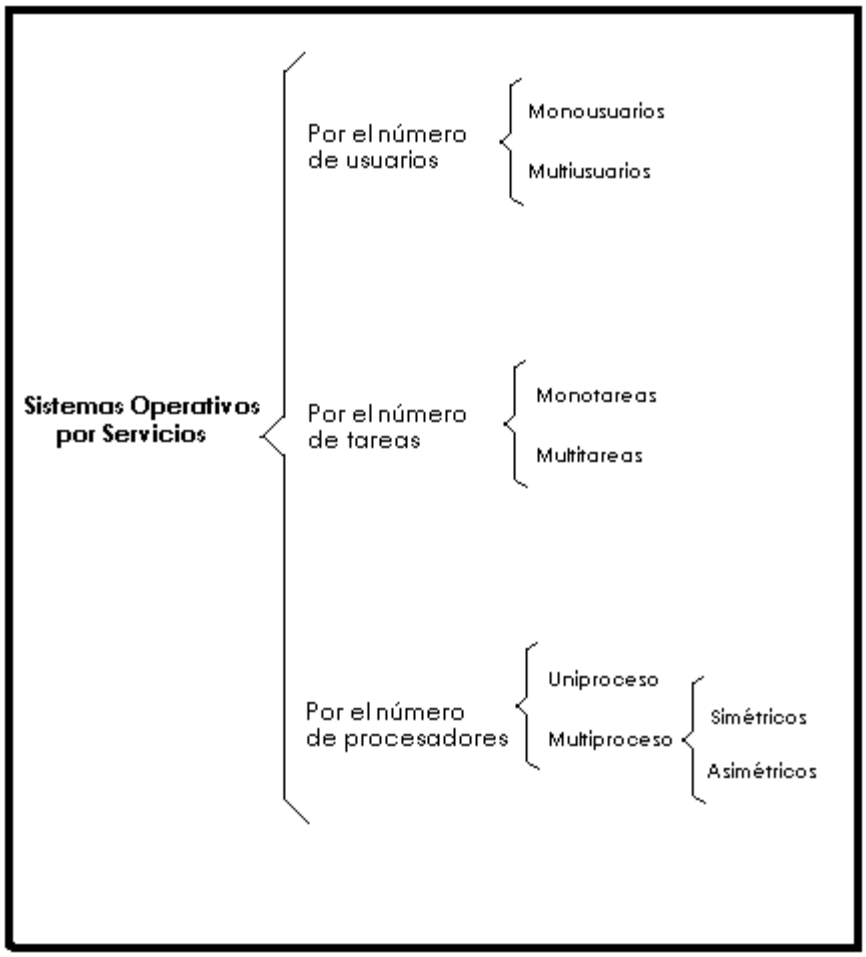


Figura 4-6
Sistemas operativos por servicios

3.2.1 Monousuarios

Los sistemas operativos monousuarios son aquellos que soportan a un usuario a la vez, sin importar el número de procesadores que tenga la computadora o el número de procesos o tareas que el usuario pueda ejecutar en un mismo instante de tiempo. Las computadoras personales típicamente se han clasificado en este renglón.

3.2.2 Multiusuarios

Los sistemas operativos multiusuarios son capaces de dar servicio a más de un usuario a la vez, ya sea por medio de varias terminales conectadas a la computadora o por medio de sesiones remotas en una red de comunicaciones. No importa el número de procesadores en la máquina ni el número de procesos que cada usuario puede ejecutar simultáneamente.

3.2.3 Monotareas

Los sistemas monotarea son aquellos que sólo permiten una tarea a la vez por usuario. Puede darse el caso de un sistema multiusuario y monotarea, en el cual se admiten varios usuarios al mismo tiempo pero cada uno de ellos puede estar haciendo solo una tarea a la vez.

3.2.4 Multitareas

Un sistema operativo multitarea es aquél que le permite al usuario estar realizando varias labores al mismo tiempo. Por ejemplo, puede estar editando el código fuente de un programa durante su depuración mientras compila otro programa, a la vez que está recibiendo correo electrónico en un proceso en background. Es común encontrar en ellos interfaces gráficas orientadas al uso de menús y el ratón, lo cual permite un rápido intercambio entre las tareas para el usuario, mejorando su productividad.

3.2.5 Uniproceso

Un sistema operativo uniproceso es aquél que es capaz de manejar solamente un procesador de la computadora, de manera que si la computadora tuviese más de uno le sería inútil. El ejemplo más típico de este tipo de sistemas es el DOS y MacOS.

3.2.6 Multiproceso

Un sistema operativo multiproceso se refiere al número de procesadores del sistema, que es más de uno y éste es capaz de usarlos todos para distribuir su carga de trabajo. Generalmente estos sistemas trabajan de dos formas: simétrica o asimétricamente. Cuando se trabaja de manera asimétrica, el sistema operativo selecciona a uno de los procesadores el cual jugará el papel de procesador maestro y servirá como pivote para distribuir la carga a los demás procesadores, que reciben el nombre de esclavos. Cuando se trabaja de manera simétrica, los procesos o partes de ellos (*threads*) son enviados indistintamente a cualesquiera de los procesadores disponibles, teniendo, teóricamente, una mejor distribución y equilibrio en la carga de trabajo bajo este esquema.

Se dice que un thread es la parte activa en memoria y corriendo de un proceso, lo cual puede consistir de un área de memoria, un conjunto de registros con valores específicos, la pila y otros valores de contexto. Un aspecto importante a considerar en estos sistemas es la forma de crear aplicaciones para aprovechar los varios procesadores. Existen aplicaciones que fueron hechas para correr en sistemas monoproceso que no toman ninguna ventaja a menos que el sistema operativo o el compilador detecte secciones de código paralelizable, los cuales son ejecutados al mismo tiempo en procesadores diferentes. Por otro lado, el programador puede modificar sus algoritmos y aprovechar por sí mismo esta facilidad, pero esta última opción las más de las veces es costosa en horas hombre y muy tediosa, obligando al programador a ocupar tanto o más tiempo a la paralelización que a elaborar el algoritmo inicial.

3.3 Sistemas operativos por la forma de ofrecer sus servicios

Esta clasificación también se refiere a una visión externa, que en este caso se refiere a la del usuario, el cómo accesa los servicios. Bajo esta clasificación se pueden detectar dos tipos principales: sistemas operativos de red y sistemas operativos distribuidos.

3.3.1 Sistemas operativos de red

Los sistemas operativos de red se definen como aquellos que tiene la capacidad de interactuar con sistemas operativos en otras computadoras por medio de una forma de transmisión con el objeto de intercambiar información, transferir archivos, ejecutar comandos remotos y un sin fin de otras actividades. El punto crucial de estos sistemas es que el usuario debe saber la sintaxis de un conjunto de comandos o llamadas al sistema para ejecutar estas operaciones, además de la ubicación de los recursos a los que desea acceder. Por ejemplo, si un usuario en la computadora hidalgo necesita el archivo matriz.pas que se localiza en el directorio /software/codigo en la computadora Morelos bajo el sistema operativo UNIX, dicho usuario podría copiarlo a través de la red con los comandos siguientes: hidalgo% hidalgo% rcp morelos:/software/codigo/matriz.pas . hidalgo% En este caso, el comando rcp que significa "*remote copy*" trae el archivo indicado de la computadora Morelos y lo coloca en el directorio donde se ejecutó el mencionado comando. Lo importante es hacer ver que el usuario puede accesar y compartir muchos recursos.

3.3.2 Sistemas operativos distribuidos

Los sistemas operativos distribuidos abarcan los servicios de los de red, logrando integrar recursos (impresoras, unidades de respaldo, memoria, procesos, unidades centrales de proceso) en una sola máquina virtual que el usuario accesa en forma transparente. Es decir, ahora el usuario ya no necesita saber la ubicación de los recursos, sino que los conoce por nombre y simplemente los usa como si todos ellos fuesen locales a su lugar de trabajo habitual. Todo lo anterior es el marco teórico de lo que se desearía tener como sistema operativo distribuido, pero en la realidad no se ha conseguido crear uno del todo, por la complejidad que suponen: distribuir los procesos en las varias unidades de procesamiento, reintegrar sub-resultados, resolver problemas de concurrencia y paralelismo, recuperarse de fallas de algunos recursos distribuidos y consolidar la protección y seguridad entre los diferentes componentes del sistema y los usuarios. Los avances tecnológicos en las redes de área local y la creación de microprocesadores de 32 y 64 bits lograron que computadoras mas o menos baratas tuvieran el suficiente poder en forma autónoma para desafiar en cierto grado a los mainframes, y a la vez se dio la posibilidad de intercomunicarlas, sugiriendo la oportunidad de partir procesos muy pesados en cálculo en unidades más pequeñas y distribuirlas en los varios microprocesadores para luego reunir los sub-resultados, creando así una máquina virtual en la red que exceda en poder a un mainframe. El sistema integrador de los microprocesadores que hacer ver a las varias memorias, procesadores, y todos los demás recursos como una sola entidad en forma transparente se le llama sistema operativo distribuido. Las razones para crear o adoptar sistemas distribuidos se dan por dos razones principales: por necesidad (debido a que los problemas a resolver son inherentemente distribuidos) o porque se desea tener más confiabilidad y disponibilidad de recursos. En el primer caso tenemos, por ejemplo, el control de los cajeros automáticos en diferentes estados de la república. Ahí no es posible ni eficiente mantener un control centralizado, es más, no existe capacidad de cómputo y de entrada/salida para dar servicio a los millones de operaciones por minuto. En el segundo caso, supóngase que se tienen en una gran empresa varios grupos de trabajo, cada uno necesita almacenar grandes cantidades de información en disco duro con una alta confiabilidad y disponibilidad. La solución

puede ser que para cada grupo de trabajo se asigne una partición de disco duro en servidores diferentes, de manera que si uno de los servidores falla, no se deje dar el servicio a todos, sino sólo a unos cuantos y, más aún, se podría tener un sistema con discos en espejo (mirror) a través de la red, de manera que si un servidor se cae, el servidor en espejo continúa trabajando y el usuario ni cuenta se da de estas fallas, es decir, obtiene acceso a recursos en forma transparente.

3.3.2.1 Ventajas de los sistemas distribuidos

En general, los sistemas distribuidos (no solamente los sistemas operativos) exhiben algunas ventajas sobre los sistemas centralizados que se describen enseguida.

- Economía: El cociente precio/desempeño de la suma del poder de los procesadores separados contra el poder de uno solo centralizado es mejor cuando están distribuídos.
- Velocidad: Relacionado con el punto anterior, la velocidad sumada es muy superior.
- Confiabilidad: Si una sola máquina falla, el sistema total sigue funcionando.
- Crecimiento: El poder total del sistema puede irse incrementando al añadir pequeños sistemas, lo cual es mucho más difícil en un sistema centralizado y caro.
- Distribución: Algunas aplicaciones requieren de por sí una distribución física.

Por otro lado, los sistemas distribuidos también exhiben algunas ventajas sobre sistemas aislados. Estas ventajas son:

- Compartir datos: Un sistema distribuido permite compartir datos más fácilmente que los sistemas aislados, que tendrían que duplicarlos en cada nodo para lograrlo.
- Compartir dispositivos: Un sistema distribuido permite accesar dispositivos desde cualquier nodo en forma transparente, lo cual es imposible con los sistemas aislados. El sistema distribuido logra un efecto sinérgico..
- Comunicaciones: La comunicación persona a persona es factible en los sistemas distribuidos, en los sistemas aislados no. _ Flexibilidad: La distribución de las cargas de trabajo es factible en el sistema distribuidos, se puede incrementar el poder de cómputo.

3.3.2.2 Desventajas de los sistemas distribuidos

Así como los sistemas distribuidos exhiben grandes ventajas, también se pueden identificar algunas desventajas, algunas de ellas tan serias que han frenado la producción comercial de sistemas operativos en la actualidad. El problema más importante en la creación de sistemas distribuidos es el software: los problemas de compartición de datos y recursos es tan complejo que los mecanismos de solución generan mucha sobrecarga al sistema haciéndolo ineficiente. El checar, por ejemplo, quiénes tienen acceso a algunos recursos y quiénes no, el aplicar los mecanismos de protección y registro de permisos consume demasiados recursos. En general, las soluciones presentes para estos problemas están aún en pañales.

Otros problemas de los sistemas operativos distribuidos surgen debido a la concurrencia y al paralelismo. Tradicionalmente las aplicaciones son creadas para computadoras que ejecutan secuencialmente, de manera que el identificar secciones de código `paralelizable' es un trabajo arduo, pero necesario para dividir un proceso grande en sub-procesos y enviarlos a diferentes unidades de procesamiento para lograr la distribución. Con la concurrencia se deben implantar mecanismos para evitar las condiciones de competencia, las postergaciones indefinidas, el ocupar un recurso y estar esperando otro, las condiciones de espera circulares y , finalmente, los "abrazos mortales" (*deadlocks*). Estos problemas de por sí se presentan en los sistemas operativos multiusuarios o multitareas, y su tratamiento en los sistemas distribuidos es aún más complejo, y por lo tanto, necesitará de algoritmos más complejos con la inherente sobrecarga esperada.

4 SISTEMAS OPERATIVOS MÁS POPULARES PARA PC´S

4.1 Disk operation system (DOS)

CP/M (Control Program for Microcomputers), desarrollado por Gary Kildall fue el primer sistema operativo que podía ejecutarse en microcomputadoras de diferentes fabricantes. Cuenta una anécdota que ejecutivos de IBM fueron a visitar a Kildall para ofrecerle un acuerdo para poner el CP/M en la IBM PC, pero Kildall al parecer estaba ocupado en una sesión de vuelo, y no los pudo atender. El resultado fue que IBM llegó a un acuerdo con un joven llamado Bill Gates para que desarrollara un sistema operativo, que tomó el nombre de MS-DOS.

Para muchos de los que hoy se encuentran trabajando con una computadora el sistema operativo D.O.S (Disk Operating System) fue el primero que les tocó utilizar. De la misma manera, fue también el comienzo para el ahora gigante Microsoft.

Fue desarrollado desde sus principios pero "de a ratos" junto con la empresa IBM, pero diferencias entre las partes hicieron que no fuera un lanzamiento en conjunto. Por el contrario, cada una de las empresas presentó su sistema operativo: PC-DOS (IBM) y MS-DOS (Microsoft).

En D.O.S. todavía hoy se utilizan antiguos comandos CP/M, tales como DIR, REN y TYPE que aun hoy funcionan bajo la máquina virtual D.O.S. de Windows.

Un tercer competidor para estos sistemas operativos (y de muy buena calidad) fue el DR-D.O.S. de la empresa Digital Research que incluía comandos más detallados y de mayor funcionalidad, pero que con el tiempo, y gracias a las campañas publicitarias de Microsoft fue quedando relegado a un segundo lugar compartido con el PC-DOS de IBM. Luego de unos años el DR-D.O.S. fue adquirido por Novell que presentó una nueva versión conocida como Novell D.O.S. que realmente casi no tuvo cabida en el mercado, aunque tenía muy buena calidad.

Volviendo a D.O.S., este era simplemente una pantalla de texto con una línea de comandos que indicaba en qué directorio se encontraba y nada más. Se tenía que saber que

cosas había que escribir para que la máquina hiciera algo. No había "menús contextuales" ni pantallas gráficas que guiaran. Era lo menos intuitivo que se puede imaginar (pero funcionó).

Las versiones que lo hicieron famoso en el mundo entero fueron la 3.0 y la 3.3, mientras que la más utilizada en México fue la 5.0, que introdujo muchos cambios a sus antecesoras. La versión 4.0 de D.O.S. estuvo plagada de errores, por lo cual casi no se uso (los usuarios se mantuvieron con la versión 3.30).

La última versión del producto como tal fue la 6.22, ya que luego apareció Windows 95 que no necesitaba de DOS, pero que incluía la versión 7.0.

4.2 Windows: una máscara

Las dos primeras versiones de este entorno operativo (nótese que no era un sistema operativo en sus principios) eran muy lentas y no tuvieron mucho éxito entre el público consumidor (ningún éxito en realidad). Lo único que logró fue que Apple iniciara un juicio a Microsoft por ser "muy parecido" a su sistema operativo MacOs. El mismo fue abandonado tiempo después por Apple debido a que no había resolución.

El éxito de Windows se produjo con su versión 3.0 (y más aún con la 3.1) cuando se comenzó a aprovechar las capacidades de los procesadores "386" y se le dio un mejor manejo a la memoria.

Fue simplemente un "shell" para DOS, ya que sin este no funcionaba. Y por esa razón no es un sistema operativo, sino un entorno operativo (una plataforma). El atractivo que tuvo para con la gente fue su casi real facilidad de uso y su presentación gráfica que hacía olvidar las pantallas negras de DOS.

Un tiempo más tarde se le agregaron capacidades para trabajar con redes y se pasó a la versión 3.11 (para grupos de trabajo). Esta fue la última versión comercial que salió al mercado antes de que Windows 95 hiciera su aparición.

4.3 Windows 95

Desde el principio, Windows 95 se publicitó como un sistema operativo de 32 bits. Pero cuando salió a la luz se pudo ver que esto no era totalmente verdad: era un sistema operativo (ya no un entorno), porque no necesitaba de ningún otro programa para poder funcionar (aunque se incluía el DOS 7.0). Por otro lado, la promesa de los 32 bits (programas más rápidos y mejor aprovechamiento de la memoria) no se cumplió. Muchas de las partes de este sistema operativo fueron de 16 bits como sus antecesores. Esto se explicó diciendo que era así por la gran cantidad de programas heredados de las versiones anteriores (Windows 3.1).

Un año antes apareció en el mercado un sistema operativo de 32 bits que sería la competencia directa al tan publicitado Windows 95: OS/2 de IBM. En principio fue

desarrollado en cooperación entre IBM y Microsoft (como años atrás con el DOS), y como años atrás surgieron diferencias que hicieron que cada empresa presentara su producto.

OS/2 es un sistema operativo totalmente de 32 bits que muchos expertos consideran mejor, más estable y con mayores prestaciones que Windows 95, pero que nuevamente las campañas publicitarias relegaron a un segundo lugar, ya que la gran mayoría de los desarrolladores decidieron hacer sus programas compatibles con Windows 95 y no con OS/2.

4.4 Windows 98

Windows 98 no representó para los usuarios comunes ningún cambio significativo. Sólo un poco de maquillaje gráfico y alguna que otra utilidad nueva o mejorada (como el "liberador de espacio" o el viejo "defrag"). Pero si trajo cosas nuevas bajo el brazo: el soporte completo para los 32 bits (al fin), y la eliminación del DOS como sistema independiente (ya que no incluye una nueva versión, sino un emulador del mismo), como algunos ejemplos.

La gran virtud de Windows 98 es la de seguir enganchando a los usuarios finales y hacer que Microsoft mantenga el liderazgo mundial en sistemas operativos.

Windows NT es un sistema operativo de 32 bits especializado en redes que utiliza otro sistema para el manejo de los archivos, y por lo tanto "incompatible" por el momento con Windows 98 y 95.

4.5 Windows 2000 /Windows ME

Microsoft dio un nuevo paso en sus principales productos y nacieron así Windows 2000 y Windows ME. El primero, es el sucesor de NT, por lo que está orientado a empresas y hereda muchas de las características de este.

Su gran estabilidad, su soporte para varios procesadores, su alto nivel de seguridad, además de sus impresionantes capacidades para desenvolverse como servidor lo hacen, como se dijo, la mejor opción para una empresa. Es rápido y lo suficientemente fácil de configurar casi para cualquier persona, pero hay que tener en cuenta que tiene poco soporte para el agregado de periféricos como tarjetas de video o de sonido. Es decir, este no es un sistema operativo totalmente apto para la multimedia.

Al ser de esta manera, no resulta muy inteligente su uso en hogares, donde comúnmente se encontraran juegos, música en la PC, enciclopedias multimedia y demás.

Ahí es donde entra Windows Millennium (ME), sucesor de Windows 98 (aunque muchos dicen que es la tercera edición de éste después de Windows SE).

Es un sistema operativo con gran facilidad de uso, robustez, mejoras en multimedia, además de comunicaciones en Internet.

Aunque no cuenta con la estabilidad de Windows 2000 es más seguro que Windows 98 y 98 SE (segunda edición) ya que se han incorporado una serie de utilidades para proteger el sistema operativo y hacerlo más resistente a las instalaciones de programas y drivers de terceros que, en definitiva, son las principales causas de paros y pantallas azules en sus predecesores.

Una de las cosas interesantes que se encuentra en Windows ME es que el modo DOS, tal como se conoce, ha dejado de existir. Ya no es posible iniciar el sistema en sólo símbolo del sistema o apagar el sistema reiniciando en modo MS-DOS. Tanto es así, que los archivos AUTOEXEC.BAT y CONFIG.SYS ya no tienen ninguna función en ME (salvo durante la instalación).

Por lo tanto, DOS, tal como se venía conociendo, ha muerto. Aunque un experto en los sistemas operativos de Microsoft (Paul Thurrott), explica que, en realidad, sólo se ha ocultado. El MS-DOS sigue estando debajo de Windows ME de la misma manera que lo estaba en el 95 o el 98 aunque se haya escondido el símbolo del sistema.

Lo que realmente se ha eliminado de Windows ME es el soporte para aplicaciones DOS de 16 bits en modo real.

4.6 Windows XP

Este sistema operativo es la mejora mas importante técnicamente desde Windows 9x, y unifica las versiones separadas que hubo estos años: WINDOWS 9x/ME para usuarios hogareños contra Windows NT/2000 para usuarios corporativos con requerimientos de trabajo en redes de alto nivel.

Windows XP se distribuye en 2 versiones principales: Windows XP Home Edition y Windows XP Profesional. La versión Home no tiene tanto soporte para redes, mientras que la profesional si.

Windows XP además de constituirse en la unión de los entornos mencionados, es en realidad la continuación de Windows NT/2000. Se destaca en este producto su alto grado de integración con las redes e Internet, además de proveer una nueva interfase gráfica que se hace notar cuando se comienza a utilizar. Los cambios de interfaz son básicamente estéticos. La diferencia real con sus predecesores esta dada por el soporte LAN, software de grabación de CDs, multimedia, escritorio remoto y manejo de mútiples usuarios.

Algo muy importante es el hecho de que con esta versión se ha puesto especial énfasis en los drivers. WXP ahora es muchísimo mas renuente que sus predecesores a instalar drivers no certificados para el mismo. Con esto se ha pretendido reducir al máximo las ya tan conocidas (y sufridas) "pantallas azules", aduciendo que la mayoría de las causas de inestabilidad de las versiones anteriores estaba dada por el uso de drivers no certificados, obsoletos o mal desarrollados. Se destaca la búsqueda inteligente que hace el S.O. al momento de instalar un dispositivo nuevo, escaneando unidades en busca de los drivers correctos.

4.7 Linux

Cuando Linus Torvalds comenzó a trabajar sobre Minix para obtener su propio sistema operativo no tenía ni la más remota idea de lo que su trabajo llegaría a ser en todo el mundo. Este sistema operativo es totalmente distinto a los vistos anteriormente por una gran cantidad de razones. He aquí algunas de ellas.

- No fue desarrollado por una gran empresa:

Linus Torvalds desarrolló el kernel (el corazón) del sistema y luego liberó el código fuente del mismo en Internet para que cualquier programador que se animara pudiera modificarlo y agregarle lo que quisiera. Así, el Linux que hoy se conoce fue creado por cientos de programadores libres alrededor del mundo y no por una empresa.

- Es gratis y abierto:

Todo el sistema operativo es totalmente gratuito (al igual que muchísimos de sus programas), si se posee una conexión a Internet es posible bajarlo. Lo que algunas empresas hacen es empacar el sistema y algunos programas y grabarlos en CD's, que junto con algún manual es lo que luego venden.

Además, junto con el sistema vienen los códigos fuentes del mismo (y de algunos programas) para que pueda ser modificado a gusto del usuario (si este es un programador experimentado), es por esto que se dice que es abierto.

- Nació a partir de otro sistema operativo:

Es una modificación del sistema Minix, que a su vez nació como una reducción de UNIX, "el único sistema operativo verdadero, a partir del cual se crearon los demás" (incluido DOS) según la opinión de muchos Hackers.

Este sistema operativo es el elegido por las empresas que proveen acceso a Internet, debido a su gran estabilidad y eficiencia (cosas imposibles de lograr con Windows aunque se tenga el mejor hardware), aparte de ser gratuito. Además, posee un muy buen manejo de redes y seguridad, lo que está haciendo que muchas empresas e instituciones (escolares sobre todo) lo tengan en cuenta para reemplazar sus sistemas actuales.

En un principio, Linux también era una pantalla negra en modo texto y muy poco intuitivo (al igual que DOS e UNIX). Pero desde hace un tiempo se desarrollaron entornos gráficos (KDE, Gnome, etc.) que no tienen nada que envidiarle a Windows y que hacen que más usuarios (menos experimentados) se animen a usarlo.

Por lo anterior y el gran auge de Internet este es el sistema operativo que más crecimiento ha tenido en los últimos años, y el que se perfila quizá como el sistema operativo del futuro.

CAPÍTULO 5

Herramientas para el análisis de inventarios

Las herramientas para el análisis de inventarios que se abordan en este capítulo están dividas en 2 grupos: el software especialmente diseñado para el análisis de inventarios, y programas de análisis matemático.

1 SOFTWARE ESPECIALIZADO EN EL ANÁLISIS DE INVENTARIOS

Dentro de este grupo se encuentran programas tales como: WinQSB, The Management Scientist, TORA, Plantillas de Excel, etc. Y aunque el campo de estudio de algunos de estos programas resulta tan amplio como el de la investigación de operaciones y administración científica, cada uno de ellos cuenta con apartados para el análisis de inventarios.

1.1 WinQSB

Desarrollado por Yih-Long Chang este programa contiene algunos de los mas útiles y populares métodos cuantitativos usados en las ciencias administrativas, investigación de operaciones y administración de operaciones.

Por medio de una interfase interactiva, los profesionales y estudiantes tienen fácil acceso a los diferentes módulos de decisión para resolver una gran variedad de problemas.

Cada módulo de WinQSB es brevemente descrito a continuación:

- ***Linear Programming (LP) e Integer Linear Programming (ILP)*:** Este programa resuelve los problemas de LP usando el metodo simplex o el método gráfico y los problemas de ILP usando el procedimiento branch-and-bound.
- ***Quadratic Programming (QP) e Integer Quadratic Programming (IQP)*:** Este programa resuelve los problemas de QP usando el método simplex modificado o el método gráfico y los problemas de IQP usando el procedimiento branch-and-bound.
- ***Nonlinear Programming (NLP)*:** Este programa resuelve los problemas no lineales no forzados usando el método de búsqueda y los problemas no lineales forzados usando el método de la función de castigo.
- ***Network Modeling (NET)*:** Este modulo resuelve los problemas de red incluyendo flujo de red (transbordo), transporte, asignación, caminos cortos, máximo flujo, cruces mínimos y problemas de viajes de vendedores.
- ***Dynamic Programming (DP)*:** Resuelve 3 tipos populares de problemas dinámicos: Diligencia, mochila y problemas de planeación de producción e inventarios.
- ***PERT/CPM*:** Este módulo resuelve los problemas de planeación de proyectos usando el método de ruta crítica y la técnica de evaluación y revisión. Así mismo realiza análisis de choque, análisis de costos, análisis de probabilidad y simulación.

- ***Queuing analysis (QA)*:** Este programa resuelve el rendimiento de sistemas de colas de etapa simple usando la formula de cercanía, aproximación o simulación.
- ***Queuing system simulation (QSS)*:** Este programa modela y simula sistemas de colas simples y multietapas con componentes, incluyendo poblaciones de clientes arribando, servidores, colas y/o colectores de basuras.
- ***Inventory theory and systems (ITS)***: Resuelve problemas de control de inventarios: problemas de cantidades económicas a pedir (EOQ), problemas de descuento de cantidad de la orden, problemas de periodos probabilísticos simples y problemas de tamaño dinámico de lotes; y evalúa y simula 4 sistemas de control de inventarios: (s, Q), (s, S), (R, S) y (R, s, S).
- ***Forecasting (FC)*:** Este módulo resuelve proyecciones de series de tiempo usando 11 diferentes métodos y además utilizando regresiones lineales de múltiples variables.
- ***Decision analisys (DA)*:** El programa resuelve 4 típicos problemas de decisión: Análisis Beyesiano, análisis de tablas de rentabilidad, análisis de árbol de decisión y la teoría del juego de cero suma.
- ***Markov process (MKP)*:** Este programa resuelve y analiza el proceso de Markov.
- ***Quality control charts (QCC)*:** Construye gráficos de control de calidad para variables y datos de atributos y así mismo realiza análisis de gráficas relacionadas.
- ***Acceptance sampling analysis (ASA)*:** Este programa desarrolla y analiza los planes de muestreos de tolerancias para atributos y características de calidad variable.
- ***Job scheduling (JOB)*:** Este programa resuelve los problemas de taller de tareas y programación del flujo de trabajo usando generación heurística y aleatoria.
- ***Aggregate planning (AP)*:** Soluciona los problemas de planeamiento agregado a las demandas de satisfacción del consumidor con mínimos o aceptables costos relacionados.
- ***Facility location and layout (FLL)*:** Este módulo resuelve los problemas de facilidades de localización, disposición funcional y balanceo de línea de producción.
- ***Material requirements planning (MRP)*:** El programa efectúa la planeación de requerimiento de materiales y determina que, cuando y cuanto cuestan los materiales y componentes que son requeridos para satisfacer un plan de producción de productos finales para un horizonte de planeación.

Adicionalmente, el modulo de teoría de inventarios y sistemas cuenta con las siguientes características:

- Resolución bajo el método convencional EOQ (Cantidad económica a ordenar).
- Resuelve problemas de descuentos por cantidad.
- Resuelve problemas probabilísticos de un solo periodo (del voceador).
- Aborda problemas de tamaño de lote dinámico, mediante 10 diferentes métodos.
- Resolución, evaluación y simulación de inventarios bajo 4 diferentes políticas de revisión: (s, Q), (s, S), (R, S), y (R, s, S).
- Análisis gráfico del costo para el modelo EOQ simple, y con descuentos por cantidad.

- Realización y presentación gráfica del análisis paramétrico para el modelo EOQ, descuentos por cantidad y del voceador.

Las cuatro diferentes políticas de revisión (s, Q), (s, S), (R, S), y (R, s, S) son resueltas por métodos de búsqueda.

La siguiente figura (Figura 5-1) muestra la interfaz de WinQSB.

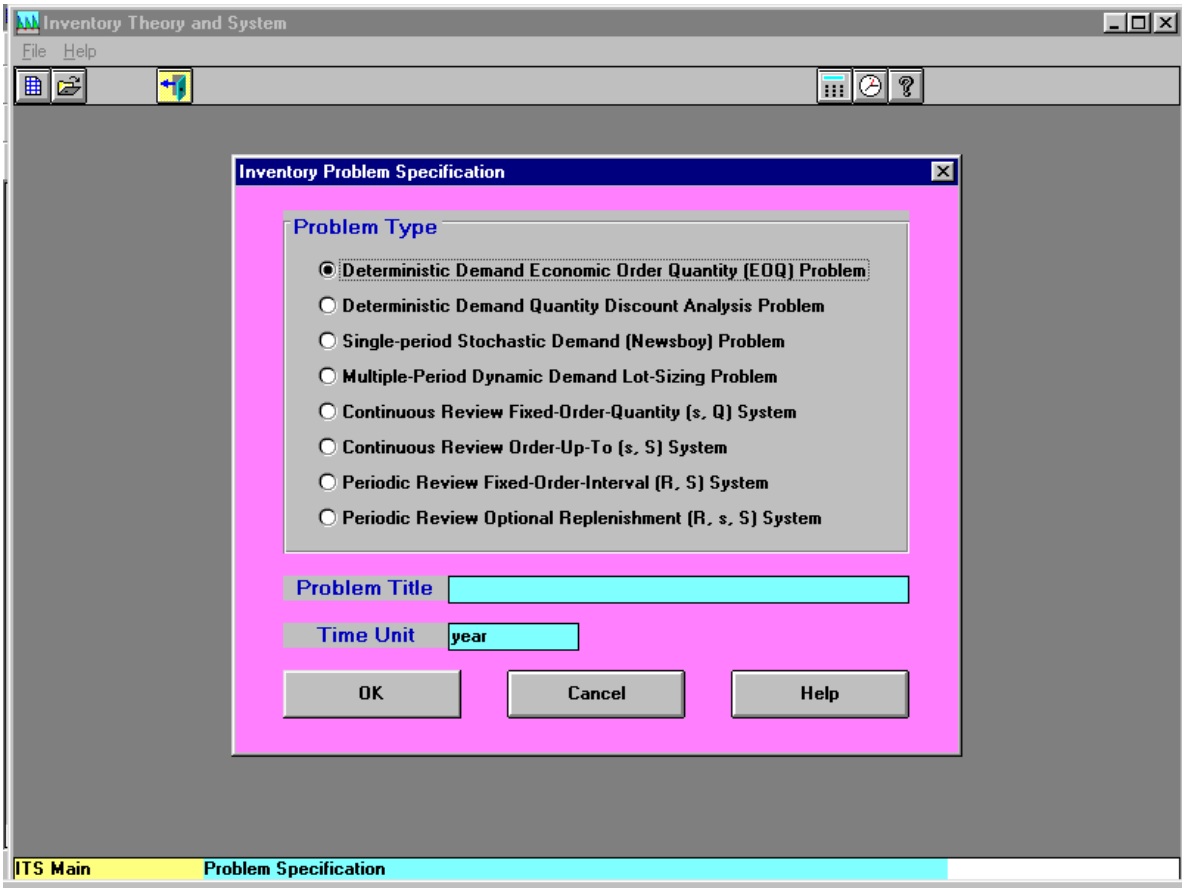


Figura 5-1
Interfaz de WinQSB

1.2 The Management Scientist

Software enfocado a la resolución de problemas clásicos dentro de las área de investigación y administración de operaciones. Desarrollado por David R. Anderson, Dennis J. Sweeney, y Thomas A. Williams.

Cuenta con módulos enfocados a la resolución de problemas tan variados como: programación lineal, trasporte, asignamiento, programación lineal de enteros, ruta más corta, mínimo árbol de expansión, PERT/CPM, *inventarios*, líneas de espera, análisis de decisión, pronósticos y procesos de Markov.

En el módulo de inventarios se proveen decisiones óptimas para los siguientes modelos:

- Cantidad económica a ordenar (EOQ).
- Cantidad económica a producir (EPQ).
- Cantidad económica a ordenar con escasez planeada.
- Descuentos por cantidad.
- Del voceador.
- Punto de reorden con demanda probabilística.

En la Figura 5-2 se presenta la interfaz de the Management Scientist.

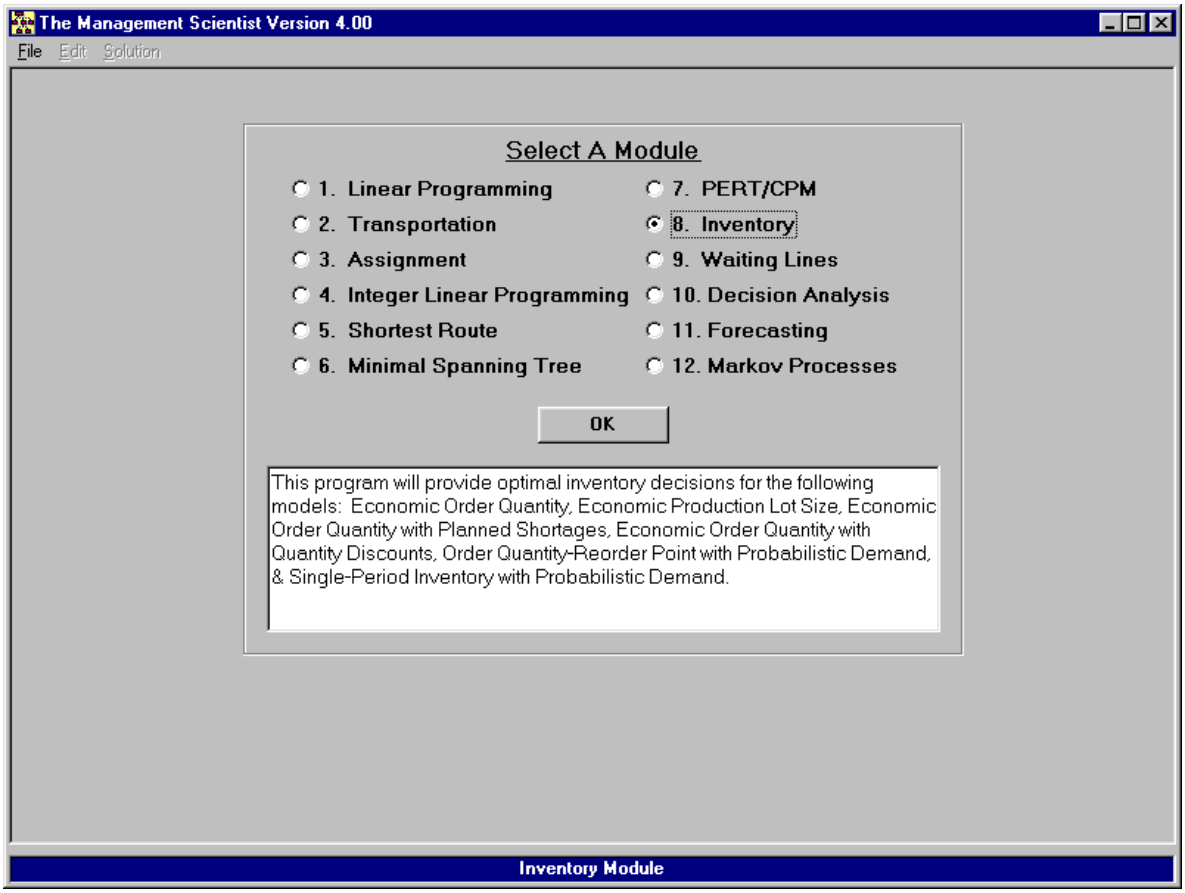


Figura 5-2
Interfaz de The Management Scientist

1.3 TORA

Es un software distribuido con el libro "*Operations research: an introduction*" de H. Taha, que permite resolver problemas de programación lineal, modelos de transporte, modelos de red, problemas de programación entera, modelos de colas, histogramas, predicciones y *modelos de inventario*.

Desarrollado por Hamdy A. Taha, es el más modesto de todos los programas presentados hasta el momento, como es un programa de 16 bits se ejecuta en el sistema operativo

MS-DOS o bien su emulador. Aunque es mas bien antiguo, resulta útil como una primera aproximación a los programas de análisis de inventarios.

Con TORA (Fig. 5-3)se pueden analizar modelos como el EOQ simple y algunas de sus variaciones, además de modelos de tamaño de lote dinámico bajo el método Silver-Meal.

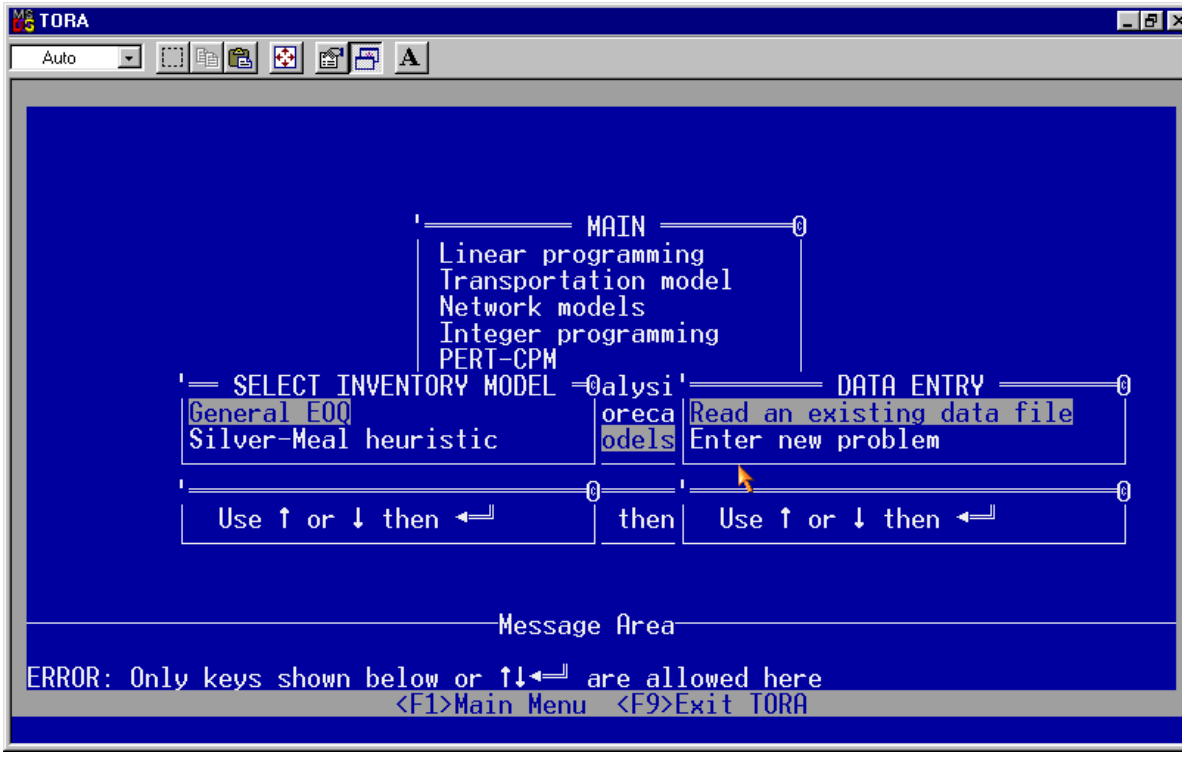


Figura 5-3
Interfaz de TORA

1.4 Plantillas de Excel

Estas son plantillas distribuidas junto con el libro "*Introducción a la investigación de operaciones*" de Hillier y Lieberman, utilizando métodos cuantitativos analizan una variedad de modelos de inventarios auxiliándose de el poder de cálculo de Excel.

Si bien estas plantillas ocupan bien poco de las funciones avanzadas de Excel como los controles Active X, Macros o el editor Visual Basic de Excel, cumplen con su cometido valiéndose en su mayor parte de herramientas tan sencillas como el manejo de fórmulas con referencia a celdas, y algunas funciones adicionales como las lógicas y estadísticas.

Una ventaja de este programa sobre los anteriores, radica en que realiza un análisis tan o más exhaustivo sobre modelos de inventarios estocásticos (probabilísticos) como en los determinísticos, algunos de ellos son:

- Modelo básico EOQ (Solver).
- Modelo básico EOQ (Analytical).

- Modelo EOQ con escasez planeada (Solver).
- Modelo EOQ con escasez planeada (Analytical).
- Modelo EOQ con descuentos por cantidad (Analytical).
- Plantilla para el modelo estocástico de revisión continua.
- Plantilla para el modelo estocástico de un solo periodo y productos perecederos, sin costo por ordenar.
- Plantilla para el modelo estocástico de un solo periodo y productos perecederos, con costo por ordenar.
- Plantilla para el modelo estocástico de 2 periodos sin costo por ordenar.
- Plantilla para el modelo estocástico con un número infinito de periodos, sin costo por ordenar.

La diferencia entre las plantillas Solver y Analytical reside en que la plantilla Solver esta diseñada para encontrar el *lote económico* mediante aproximaciones, y la Analytical lo calcula de inmediato.

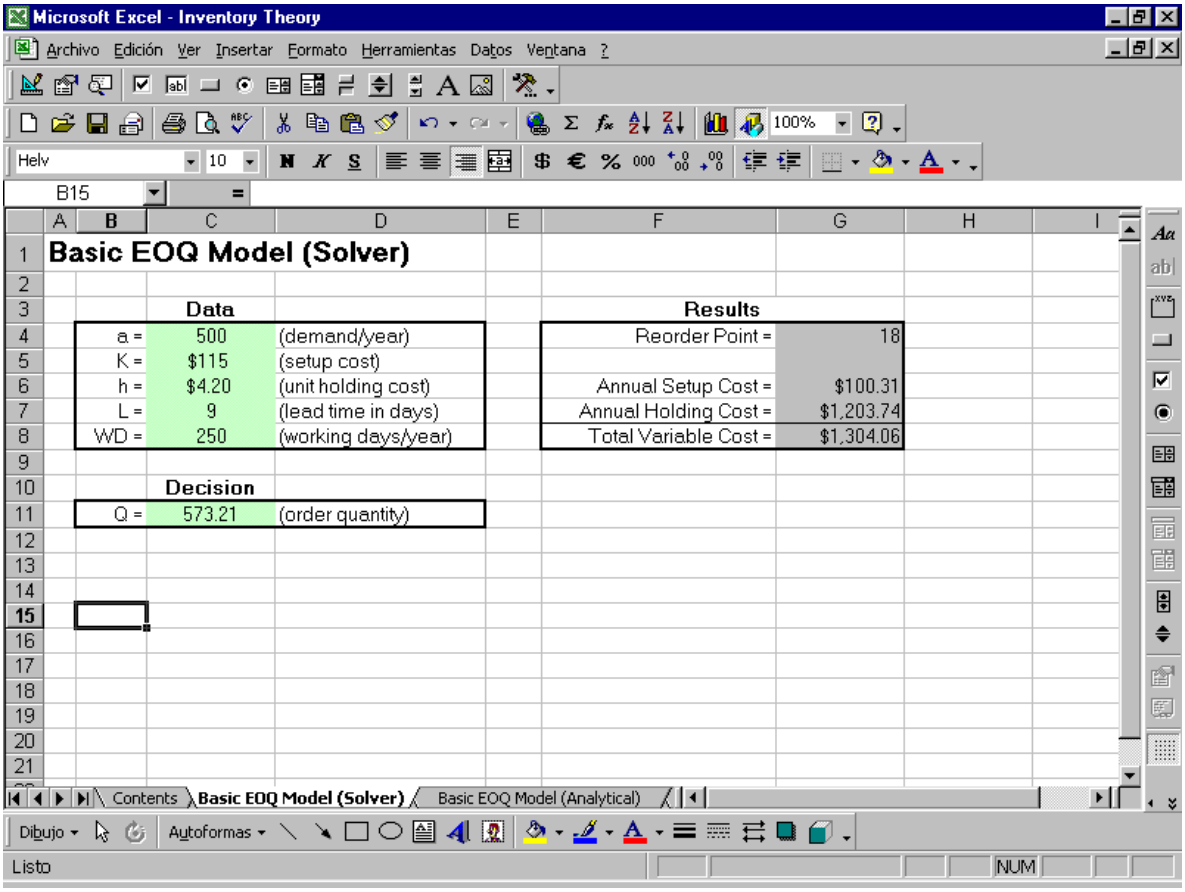


Figura 5-4
Plantilla para el modelo básico EOQ (Solver)

2 SOFTWARE PARA ANÁLISIS MATEMÁTICO

Este es un software de propósito general que puede ser útil en gran variedad de aplicaciones científicas y de ingeniería. Dentro de esta clasificación se encuentran programas como DERIVE, Maple, MATLAB, etc.

2.1 DERIVE

Derive es un "ayudante matemático para el ordenador personal". Con este programa se pueden realizar cálculos numéricos, simbólicos y gráficos, permitiendo la resolución de numerosos ejercicios de álgebra, análisis, cálculo, geometría, estadística, etc.

Las posibilidades del programa, junto a las mínimas necesidades de hardware que requiere, lo hacen bastante útil en cualquier nivel de la enseñanza; su utilización y aprovechamiento dependerá de la imaginación para el desarrollo de aplicaciones y ejercicios que permitan la utilización de DERIVE como una herramienta más en el proceso de aprendizaje.

Dentro de las capacidades de análisis que tiene DERIVE se encuentran:

- Edición y análisis de expresiones matemáticas.
- Resolución de ecuaciones, inecuaciones y sistemas de ecuaciones.
- Manejo de funciones financieras, estadísticas, probabilísticas, etc.
- Aplicaciones al cálculo diferencial e integral.
- Manejo de números complejos.
- Representación gráfica de curvas y superficies.
- Cálculo matricial.

La siguiente figura (Figura 5-5) muestra la interfaz de DERIVE.

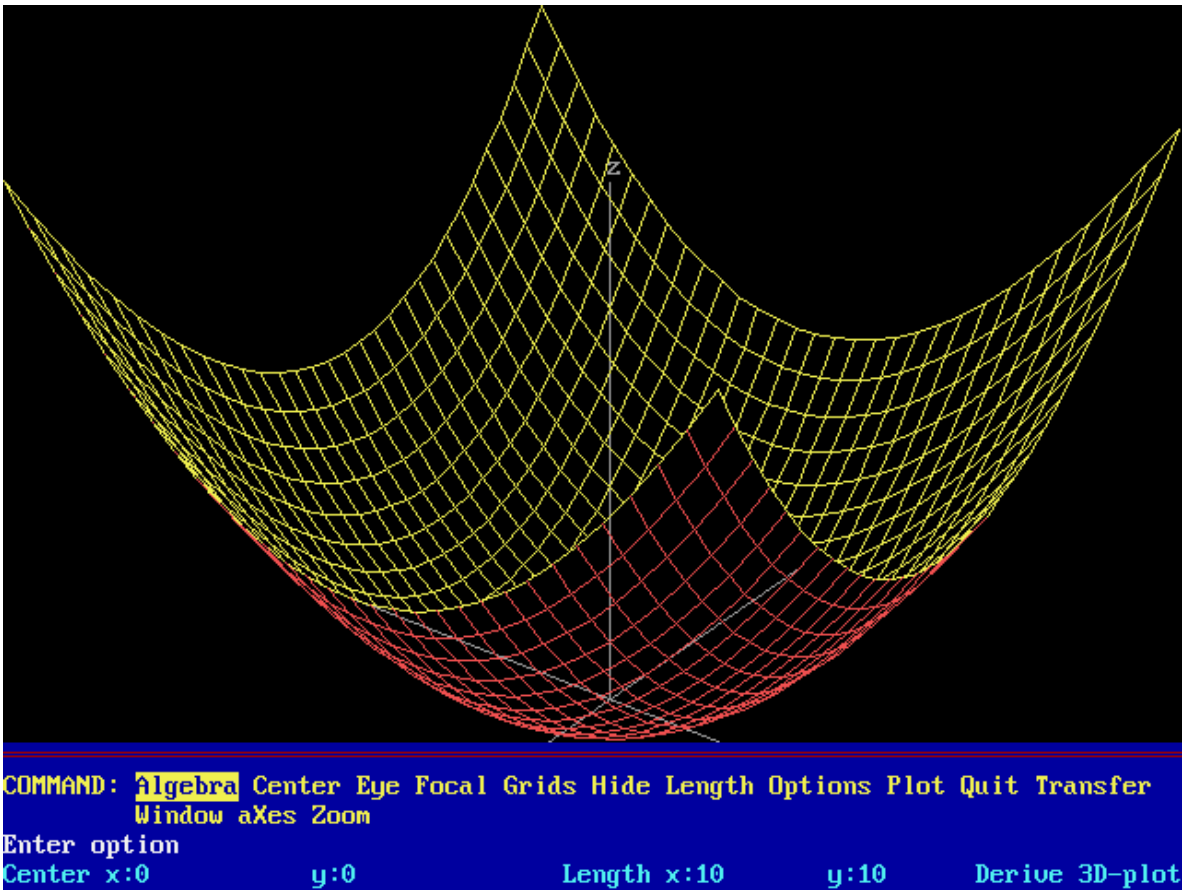


Figura 5-5
Interfaz de DERIVE

2.2 Maple X

Maple X es un programa fácil de entender enfocado a matemáticas avanzadas. Incluye facilidades para álgebra, cálculo, matemáticas discretas, gráficas, cómputo numérico y muchas otras áreas de la matemática. También provee un ambiente único para el rápido desarrollo de programas matemáticos usando su vasta librería de funciones y operaciones.

Maple X incluye paletas, menús contextuales, y un nuevo modo de entrada. Notación Matemática Estándar, que permite la entrada y manipulación de expresiones matemáticas sin un conocimiento detallado de la sintaxis de Maple. En la Figura 5-6 se muestra la interfaz de dicho programa.

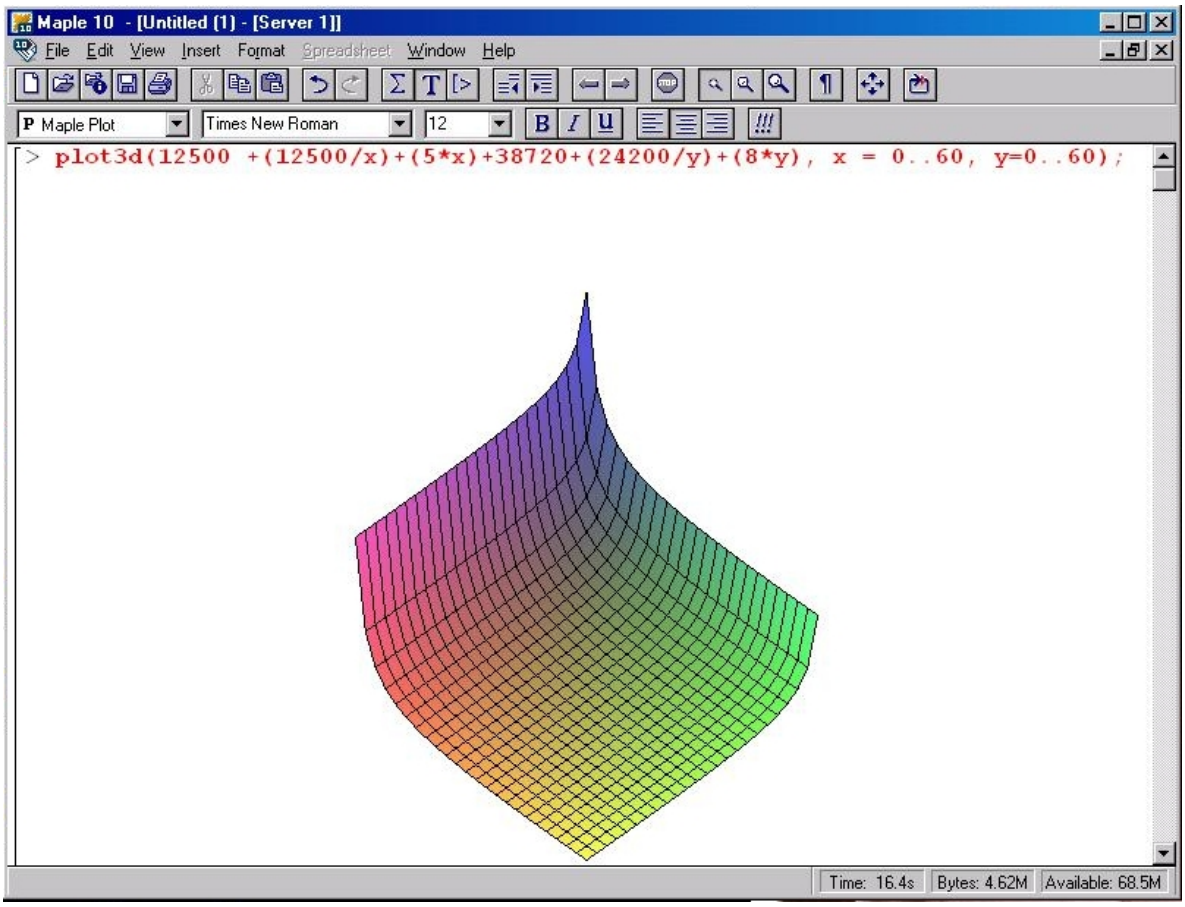


Figura 5-6
Interfaz de Maple X

2.3 MATLAB

MATLAB puede considerarse como un lenguaje de programación enfocado a análisis matemático, aunque sería difícil describirlo en unas cuantas palabras. He aquí algunas de sus características notables para los análisis matemáticos.

- La programación es sencilla
- Hay continuidad entre valores enteros, reales y complejos.
- La amplitud de intervalo y la exactitud de los números es considerable.
- Cuenta con una amplia biblioteca matemática.
- Abundantes herramientas gráficas.
- Capacidad para vincularse con lenguajes de programación tradicionales.
- No hay distinción entre números reales, complejos, enteros, de precisión sencilla y doble. Todos están conectados continuamente.

Incluye capacidades para álgebra, álgebra lineal, manejo de polinomios e interpolación, integración, diferenciación, raíces de ecuaciones no lineales, etc. En la Figura 5-7 se presenta la interfaz de MATLAB.

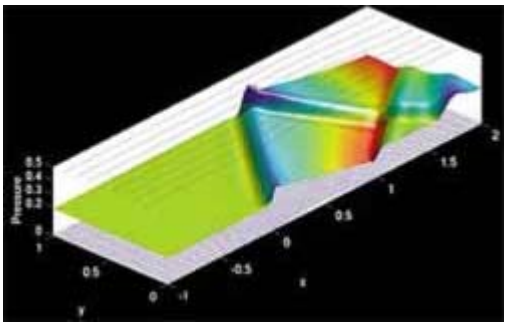


Figura 5-7
Interfaz de MATLAB

CAPÍTULO 6

Desarrollo del Programa

Se ha decidido que el Sistema Operativo bajo el cuál corra el programa final sea Windows (en cualquiera de sus versiones), a causa de que:

1. Es el sistema operativo más difundido del mundo, por lo tanto, cualquier software desarrollado para funcionar en él tendrá un alto grado de portabilidad y uso.
2. La aplicación será diseñada esencialmente para adecuarse a las necesidades de cátedra de la carrera de Ingeniería Industrial en la Facultad. Y los estudiantes de la carrera aunque manejan otros sistemas operativos como LINUX, UNIX, OS-DOS, GNOME, Mac OS, etc. Principalmente utilizan alguna versión de Windows como su sistema operativo base.
3. Cuenta con una formidable facilidad de uso (cualquier estudiante a nivel universitario sabe manejar Windows).
4. La mayoría de los programas especializados en la Administración y Control de Inventarios trabajan en una plataforma Windows.

Y para desarrollar el programa se utilizará el Lenguaje de Programación Visual Basic 6.0, por las siguientes razones:

1. Con Visual Basic (VB) se aprovechan al máximo las posibilidades de Windows en cuanto a intercambio de información, librerías, drivers y controladores, manejo de bases de datos, etc. Por lo tanto, se satisfacen todas las necesidades de programación en el entorno Windows.
2. Facilidad de programación. Para desarrollar una aplicación de las prestaciones que se requiere en esta tesis es preciso ser un programador experto en C o Visual C (por ejemplo), y el que redacta este texto apenas es un programador de nivel intermedio.
3. VB es la primera herramienta (la más popular) en construcción de aplicaciones en el entorno Windows.
4. En la mayor parte de las aplicaciones, las herramientas aportadas por VB son más que suficientes para lograr un programa de altas prestaciones.
5. Se pueden desarrollar aplicaciones para Windows más rápidamente.

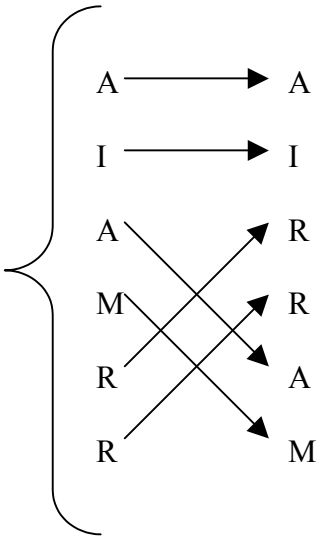
1 DISEÑO DE LA INTERFAZ PRINCIPAL

1.1 Nombre de la aplicación

El objetivo central de la presente tesis es el de desarrollar un software capaz de analizar el problema del modelo de inventarios de artículos múltiples con restricción de recursos (Presupuesto y Espacio). De aquí que el nombre de la aplicación sea:

“Análisis de Inventarios de Artículos Múltiples con Restricción de Recursos”

Tomando la primera letra de cada palabra y ordenándolas



= **“AIRRAM”**

Se obtiene la palabra “**AIRRAM”**, que será el nombre corto de la aplicación (desde este punto para referirse al programa principal se utilizará el término AIRRAM).

1.2 El Proyecto

Para empezar hay que correr Visual Basic 6.0, Figura 6-1.

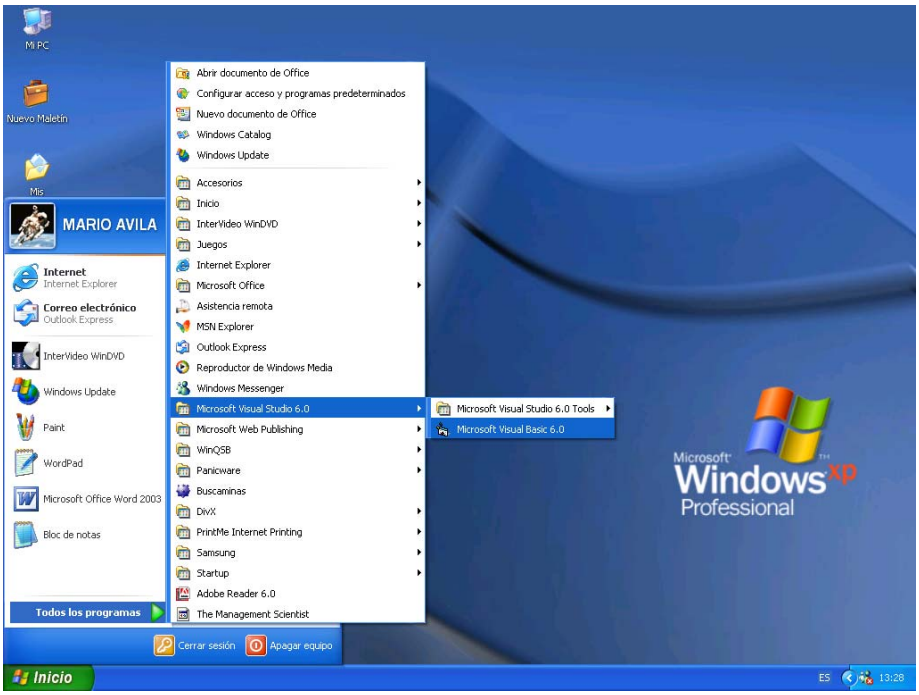


Figura 6-1
Ubicación de Visual Basic 6.0

Y escoger la creación de un nuevo proyecto estándar ejecutable, Figura 6-2.

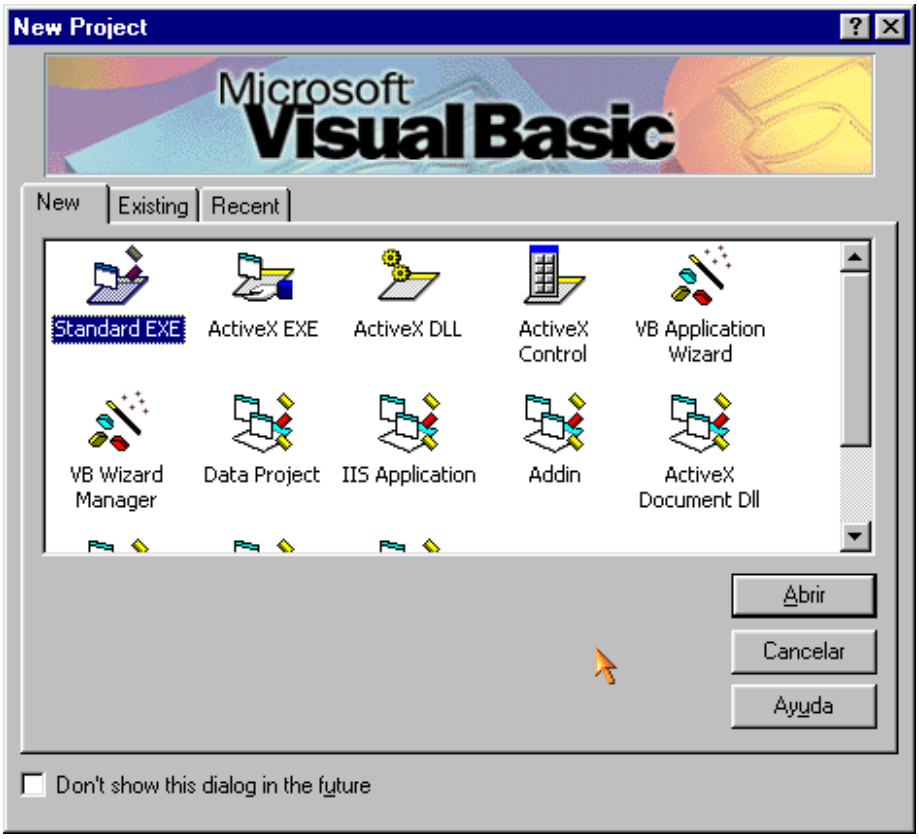


Figura 6-2
Ventana del tipo de Proyecto

Ya que se ha cargado el nuevo Proyecto en Visual Basic, se debe crear la interfaz principal de usuario que puede ser de 2 tipos: *interfaz de documento único (SDI – Single Document Interface) o interfaz de documentos múltiples (MDI – Multiple Document Interface)*, Figura 6-3. En este caso se usará la interfaz MDI (Figura 6-4), debido a que fue diseñada para simplificar el intercambio de información entre documentos dependientes todos de una misma aplicación. Así, en lugar de tener múltiples copias abiertas de la misma aplicación, se tendrá una sola copia y múltiples ventanas de documentos abiertas. Un ejemplo de aplicación con una interfaz MDI es Word.

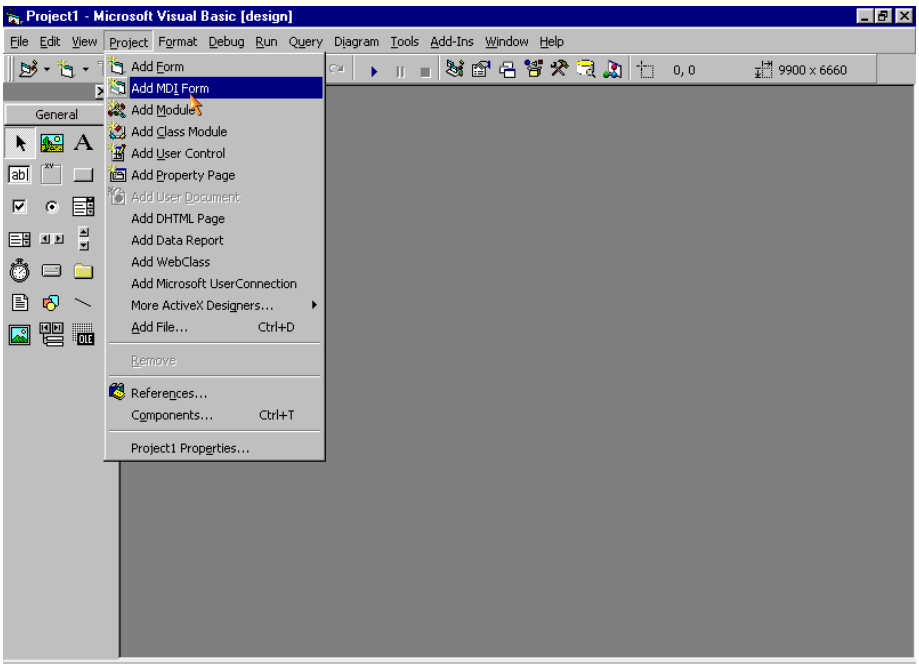


Figura 6-3
Carga de la interfaz MDI o formulario MDI

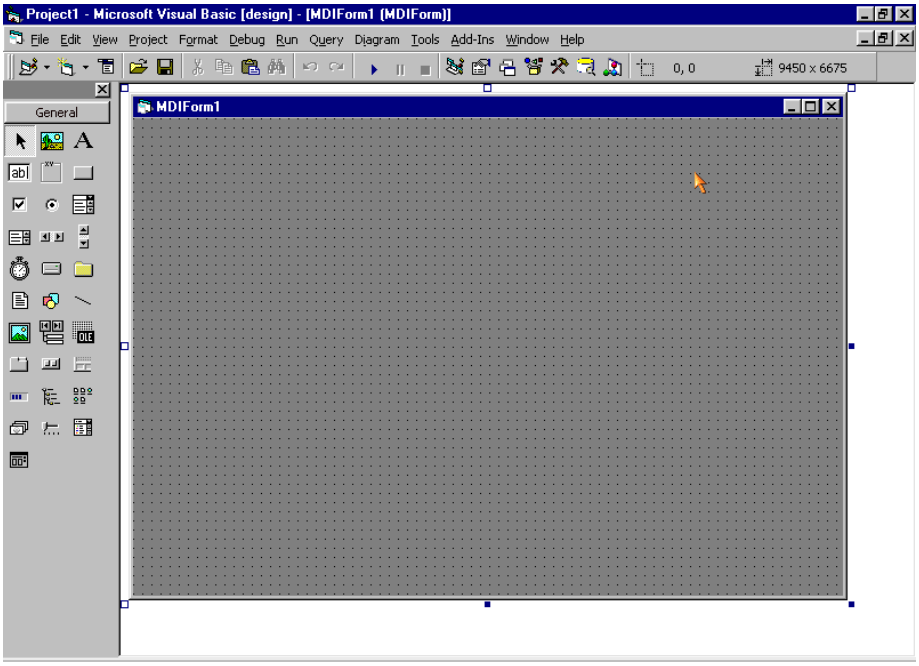


Figura 6-4
Formulario MDI

Posteriormente, hay que ejecutar la orden *Propiedades* del menú *Proyecto*, hacer clic en la pestaña *General* y seleccionar como formulario inicial el formulario MDI. El nombre de código del formulario MDI es *MDIForm1*.

El área en gris del formulario MDI proporciona un área de trabajo para todos los formularios (ventanas) que se creen. Al formulario MDI se le considera la ventana "padre" de los formularios subsecuentes creados dentro de él, conocidos como ventanas "hija".

1.3 Diseño del área de trabajo

Con el deseo de imprimir profesionalismo a la aplicación, se decide diseñar una presentación más elaborada para el área de trabajo del formulario MDI. Los pasos a seguir son:

1. Insertar un escudo de la Universidad y otro de la Facultad en una imagen generada en el editor gráfico PAINT.
2. Rotular la leyenda Departamento de Ingeniería Industrial, nombre de la aplicación y del programador.
3. Exportar esta composición al editor de imágenes Corel PHOTO-PAINT 9.
4. Utilizar el efecto TERRAZZO para crear un mosaico con el escudo de la Facultad.
5. Regresar el mosaico a PAINT e invertir sus colores.
6. Abrir la *Ventana de Propiedades* del formulario MDI y vincular el mosaico a la propiedad *Picture*.

El resultado se muestra en la siguiente figura (Figura 6-5):

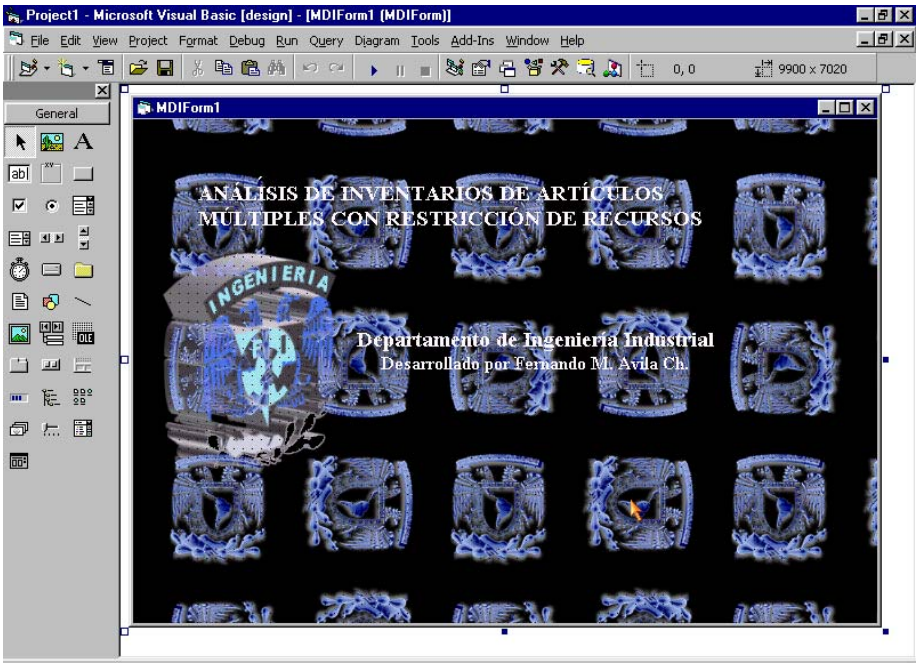


Figura 6-5
Área de trabajo del formulario MDI

1.4 Diseño del Menú o Barra de Menús

Con el objetivo de proporcionar al usuario final un conjunto de órdenes, lógicamente relacionadas, agrupadas bajo un mismo título se creará un Menú o Barra de Menús.

La Barra de Menús debe de tener la siguiente estructura:

Barra de Menús

- Archivo
  - Abrir... Ctrl.+A
  - Guardar como... Ctrl.+G
  - Imprimir Ctrl.+I
  - Salir Ctrl.+S
- Editar Tabla
  - Copiar Ctrl.+C
  - Pegar Ctrl.+P
  - Borrar Del
- Análsis con Restricción de...
  - Presupuesto
  - Espacio
  - Presupuesto y Espacio
- Más Aplicaciones
  - WinQSB (ITS)
  - The Management Scientist
  - Plantilla de Excel (IT)
  - Tora
- Herramientas
  - Calculadora
  - Derive
- Ayuda
  - Temas de Ayuda F1
  - Acerca de AIRRAM

Ya que se tiene la estructura del menú hay que implantarla en el formulario MDI con la ayuda del *Editor de Menús*, Figura 6-6.

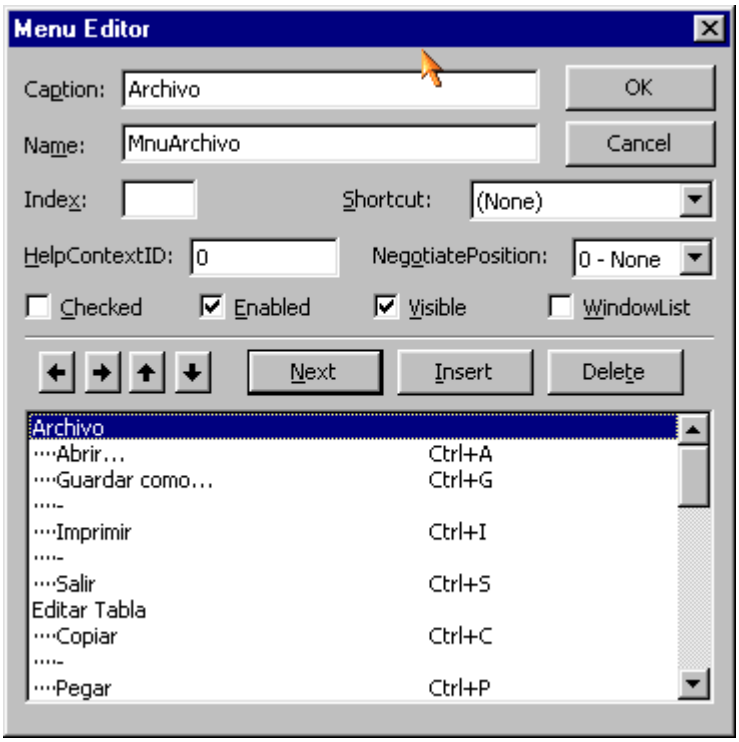


Figura 6-6
Editor de Menús

Para crear el menú, los pasos a ejecutar son los siguientes:

1. Seleccionar el formulario MDI y a continuación ejecutar la orden *Editor de Menús* del menú *Herramientas*, o bien hacer clic en el botón  dispuesto para tal fin en la barra de herramientas estándar.
2. En la caja de texto *Caption* se escribe el título del menú que se desea crear, el cual aparecerá en la barra de menús. En la casilla *Name* se escribe el nombre utilizado en el código para referirse al menú.
3. Para diferenciar un elemento de un menú del propio menú, se sangra el título del elemento. Para ello, se selecciona y se hace clic en el botón flecha hacia la derecha (→). Por ejemplo, *Abrir...* es un elemento del menú *Archivo*.
4. Para agrupar las órdenes en función de su actividad se puede utilizar un *separador*, por lo cual se escribe un único guión (-) en la caja *Caption* del editor de menús. También se especifica un nombre cualquiera (*Name*).

Algunos de los elementos del menú tienen activa la propiedad *Shorcut*, es decir, una tecla o combinación de teclas que permiten activarlos. Por ejemplo, la orden *Imprimir* del menú *Archivo*, tiene asociado el *Shorcut* Ctrl.+I.

Las propiedades *Enabled* y *Visible* serán relevantes a lo largo del desarrollo de AIRRAM, pero eso se abordará más adelante. Solo hace falta decir que *Enabled* es útil para desactivar una orden en un momento en el cual no tiene sentido que esté activa y *Visible* es útil cuando durante la ejecución se desea ocultar una orden.

La Barra de Menús queda de la siguiente forma (Figura 6-7):

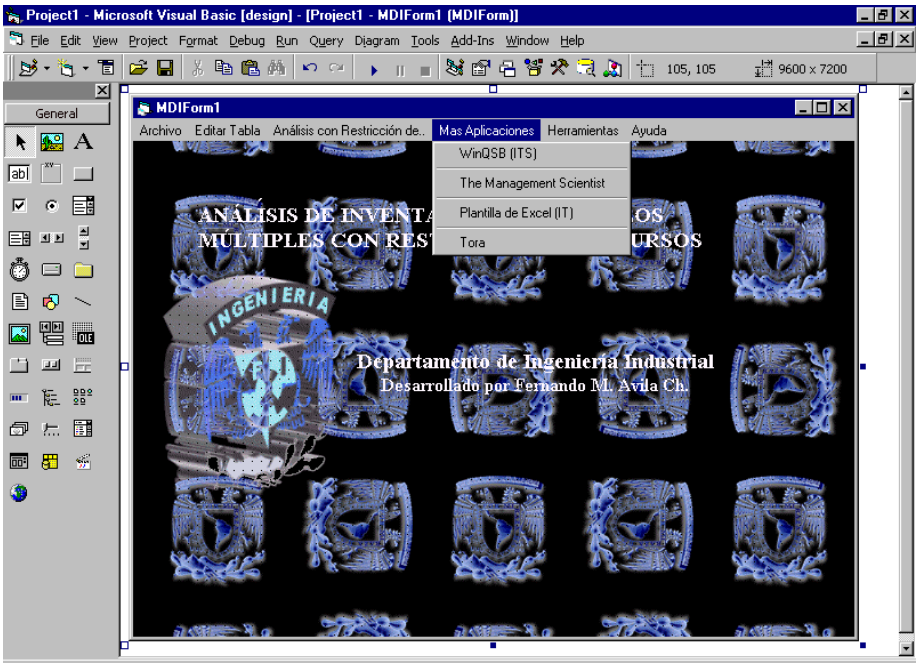


Figura 6-7
Barra de Menús

1.5 Barra de Título

La barra de título de una aplicación con interfaz gráfica por lo general indica si la aplicación esta activa, el título de la misma, si se encuentra procesando información y el icono que la identifica. La forma de ligar todas estas características a la interfaz principal de AIRRAM es:

1. Abrir la *ventana de propiedades* del formulario MDI (Figura 6-8) y vincular el nombre de la aplicación (AIRRAM Análisis de Inventarios de Artículos Múltiples con Restricción de Recursos) a la propiedad *Caption*.
2. Vincular a la propiedad *Icon* alguno de los iconos (archivos *.ico) de la basta colección con la que cuenta Visual Basic (en este caso una estrella amarilla de picos naranja).

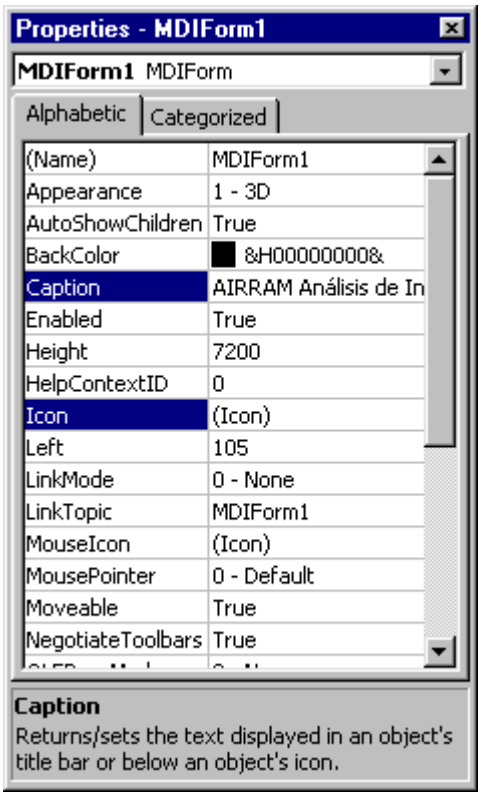


Figura 6-8
Ventana de propiedades del formulario MDI

La barra de título queda de la siguiente manera (Figura 6-9):

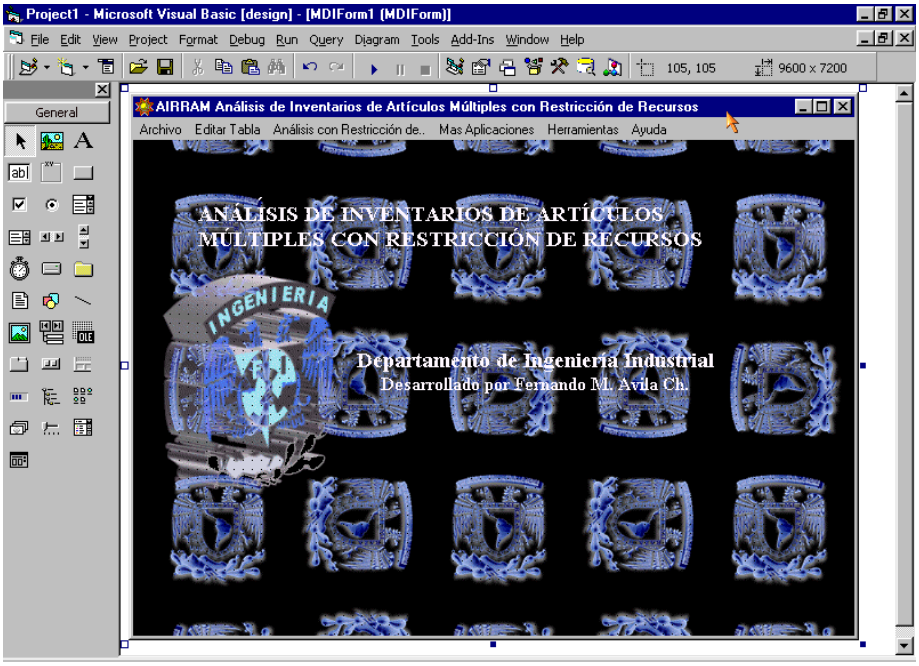


Figura 6-9
Barra de Título

1.6 Barra de Herramientas

Generalmente, una barra de herramientas contiene botones que realizan la misma función que las órdenes de uso más frecuente de los menús de la barra de menús de una aplicación. Lo que se persigue con una barra de herramientas es proporcionar al usuario una interfaz gráfica que le permita tener un acceso rápido a las operaciones que se realizan con más frecuencia en una aplicación.

El control *Toolbar* que es necesario para añadir una barra de herramientas al formulario MDI es un control *ActiveX* (*.ocx), y por lo tanto, no se encuentra dentro de los controles intrínsecos incluidos en la *caja de herramientas* de Visual Basic. Para añadir este control (y algunos otros muy útiles) hay que ejecutar la orden *Componentes* del menú *Proyecto* y añadir el componente *ActiveX* denominado *Microsoft Windows Common Controls 6.0.*

La barra de herramientas va a proporcionar un acceso rápido a las siguientes órdenes de la barra de menús:

- Abrir, Guardar, Imprimir.
- Presupuesto, Espacio, Presupuesto y Espacio.
- WinQSB, Calculadora, Derive.
- Ayuda.

1.6.1 Lista de Imágenes

La barra de herramientas va a necesitar alrededor de 10 imágenes para sus botones de mapa de bits. El control que almacena las imágenes y permite un acceso fácil a las mismas, es el control "lista de imágenes" (*ImageList*). 

Una *lista de imágenes* es una colección de imágenes del mismo tamaño accesibles por su índice.

Algunas de las imágenes (Figura 6-10) se diseñaron en el editor de gráficos PAINT, mientras otras se tomaron de aplicaciones Windows y se modificaron con el mencionado editor gráfico.

Figura 6-10
Imágenes de la barra de herramientas

Lógicamente, para utilizar *ImageList* es necesario agregar dicho control al formulario MDI. El nombre de código del control *ImageList* es *ImageList1*.

Antes de añadir las imágenes anteriores al control *ImageList*, primero hay que visualizar el diálogo *página de propiedades* del control (Figura 6-11). Para ello, se hace clic sobre el control utilizando el botón derecho del ratón y se elige la orden *Propiedades* del menú contextual que se visualiza. Después se siguen los pasos indicados a continuación:

1. Se establece el tamaño de las imágenes en la ficha *General* (35 x 38 *Decatwips*). Un *twip* es 1/20 de punto de impresora (1440 *twips* equivalen a una pulgada).[6]
2. Se insertan las 10 imágenes en la ficha *Imágenes* (cada una tiene un número índice que la identifica).

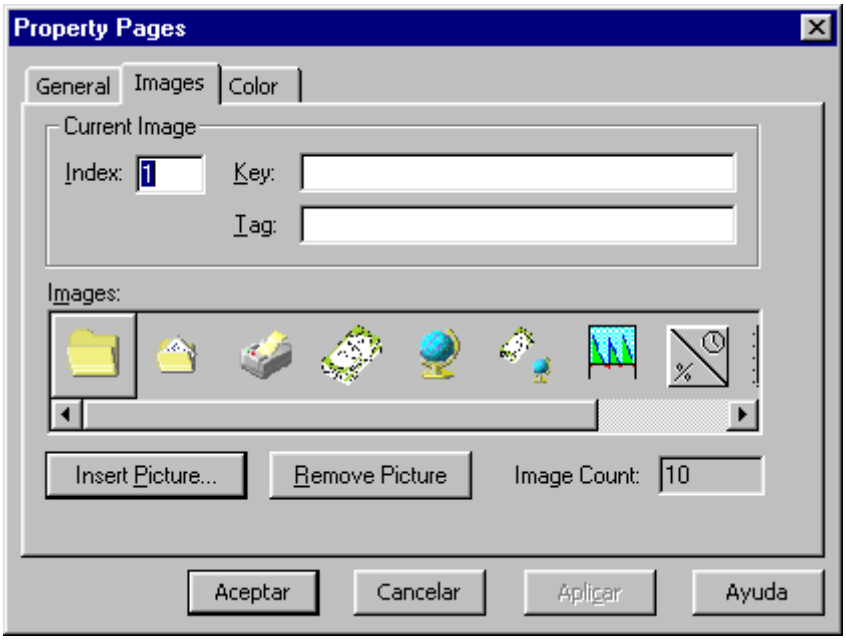


Figura 6-11
Página de propiedades del control *ImageList*

1.6.2 Control *Toolbar*

Para utilizar la barra de herramientas en el formulario MDI se debe agregar un control *Toolbar* en dicho formulario. El nombre de código del control *Toolbar* es *Toolbar1*.

A continuación hay que añadir los botones, y se hace visualizando el diálogo *Página de propiedades* del control. Se hace clic sobre el control utilizando el botón derecho del ratón y se elige la orden *Propiedades* del menú contextual que se visualiza. El procedimiento se continua como sigue:

[6] Tomado de Fco. Javier Ceballos, *Enciclopedia de Microsoft Visual Basic 6*, México, ed. Alfaomega, 2001, p. 337.

1. Se vincula el nombre del control *ImageList* que se utiliza (*ImageList1*) a la propiedad **ImageList** de la ficha *General* (Figura 6-12).

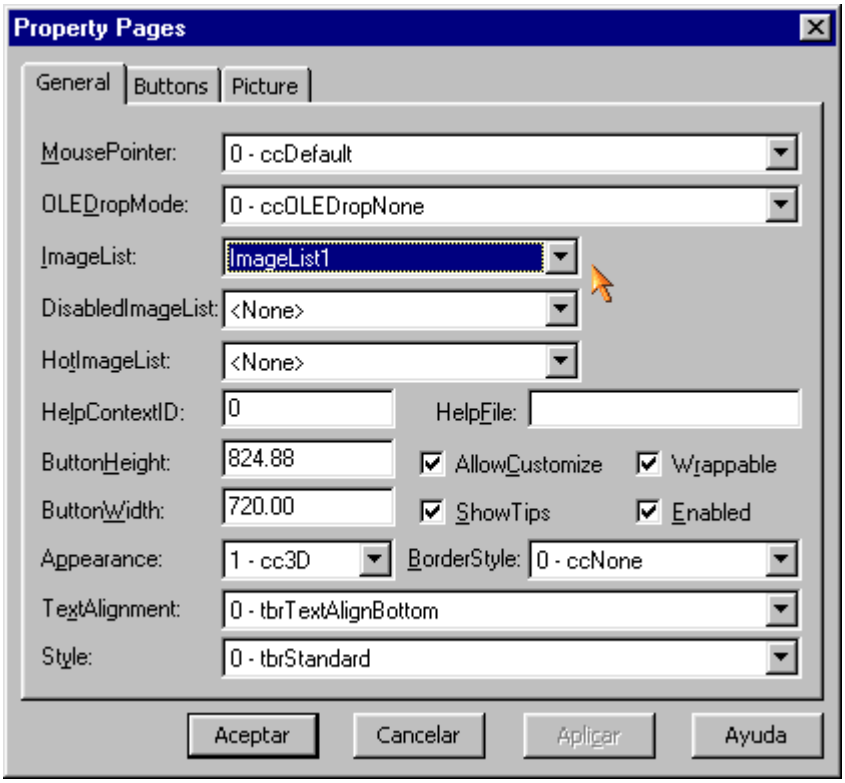


Figura 6-12
Página de propiedades del control *Toolbar* (ficha *General*)

2. En la ficha *Botones* (Figura 6-13) se insertan uno a uno los 10 botones de la barra de herramientas, y se asignan las siguientes propiedades:
    - *Caption*: Título del botón.
    - *Key*: Nombre de código del botón.
    - *Style*: Estilo del botón, se utilizan botones de pulsación (*tbrDefault*) y separadores (*tbrSeparator*); un separador no requiere ninguna otra propiedad. Los separadores sirven para agrupar botones de la misma naturaleza (se insertan 3).
    - *Image*: Índice de la imagen en la lista de imágenes que se asigna al botón.

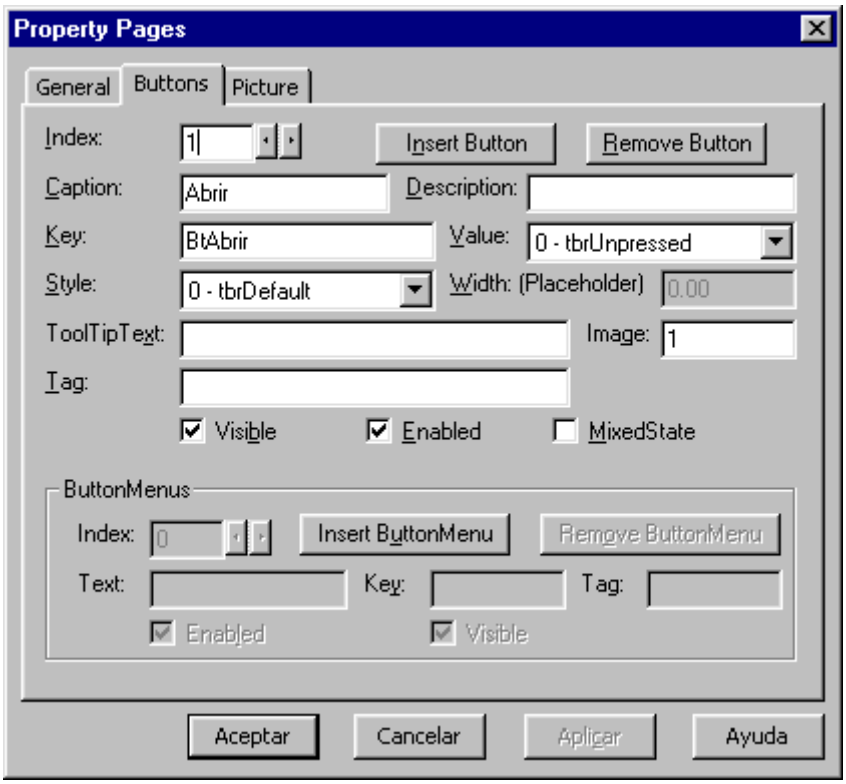


Figura 6-13
Página de propiedades del control *Toolbar* (ficha *Botones*)

Las propiedades *Visible* y *Enabled* de cada botón están por default en su estado verdadero, eso indica que todos los botones se pueden ver y usar. La barra de herramientas queda así (Figura 6-14):

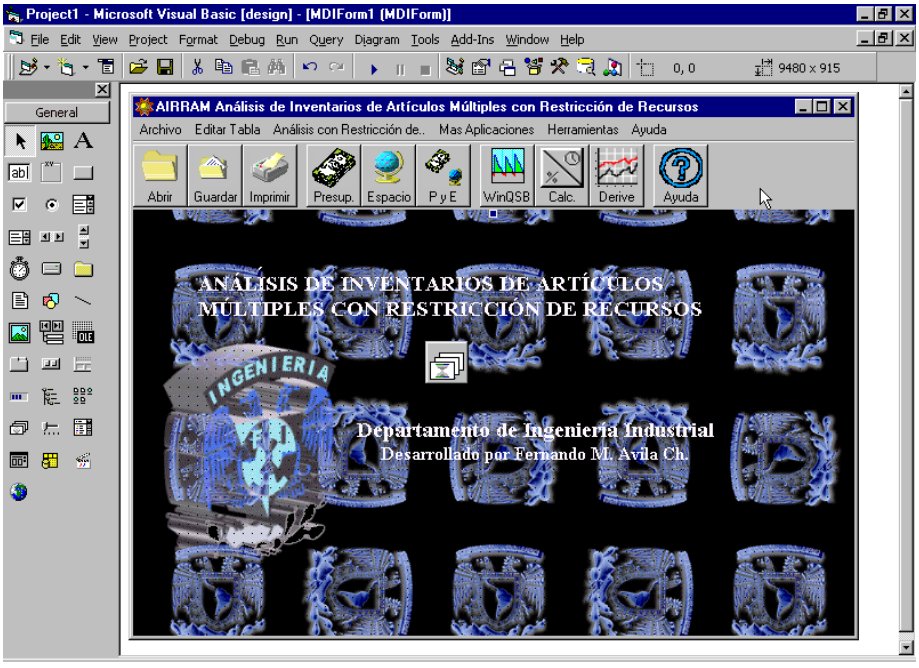


Figura 6-14
Barra de Herramientas

1.6.3 Código asociado al control *Toolbar*

Ahora hay que escribir el código que haga trabajar a la barra de herramientas. Cada vez que el usuario final hace clic en uno de los botones ocurre el evento **ButtonClick**. Para responder a este evento hay que programar el procedimiento privado *Sub Toolbar1_ButtonClick* en la *ventana de código* del formulario.

El argumento *Botón* del procedimiento es una referencia al objeto **Button** representativo del botón sobre el que se ha hecho clic.

La respuesta del evento **ButtonClick** depende de en que botón se hizo clic, por lo que la sentencia de control **Select Case** utiliza la propiedad *Key* del objeto **Button** para determinar el botón en que se hizo clic. Cuando se determina en que botón se hizo clic, simplemente se ejecuta el procedimiento vinculado con la orden de los menús a la que hace referencia el botón (todavía no se programa el procedimiento vinculado con la orden de los menús).

Se utiliza la propiedad *Key* porque su valor esta garantizado como único.

```
Private Sub Toolbar1_ButtonClick(ByVal Button As MSComctlLib.Button)

Select Case Button.Key

  Case "BtAbrir"
  MnuAbrir_Click
  Case "BtGuardar"
  MnuGuardarComo_Click
  Case "BtImprimir"
  MnuImprimir_Click
  Case "BtPresup."
  MnuPresupuesto_Click
  Case "BtEspacio"
  MnuEspacio_Click
  Case "BtPyE"
  MnuPyE_Click
  Case "BtWinQSB"
  MnuWinQSB_Click
  Case "BtCalc."
  MnuCalculadora_Click
  Case "BtDerive"
  MnuDerive_Click
  Case "BtAyuda"
  MnuTemasDeAyuda_Click

End Select
```

End Sub

1.7 Barra de Estado

La barra de estado es una ventana, generalmente ubicada en la parte inferior de un formulario, a través de la cual una aplicación puede mostrar información al usuario.

Se va a añadir una barra de estado al formulario MDI con ayuda del control *StatusBar*, es decir, hay que agregar el control al formulario MDI. El nombre de código del control es *StatusBar1*.

El control *StatusBar* consta de uno o más paneles (objetos **Panel**), hasta un máximo de 16, contenidos en una colección **Panels**.

Hay 2 estilos de barra de estado (propiedad *Style* de la ficha o configuración *General*), el normal (*sbrNormal*) y el simple (*sbrSimple*). El primero que es el predeterminado permite mostrar varios paneles, y el segundo sólo muestra un panel grande, por lo tanto se usará el primero.

Para agregar los paneles, primero se tiene que visualizar el diálogo *Página de propiedades* del control *StatusBar*. Se hace clic sobre el control utilizando el botón derecho del ratón y se elige la orden *Propiedades* del menú contextual que se visualiza. Después se siguen los pasos indicados a continuación:

1. Seleccionar la ficha *Paneles* (Figura 6-15).
2. Insertar uno a uno los 4 paneles que va a contener la barra de estado.
    - El primer panel es de propósito general, por lo que es un panel de texto en el que sólo hay que especificar la propiedad *Style* en *sbrText* para tal efecto. Su anchura es de 5000 *twips*.
    - El segundo panel indica el estado de la tecla *Bloqueo Mayúsculas*, por lo que se establece su propiedad *Style* en *sbrCaps*. Su anchura es de 1440 *twips*.
    - El tercero visualiza la hora (*Style: sbrTime*). Anchura de 1440 *twips*.
    - El último visualiza la fecha (*Style: sbrDate*). Achura de 1440 *twips*.

Todos los paneles tienen la propiedad *AutoSize* especificada en *SbrSpring*, lo que permite que ajusten su tamaño cuando el tamaño del contenedor (formulario MDI) cambia.

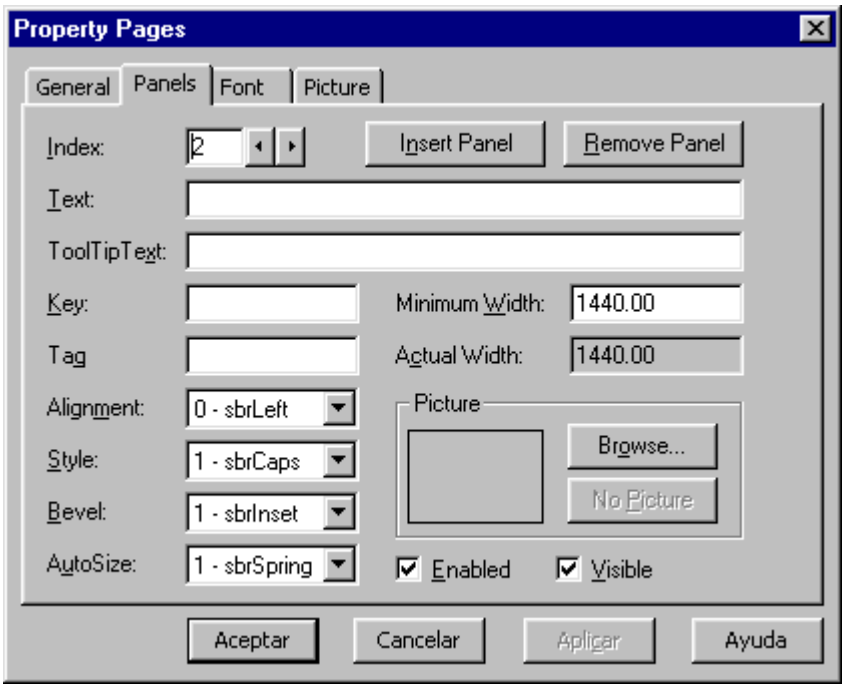


Figura 6-15
Página de propiedades del control *StatusBar* (ficha *Paneles*)

La barra de estado queda de la siguiente forma (Figura 6-16):

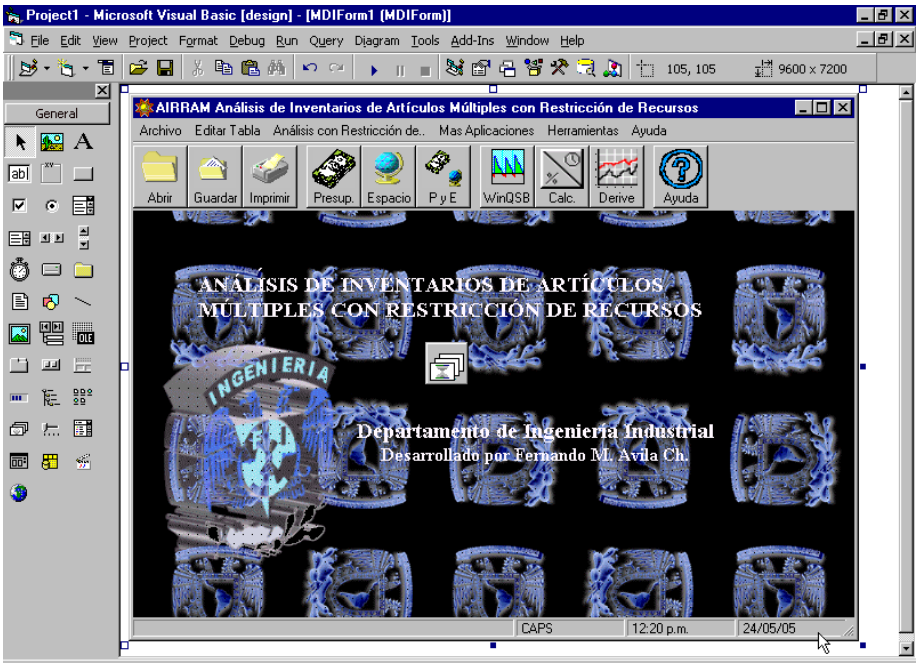


Figura 6-16
Barra de Estado

1.8 Enlace a la Facultad de Ingeniería

AIRRAM ha sido diseñado para cumplir con las necesidades de cátedra en la Facultad de Ingeniería, pero sería excelente si pudiese ser útil a cualquier usuario externo a la comunidad estudiantil de la Facultad. El medio que garantiza la más extensa distribución de cualquier aplicación es Internet, por lo que al finalizar la fase de diseño, AIRRAM será distribuido a través de este medio.

Teniendo en cuenta lo anterior, la aplicación se puede utilizar como un medio para que sus usuarios conozcan la Facultad de Ingeniería por medio de una animación que contenga una liga a la página electrónica de la escuela.

La animación estará ubicada en el extremo superior derecho del formulario MDI, en la barra de herramientas (será un escudo de la Facultad girando sobre si mismo).

Las imágenes que darán la impresión de movimiento son las siguientes:

Fueron modificadas con el editor de gráficos PAINT, con el objetivo de reducir y uniformizar su tamaño, e invertir sus colores.

Para mostrar y controlar la animación en el formulario MDI, será necesario incluir otras 2 clases de controles en la aplicación: *Caja de imagen* y *Temporizador*.

Un control *Caja de imagen* (**PictureBox**)  puede visualizar un gráfico a partir de un mapa de bits, un icono, un metarchivo, un metarchivo mejorado, un fichero **.jpeg* o un fichero **.gif*. La ventaja de la utilización de controles gráficos (como *PictureBox*) es que requieren menos recursos del sistema que otros controles de Visual Basic, lo que redunda en una mayor velocidad de ejecución.

El control *Temporizador* (**Timer**)  permite ejecutar acciones controladas por lapsos. La propiedad crítica y verdaderamente única del control *Timer* es *Interval*. Esta propiedad determina la frecuencia con la que se desea que este control genere eventos. En tiempo de diseño o de ejecución se asigna el número de milisegundos que tienen que pasar, antes de que el control *Timer* dispare un evento.

El procedimiento para formar la animación es:

1. Insertar 12 controles *PictureBox* en el lado derecho de la barra de herramientas. El nombre de código de cada control será *PictureN* (para N = 1....12).
2. Dimensionar el primer control *PictureBox* (*Picture1*) de acuerdo al tamaño de las imágenes que va a contener (escudos de la Facultad). Las dimensiones de los otros

11 controles *PictureBox* en realidad no importan, pero con el objetivo de que puedan ser insertados en la barra de herramientas hay que disminuir su tamaño al mínimo (en tiempo de ejecución sólo se observará el primer control *PictureBox*).

3. Vincular una a una las once imágenes de la animación a la propiedad *Picture* de cada uno de los controles *PictureBox* (desde el segundo control) . El control *Picture1* no va a estar ligado a alguna imagen en tiempo de diseño, ya que en tiempo de ejecución visualizará alternativamente a todas.
4. En la propiedad *ToolTipText* de *Picture1* se debe escribir: Enlace a la Facultad de Ingeniería. Para que cuando el ratón sea detenido sobre la animación se informe al usuario que esta es un enlace a la Facultad.
5. El evento doble clic del control *Picture1*, dispara la carga de la página electrónica de la escuela en el navegador *Microsoft Internet Explorer*. El procedimiento que responde al evento doble clic será programado posteriormente, ya que utilizará algunos conceptos que no se han revisado.
6. La animación estará ubicada a 300 *twips* del borde derecho de la interfaz principal. En consecuencia, es preciso que se ubique y reubique en esa posición cada vez que se cargue y redimensione el formulario MDI. Los procedimientos que permiten lo anterior son:

```
Private Sub MDIForm_Load()

Picture1.Left = MDIForm1.Width - 1275

End Sub
```

- Procedimiento privado (tipo *Sub*) *MDIForm_Load* conducido por el evento *Load* (carga) del Formulario MDI.[7]
- La propiedad *Left* del control *Picture1* define su posición horizontal (coordenada *x*).
- La propiedad *Width* del formulario MDI define su anchura.
- 1275 = Anchura de *Picture1* (975) + Distancia entre el borde derecho de *Picture1* y el borde derecho del formulario MDI (300).
- La coordenada *x* de *Picture1* siempre será igual a la anchura del formulario (*Width*) – 975 – 300. La coordenada *y* no se mueve.

Nota: Las cajas que se encuentran a la derecha del código suministran aclaraciones con respecto a él.

[7] Para mayor referencia con respecto a los tipos de Procedimientos en Visual Basic, dirigirse al capítulo 3 apartado 5.1.7 del presente documento.

```
Private Sub MDIForm_Resize()

Picture1.Left = MDIForm1.Width - 1275

End Sub
```

- Procedimiento privado (tipo *Sub*) *MDIForm_Resize* conducido por el evento *Resize* (redimensionamiento) del Formulario MDI.
- La propiedad *Left* del control *Picture1* define su posición horizontal (coordenada *x*).
- La propiedad *Width* del formulario MDI define su anchura.
- 1275 = Anchura de *Picture1* (975) + Distancia entre el borde derecho de *Picture1* y el borde derecho del formulario MDI (300).
- La coordenada *x* de *Picture1* siempre será igual a la anchura del formulario (*Width*) – 975 – 300. La coordenada *y* no se mueve.

7. Insertar un control *Timer* (su nombre de código es *Timer1*) en el formulario MDI.
8. Establecer su propiedad *Interval* (Figura 6-17) en 100 *milisegundos*, es decir, cada décima de segundo el código asociado a *Timer1* se repetirá.

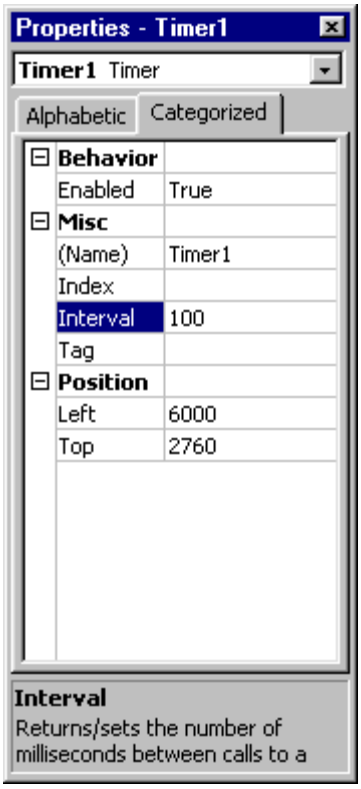


Figura 6-17
Ventana de propiedades del control *Timer1*

9. El procedimiento de control (en *Timer1*) para la animación es:

```
Private Sub Timer1_Timer()

Select Case Picture1.Picture
  Case Empty
  Picture1.Picture = Picture2.Picture
  Case Picture2.Picture
  Picture1.Picture = Picture3.Picture
  Case Picture3.Picture
  Picture1.Picture = Picture4.Picture
  Case Picture4.Picture
  Picture1.Picture = Picture5.Picture
  Case Picture5.Picture
  Picture1.Picture = Picture6.Picture
  Case Picture6.Picture
  Picture1.Picture = Picture7.Picture
  Case Picture7.Picture
  Picture1.Picture = Picture8.Picture
  Case Picture8.Picture
  Picture1.Picture = Picture9.Picture
  Case Picture9.Picture
  Picture1.Picture = Picture10.Picture
  Case Picture10.Picture
  Picture1.Picture = Picture11.Picture
  Case Picture11.Picture
  Picture1.Picture = Picture12.Picture
  Case Picture12.Picture
  Picture1.Picture = Picture2.Picture
End Select

End Sub
```

- Procedimiento privado (tipo *Sub*) *Timer1_Timer* conducido por el evento *Timer* del control *Timer1*.
- La sentencia de control *Select Case* interroga la propiedad *Picture* del control *Picture1* cada décima de segundo.
- Si *Picture1* está vacío, se le vincula la imagen que contiene *Picture2*, si *Picture1* contiene la imagen de *Picture2*, se le vincula la imagen que contiene *Picture3*, y así hasta llegar a *Picture12* cuando el ciclo se repite.
- Todo el ciclo tiene una duración de 1.1 segundos.

Hasta aquí llega el diseño de la interfaz principal de la aplicación, el resultado se muestra en la siguiente figura (Figura 6-18):

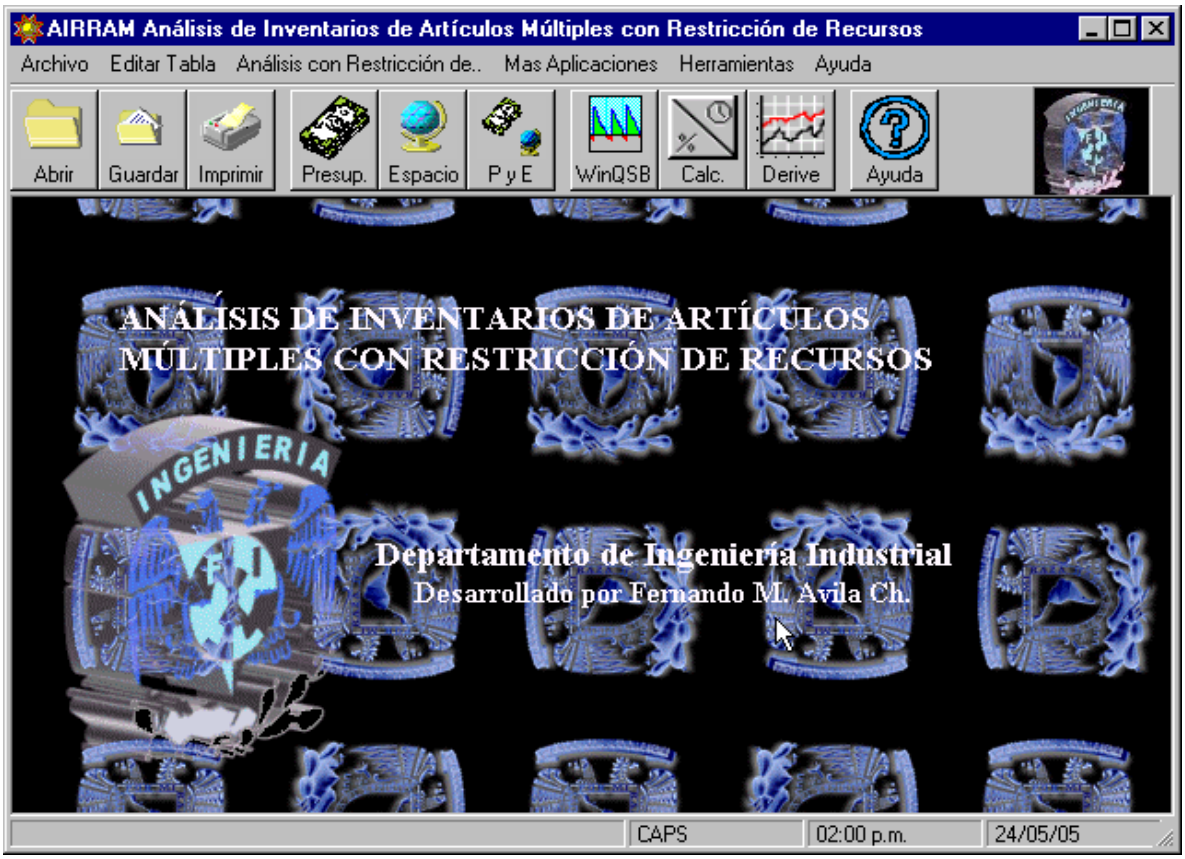


Figura 6-18
Interfaz principal de AIRRAM

2 ANÁLISIS CON LA RESTRICCIÓN DE PRESUPUESTO

En este punto es necesario recordar el objetivo central del presente documento, el cual es: "*desarrollar un software capaz de analizar el problema del modelo de inventarios de artículos múltiples con restricción de recursos (**Presupuesto** y Espacio)*, dado que ya se cuenta con la *Interfaz Principal de Usuario*, el paso lógico inmediato es el de comenzar el desarrollo del apartado que analice el problema bajo la restricción de Presupuesto.

Lo anterior justifica la afirmación de que aquí principia la parte cardinal de la tesis, puesto que se le empieza a dar respuesta a su objetivo central.

2.1 Diseño de la interfaz

Una de las preguntas clave a la que hay que responder cuando se analizan modelos de inventarios es: ¿Cuánto pedir?, ¿Cuánto ordenar?, ¿Cuántas unidades comprar de cada clase de artículo?, es decir, ¿Cuál será el tamaño de lote óptimo?, entendiendo como lote óptimo el que minimice los costos de inventario. Para responder esta pregunta se requiere que en una interfaz se proporcione la siguiente información (Datos de entrada):

- $i =$ costo total anual de mantener el inventario; que debe ser suministrado en porcentaje anual.
- $C =$ límite de la inversión total en inventario (restricción de presupuesto), suministrada en $.
- $n =$ número de clases de artículos en el sistema.
- $A_i =$ costo de ordenar, es decir, todos los costos en que se incurre cuando se lanza una orden de compra (el costo de ordenar es independiente de la cantidad que se compra), suministrado en $. Para toda $i = 1,2,.....n$.
- $D_i =$ demanda por unidad de tiempo, suministrada en unidades por año. Para toda $i = 1,2,.....n$.
- $c_i =$ costo unitario para cada clase de artículo, suministrado en $ por unidad. Para toda $i = 1,2,.....n$.

Ya que se tienen los datos de entrada es factible responder a la pregunta inicial (y algunas más), desplegando los siguientes resultados (información de salida):

- $Q_i^* =$ tamaño de lote óptimo (cantidad económica a ordenar) para cada clase de artículo, desplegado en unidades. Para toda $i = 1,2,.....n$.
- $\lambda =$ lambda, multiplicador de Lagrange, es adimensional.
- $K(Q) =$ costo total anual promedio del inventario, desplegado en $.

Con los *Datos de entrada y la Información de salida*[8], ya es posible diseñar gráficamente la Interfaz de Usuario para el análisis con la restricción de presupuesto.

El primer paso del diseño es insertar un *formulario* del tipo *SDI* (*Single Document Interface*) a la aplicación, debido a que será una *ventana de documento*, es decir, será una *ventana hija* de la interfaz principal de usuario.

Durante la ejecución, los formularios hijo se muestran dentro del área de trabajo del formulario padre. Cuando se minimiza un formulario hijo su icono aparece en el fondo del formulario padre en lugar de aparecer en la barra de tareas, y cuando se maximiza su título se combina con el título del formulario MDI visualizándose ambos en la barra de título del formulario padre.

Para agregar el formulario hijo a la aplicación, hay que ejecutar la orden *Agregar formulario* del menú *Proyecto*, y asignar a su propiedad **MDIChild** el valor **True**, de otra manera el formulario SDI se comportaría de manera independiente al formulario MDI, es decir, no sería un formulario hijo, sino un formulario estándar que puede aparecer fuera de los límites del formulario padre. El nombre de código del primer formulario hijo (interfaz de usuario para el análisis con la restricción de presupuesto) es *Form1*.

[8] Para mayor referencia con respecto a los *Datos de Entrada y la Información de Salida* dirigirse a los capítulos 1 apartado 5.1 y 2 apartado 1.2 de este documento.

En la *ventana de propiedades* de *Form1* se deben de asignar algunas propiedades de la siguiente forma:

- ***Caption***: *Análisis con la Restricción de Presupuesto*. Para que el nombre de la interfaz aparezca en su barra de título.
- ***BorderStyle***: *3 – Fixed Dialog*. Así la ventana no se podrá maximizar, minimizar o cambiar de tamaño.
- ***BackColor***: *White*. El área de trabajo será blanca.
- ***Icon***: Hay que vincularle el icono característico de AIRRAM (estrella amarilla de picos naranja).
- ***Height***: *3405*. Establece el alto de la ventana en 3405 *twips*.
- ***Width***: *8265*. Establece el ancho de la ventana en 8265 *twips*.

En la Figura 6-19 se muestra la apariencia de *Form1* hasta este momento.

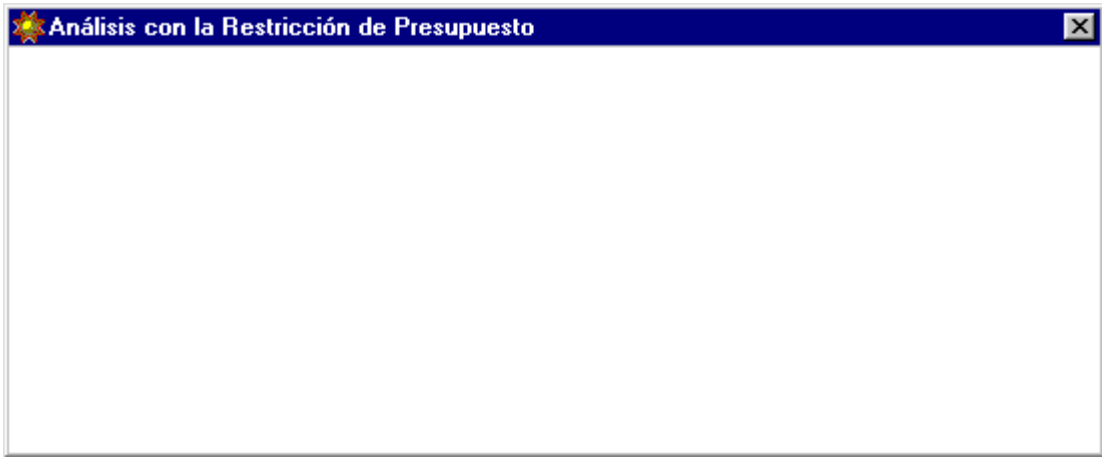


Figura 6-19
Apariencia intermedia de *Form1*

Los objetos gráficos que permiten introducir o extraer datos en una aplicación Visual Basic son llamados *Controles*. El formulario más los controles forman la interfaz o medio de comunicación.

Para completar el diseño de *Form1*, hay que insertar los siguientes controles en el área de trabajo:

- Etiqueta (*Label*).  El control *Label* exhibe texto. En tiempo de ejecución el usuario no puede alterar el texto que aparece en una etiqueta.
- Caja de Texto (*TextBox*).  El control *TextBox* se utiliza cuando se quiere que el usuario escriba algo, como una respuesta a una petición, o cuando se necesita información.

- Marco (*Frame*).  Un control *Frame* permite identificar fácilmente un grupo de controles. Se utiliza para efectos de orden y cuando se quiere realzar el aspecto de un formulario.
- Botón de Comando (*CommandButton*).  Casi todo formulario contiene un botón de comando. Es la herramienta usual cuando se quiere disparar un acción. Al hacer clic en un botón de comando, el usuario puede indicar que una respuesta esta lista, que una impresora tiene papel o que es el momento de finalizar el programa.

Atención especial merece el último control de la lista, el control **Cuadrícula** (*MSFlexGrid*)  que permite visualizar información distribuida en filas y columnas.

El control *MSFlexGrid* visualiza la información en celdas. El usuario puede situarse en una celda cualquiera haciendo clic sobre ella o utilizando las teclas de movimiento del cursor. Durante la ejecución, el usuario puede seleccionar una o más celdas, pero no puede modificar su contenido escribiendo directamente sobre ellas. La forma de modificar una celda es ejecutando el código apropiado.

El control *MSFlexGrid* es un control *ActiveX* , y por lo tanto no se encuentra dentro de los controles intrínsecos incluidos en la *caja de herramientas* de Visual Basic. Para añadir este control hay que ejecutar la orden *Componentes* del menú *Proyecto* y añadir el componente *ActiveX* denominado *Microsoft FlexGrid Control 6.0*.

De alguna manera *MSFlexGrid* es inusual, debido a que varias de sus propiedades deben de asignarse en tiempo de ejecución, ya que de no hacerlo el control no sería funcional.

El procedimiento para completar el diseño de *Form1* es (cada control hay que dimensionarlo y colocarlo de manera adecuada en el formulario):

1. Insertar un par de controles *Frame* (sus nombres de código serán *Frame1* y *Frame2*). Establecer en ambos la propiedad ***BackColor:*** *White*, así tendrán el mismo color del área de trabajo. El título del primero será *Datos* y el del segundo *Resultados*, por lo que sus propiedades *Caption* se deben de establecer ***Frame1.Caption:*** *Datos* y ***Frame2.Caption:*** *Resultados*.
2. Insertar 5 controles *TextBox* (con nombres de código *TextN* para N = 1....5). Tres en el *Frame1(Datos)* y dos en el *Frame2 (Resultados)*. La propiedad *Text* (que define el contenido del control) de cada uno se debe vaciar, ya que servirán para introducir y desplegar información. Cada control *TextBox* se usará para introducir y desplegar los valores de los *Datos de entrada y la Información de salida* distribuidos de la siguiente forma:

    - En *Text1* se introducirá el valor de *i*.
    - En *Text2* se introducirá el valor de *C*.
    - En *Text3* se introducirá el valor de *n*.

    } *Frame de Datos*

- En *Text4* se desplegará el valor de λ.
- En *Text5* se desplegará el valor de *K(Q)*.

} *Frame de Resultados*

3. Con el propósito de que el usuario final conozca la naturaleza de los datos que va a proporcionar y a obtener en la interfaz (*Form1*), hay que etiquetar los controles *TextBox* que se acaban de introducir de acuerdo a la estructura anterior, por lo que hay que insertar 6 controles *Label* (con nombres de código *LabelN* para N = 1....6) en el *Frame1* y otros 3 controles *Label* (con nombre de código *LabelN* para N=7....9) en el *Frame2*. En todos los controles *Label* se debe establecer su propiedad ***BackColor:*** *White*, para que tengan el mismo color del área de trabajo. En la propiedad *Caption* de cada control *Label*, se debe escribir el símbolo de cada dato que va a ser introducido o desplegado en los controles *TextBox*, y las unidades en las que se expresa el dato.
4. Colocar un botón de comando (con nombre de código *Command1*) y modificar su propiedad *Caption* para que despliegue la leyenda *Analizar*. Al hacer clic en *Command1* se va a disparar el análisis en la interfaz.
5. Colocar un segundo botón de comando (con nombre de código *Command2*) y modificar su propiedad *Caption* para que despliegue la leyenda *Salir*. Al hacer clic en *Command2* se va a descargar la interfaz.
6. Por último, introducir un control *Cuadrícula* (con nombre de código *MSFlexGrid1*), con el propósito de suministrar los datos *A, D* y *c* para cada clase de artículo, y desplegar su tamaño de lote óptimo *Q**. Propiedades como el número de renglones, las leyendas de los renglones y columnas fijas, etc. Se deben de especificar por código en tiempo de ejecución. Las propiedades que se pueden establecer en tiempo de diseño para la cuadrícula son:

    - Número de columnas (***Cols:*** *5*).
    - Renglones Fijos (***Fixed Rows:*** 1) y Columnas Fijas (***Fixed Cols:*** *1*), que son los renglones y columnas que sirven para mostrar los títulos. Las propiedades se pueden establecer en la *ventana o página de propiedades* del control.

La apariencia del control *Cuadrícula* (*MSFlexGrid1*) en tiempo de ejecución va a diferir de su apariencia en tiempo de diseño, y esto se debe a que todavía se tienen que especificar ciertas de sus propiedades mediante código. La apariencia final de la interfaz de usuario para el análisis con la restricción de presupuesto (*Form1*) se muestra en la Figura 6-20.

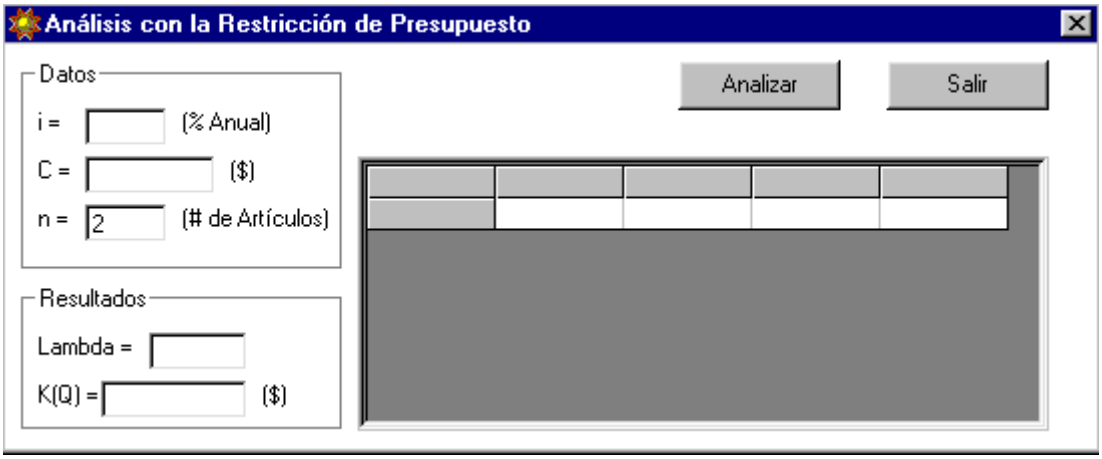


Figura 6-20
Apariencia final de *Form1* en tiempo de diseño

2.2 Funcionalidad de la interfaz

El medio del que se sirve Visual Basic para depositar el código de cualquier aplicación, es una ventana llamada ***Módulo***, es decir, el código de una aplicación Visual Basic se edita en módulos. Los módulos tienen incorporadas funciones automáticas de formato y comprobación de sintaxis. Hay 3 tipos de módulos: de formulario, estándar y de clase (en esta aplicación sólo se utilizarán los 2 primeros). Cada uno de ellos puede contener declaraciones y procedimientos. Un formulario (**.frm*) incluye controles más código y un módulo (**.bas*) o una clase (**.cls*) incluyen sólo código.

Antes de programar los procedimientos que harán funcional a la interfaz, hay que declarar ciertas variables que serán comunes a lo largo del desarrollo. En Visual Basic no es necesario declarar una variable antes de utilizarla. Sin embargo, esta forma de trabajar puede ser una fuente de errores. Cuando se declaran variables hay que tener en cuenta 2 características fundamentales: el tipo de datos que pueden almacenar, y el ámbito o alcance de las mismas.

Para empezar, se van a utilizar 2 tipos de variables:

- *Variant*. Que es el tipo de dato por omisión, es decir, cuando se declara una variable y no se especifica su tipo, se asume que es de tipo *Variant*. Puede almacenar números o caracteres.
- *Double*. Almacena sólo números de punto flotante con precisión doble (8 bytes).

Se entiende como ámbito de una variable el espacio de la aplicación donde la variable es visible y por lo tanto se puede utilizar. Existen 3 tipos de ámbito:

- *Local*. Una *variable local* se reconoce solamente en el procedimiento en el que está definida. Fuera de este procedimiento, la variable no es conocida. Su utilización

más común es intervenir en cálculos intermedios. La forma más habitual de declararla es con la sentencia ***Dim*** dentro del procedimiento.

- *Módulo*. Una *variable declarada a nivel de módulo* (formulario, módulo estándar o clase) puede ser compartida por todos los procedimientos de ese módulo. Se declara con las sentencias ***Dim*** o ***Private*** en la sección de declaraciones del módulo (sección *General*). Para editar esta sección, hay que abrir la *ventana de código del formulario*, de un módulo estándar o de una clase. Después seleccionar "(*General*)" de la lista de la izquierda, lista de objetos y "(*Declaraciones*)" de la lista de la derecha, lista de procedimientos.
- *Global*. Una *variable global* es una variable declarada a nivel de módulo pero que puede ser accedida desde cualquier otro módulo. Se declara con la sentencia ***Public*** en la sección de declaraciones del módulo.

Se deben declarar las variables *n, i, C* y *Cs* como variables globales, porque serán utilizadas en varios módulos y procedimientos de la aplicación. Las tres primeras variables representan a los datos de entrada definidos en el apartado 2.1 del presente capítulo. Y *Cs* representa la inversión total en el inventario, es decir, $Cs = \sum_{i=1}^{n} c_i Q_i$.[9]

Las variables *C* y *Cs* deben ser del tipo *Double* (precisión doble), porque se van a comparar sus valores, y sólo manejando valores de gran precisión se puede encontrar alguna diferencia entre 2 valores que sean muy similares. Las variables *n* e *i* pueden ser del tipo *Variant*.

La siguiente línea de código declara las variables descritas en los 2 últimos párrafos.

```
Public n, i, C As Double, Cs As Double
```

Para que la interfaz (*Form1*) sea completamente funcional, se deben programar algunos procedimientos conducidos por ciertos eventos del formulario. El primero de ellos es el procedimiento *Text3_Change* del control *Text3*, que es utilizado para introducir el número de clases de artículos (*n*). El evento *Change* (*Cambio*) ocurre cada vez que un usuario intenta cambiar el contenido de *Text3*. Cuando esto ocurre, se activa el siguiente procedimiento conducido por *Change*.

[9] Para mayor referencia dirigirse a el capítulo 2 apartado 1.2 de este documento.

```
Private Sub Text3_Change()

On Error GoTo ManipularError

n = Text3.Text
```

- Procedimiento privado (tipo *Sub*) *Text3_Change* conducido por el evento *Change* (cambio) del control *Text3*.
- La sentencia *On Error GoTo* intercepta cualquier tipo de error que se produzca en el procedimiento. Y la etiqueta *ManipularError* identifica la primera línea de la rutina que manipula el error.
- Asignación del contenido de *Text3* a la variable global *n*.

```
If n <> "" And n > 0 And IsNumeric(n) And (n <= 10000) Then
   If (n / CInt(n)) = 1 Then
   MSFlexGrid1.Rows = n + 1
   Else
   MSFlexGrid1.Rows = 1
   End If
Else
MSFlexGrid1.Rows = 1
End If
```

- Esta rutina permite fijar la cantidad de renglones de la cuadrícula (*MSFlexGrid1*), de acuerdo al número de clases de artículos (*n*) que se introduce en la caja de texto (*Text3*).
- La sentencia de control *if* anidada cuestiona si el número de clases de artículos (*n*) es diferente de nulo, mayor que cero, si es un número, menor o igual a 10 000 y si es entero. En caso afirmativo, el número de renglones (propiedad *Rows*) de la cuadrícula *MSFlexGrid1* será igual a *n+1*, en caso contrario, sólo se desplegará un renglón (el de los títulos).
- La función intrínseca[10] *IsNumeric()*, determina si su argumento contiene un número (o si los datos pueden convertirse en un número válido).
- La función de conversión de tipo de dato *CInt()* convierte su argumento al tipo de dato *Integer* (*entero*). Si a la función no se le pasa como argumento un número produce un error, es por ello que se encuentra en un *if* anidado.
- El cociente *n/CInt(n)* es igual a 1 cuando *n* es un entero.

[10] Visual Basic usa 2 tipos de funciones: externas (*DLL´s*) e internas. Las internas a su vez se dividen en las que son programadas por el desarrollador, y las que provee Visual Basic (*Intrínsecas*). Para mayor referencia con respecto al tópico *Funciones*, dirigirse a el capítulo 3 apartado 5.1.7.3 de este documento.

```
With MSFlexGrid1

  .Row = 0
  For i = 0 To 4
    .Col = i
    Select Case i
      Case 0: .Text = "Artículo"
      Case 1: .Text = "A($)"
      Case 2: .Text = "D(Uni/año)"
      Case 3: .Text = "c($)"
      Case 4: .Text = "Q*(Uni)"
    End Select
  Next i
```

- Con esta rutina se etiqueta el nombre de cada tipo de dato (*No. De clase de Artículo, A, D, c* y *Q**)que se va a introducir y desplegar en cada columna de *MSFlexGrid1*, en el renglón fijo destinado para tal propósito (*.Row = 0*).
- La instrucción *With...End With* permite asignar fácilmente varias propiedades para un solo objeto (en este caso *MSFlexGrid1*). Cuando se encierra el control *MSFlexGrid1* dentro de *With...End With* se puede eliminar la repetición del nombre del control.
- Se establece el renglón de la celda que se quiere modificar en el renglón de los títulos (propiedad *.Row = 0*). La sentencia de control *For* alterna la columna seleccionada desde la primera hasta la última (propiedad *.Col = i* para *i = 0.....4*), mientras *Select Case* en cada ciclo *For* determina cual de las columnas está seleccionada para etiquetarle su respectivo título (propiedad *.Text*).

```
If n <> "" And n > 0 And IsNumeric(n) And (n <= 10000) Then
  If (n / CInt(n)) = 1 Then
  Number = 0
  .Col = 0
  For i = 1 To n
    .Row = i
    Number = Number + 1
      .Text = Number
  Next i
  Else

  End If
Else

End If

End With
```

- Con esta rutina se etiqueta el número de clase de artículo al que corresponde cada renglón de *MSFlexGrid1*, en la columna fija destinada para tal propósito (*.Col = 0*).
- Si se cumplen las mismas condiciones para *n* que al principio del procedimiento, la sentencia *if* anidada permite inicializar el contador *Number* en cero, y establecer a la primera columna de *MSFlexGrid1*, como la columna que se va a modificar (propiedad *.Col = 0*). La sentencia *For* alterna el renglón seleccionado desde el *1* hasta el *n* (propiedad *.Row = i* para *i = 1.....n*), en cada ciclo *For* el contador *Number* incrementa su valor en 1 unidad, y etiqueta este valor en la celda en uso de la primera columna (propiedad *.Text*).

```
ManipularError:
   If Err.Number = 0 Then
   Exit Sub
   Else
   MsgBox "Dato inválido:" & Err.Description, 16
   Err.Clear
   Exit Sub
   End If

End Sub
```

- Cuando se produce un error, el error se intercepta con la sentencia *On Error GoTo* y se manipula en esta rutina.

- En Visual Basic se encapsula la información relativa a un error de ejecución en el objeto ***Err***, donde cada tipo de error tiene un número que lo identifica (propiedad *Number*). Si no se presenta error alguno en tiempo de ejecución la propiedad *Number* del objeto *Err* es igual a cero, es por esto que aunque en la ejecución no se presenten contratiempos siempre se va a ejecutar la rutina *ManipularError*.
- Si no se presentan errores en tiempo de ejecución (*Err.Number = 0*), se finaliza el procedimiento (sentencia *Exit Sub*). En caso contrario, la función intrínseca *MsgBox* visualiza una caja de diálogo del tipo 16, es decir, que tiene que ser cerrada para poder continuar con el procedimiento (*Modal*) pulsando el botón aceptar. La caja de diálogo va a desplegar el mensaje *Dato inválido:* concatenado con la descripción del error que se produjo (propiedad *Description* del objeto *Err*), asimismo, después se van a iniciar las propiedades del objeto *Err* a cero o nulo, usando el método *Clear*, finalizando el procedimiento con la sentencia *Exit Sub*.

El procedimiento anterior sirvió para definir el número de renglones de *MSFlexGrid1* (y sus correspondientes etiquetas) en función del número de clases de artículos (*n*) que proporciona el usuario, además de etiquetar el nombre de cada tipo de dato (en el renglón de los títulos) que se va a introducir y desplegar en *MSFlexGrid1*. Todo esto sucede cada vez que el usuario final cambia el contenido de la caja de texto (*Text3*) que define a *n*, luego entonces, es necesario implementar un procedimiento similar cuando el usuario carga o llama a la interfaz (*Form1*). Tal procedimiento es el presentado a continuación, denominado *Form_Load* y conducido por el evento *Load* (carga) del formulario (*Form1*).

```
Private Sub Form_Load()

n = Text3.Text

MSFlexGrid1.Rows = n + 1
```

- Procedimiento privado (tipo *Sub*) *Form_Load* conducido por el evento *Load* (carga) del formulario (*Form1*).
- Asignación del contenido de *Text3* a la variable global *n*. La propiedad *Text* del control *Text3* tiene asignada el número 2 como valor por omisión, es decir, al momento de cargar el formulario se va a visualizar el renglón de los títulos y 2 renglones más no fijos (*n* = 2). El considerar que se tienen 2 clases de artículos al momento de la carga de *Form1* es completamente arbitrario.
- Con la tercera sentencia se establece el número de renglones de *MSFlexGrid1* en *n + 1*, es decir, 2 renglones no fijos (uno para cada clase de artículo) y 1 fijo para los títulos.

```
With MSFlexGrid1

  .Row = 0
  For i = 0 To 4
    .Col = i
    Select Case i
      Case 0: .Text = "Artículo"
      Case 1: .Text = "A($)"
      Case 2: .Text = "D(Uni/año)"
      Case 3: .Text = "c($)"
      Case 4: .Text = "Q*(Uni)"
    End Select
  Next i
```

- Con esta rutina se etiqueta el nombre de cada tipo de dato (*No. De clase de Artículo, A, D, c* y *Q**)que se va a introducir y desplegar en cada columna de *MSFlexGrid1*, en el renglón fijo destinado para tal propósito (*.Row = 0*).
- Si se requiere consultar la explicación de esta rutina se puede dirigir a la rutina correspondiente del procedimiento anterior (ambas son idénticas).

```
  Number = 0
  .Col = 0
  For i = 1 To n
    .Row = i
    Number = Number + 1
      .Text = Number
  Next i
```

- Con esta rutina se etiqueta el número de clase de artículo al que corresponde cada renglón de *MSFlexGrid1*, en la columna fija destinada para tal propósito (*.Col = 0*).
- Si se requiere consultar la explicación de esta rutina se puede dirigir a la rutina correspondiente del procedimiento anterior (a excepción de las condiciones para *n* ambas son idénticas).

```
.Row = 1
.Col = 1

End With

End Sub
```

- Si no se establece explícitamente la celda que estará seleccionada de *MSFlexGrid1* al momento de la carga de *Form1*, esta será por omisión la definida por las coordenadas (*.Row = 0 , .Col = 0*), la cual es una celda fija que se usa para desplegar un título.
Cualquier celda seleccionada es una celda que se puede modificar, y esto no es conveniente en el caso de la celda (*.Row = 0 , .Col = 0*) debido a lo expuesto en el párrafo anterior, por lo que con la sentencia de la izquierda la celda seleccionada al momento de la carga de *Form1* será la celda no fija definida por las coordenadas (*.Row = 1 , .Col = 1*).

Antes de continuar, es conveniente definir algunas funciones de cadena que serán utilizadas en procedimientos subsecuentes. Las *funciones de cadena* son funciones intrínsecas que reciben por argumento una cadena y/o devuelven una cadena, es decir, reciben por argumento caracteres y/o devuelven caracteres, entre las más significativas se encuentran las siguientes:

- *Len ()*. Devuelve la cantidad de bytes de memoria necesarios para contener su argumento, es decir, devuelve la longitud (cantidad de caracteres) de la variable, constante o expresión dentro de sus paréntesis.
- *Chr()*. Se utiliza para convertir de valores ASCII numéricos a sus correspondientes cadenas. La tabla ASCII (*American Standard Code for Information Interchange*) hace una lista con todos los posibles caracteres disponibles en la **PC** y asigna un número secuencial (un *código* ASCII) a cada carácter. Al poner un número dentro de los paréntesis de *Chr()*, puede producir el carácter que corresponde a ese número en la tabla ASCII.
- *Left()*. Es una función que devuelve secciones de una cadena, específicamente devuelve los caracteres solicitados de la izquierda de una cadena.

Con el propósito de que la interfaz para el análisis con la restricción de presupuesto sea completamente funcional, es necesario añadir un par de procedimientos más en este apartado, los cuales van a permitir la edición de la cuadrícula *MSFlexGrid1* (sin el código adecuado un control cuadrícula sólo sirve para desplegar datos). El primer procedimiento va a permitir que se puedan introducir caracteres (números) en las celdas de *MSFlexGrid1*. Cuando el usuario *"presione"* algún carácter del teclado del ordenador mientras se encuentra seleccionada cualquier celda de *MSFlexGrid1*, se va a disparar el procedimiento *MSFlexGrid1_KeyPress* conducido por el evento *KeyPress* (tecla presionada) del control *MSFlexGrid1*. Tal procedimiento es:

```
Private Sub MSFlexGrid1_KeyPress(KeyAscii As Integer)
```

- Procedimiento privado (tipo *Sub*) *MSFlexGrid1_KeyPress* conducido por el evento *KeyPress* (tecla presionada) del control *MSFlexGrid1*. Al procedimiento se le pasa por *referencia*[11] la variable de tipo *Integer*[12] (entera) *KeyAscii*, es decir, se le pasa un número entero.

```
If KeyAscii >= 46 And KeyAscii <= 57 Then
MSFlexGrid1.Text = MSFlexGrid1.Text & Chr(KeyAscii)
End If

End Sub
```

- Cuando se presiona algún carácter del teclado mientras se tiene seleccionada cualquier celda de *MSFlexGrid1*, se modifica el valor de la variable *KeyAscii* y se pasa por referencia a este procedimiento. La variable *KeyAscii* almacena un número entero (código *ASCII*) que representa al carácter tecleado en el ordenador.
- La sentencia de control *If* inspecciona si *KeyAscii* tiene un código *ASCII* almacenado (número entero) entre 46 y 57, lo que indicaría que se presionó una tecla que representa un número entre *0* y *9*, o la tecla del símbolo "*punto"* en la computadora. En caso de que se cumpla la condición anterior, el contenido de la celda seleccionada (propiedad .*Text*) de *MSFlexGrid1* al momento de presionar la tecla, será igual a su contenido previo concatenado (*&*) con el carácter equivalente (número entre *0* y *9* o *"punto"*) del código *ASCII* (número entero) almacenado en la variable *KeyAscii*.
- La función intrínseca *Chr()* es la que permite transformar el código *ASCII* (número entero) almacenado en la variable *KeyAscii* en su carácter equivalente (número entre *0* y *9* o *"punto"*).

[11] Para mayor referencia con respecto a el paso de variables a procedimientos, dirigirse a el capítulo 3 apartado 5.1.7.7 de este documento.
[12] Una variable de tipo *Integer* almacena números enteros con una precisión de 2 bytes.

El último procedimiento de este apartado va a permitir eliminar el contenido de la celda en uso de *MSFlexGrid1*, ya sea bien eliminando por completo dicho contenido, o un carácter a la vez. Cuando el usuario *"presione y suelte"* alguna tecla de *función* del ordenador mientras se encuentra seleccionada cualquier celda de *MSFlexGrid1*, se va a disparar el procedimiento *MSFlexGrid1_KeyUp* conducido por el evento *KeyUp* (tecla presionada y soltada) de *MSFlexGrid1*. Tal procedimiento es:

```
Private Sub MSFlexGrid1_KeyUp(KeyCode As Integer, Shift As Integer)

Select Case KeyCode
  Case vbKeyDelete
    MSFlexGrid1.Text = ""
  Case vbKeyBack
    If Len(MSFlexGrid1.Text) > 0 Then
      MSFlexGrid1.Text = Left(MSFlexGrid1.Text, Len(MSFlexGrid1.Text) - 1)
    End If
End Select


End Sub
```

- Procedimiento privado (tipo *Sub*) *MSFlexGrid1_KeyUp* conducido por el evento *KeyUp* (tecla presionada y soltada) del control *MSFlexGrid1*. Al procedimiento se le pasan por *referencia* las variables de tipo *Integer* (entera) *KeyCode* y *Shift*. Un procedimiento conducido por un evento no se crea, sólo se edita, puede tener variables declaradas que se llegan a utilizar o "no" en el cuerpo del procedimiento (rutina), ya que el programador no crea el *esqueleto* del procedimiento (ya fue creado). Tal es el caso de la variable *Shift*, que aunque esta declarada en la *cabecera* de *MSFlexGrid1_KeyUp* no se utiliza.[13]

- El teclado de una PC convencional cuenta con 2 tipos de botones: de *carácter* y de *función*. Los botones de carácter son los que representan letras, números y símbolos especiales, mientras que los de función realizan alguna acción en específico al ser presionados. El procedimiento previo es disparado por botones de carácter, mientras que este es disparado por botones de función, como lo son: *enter* (entrar), *backspace* (borrar o retroceder un espacio), *delete* (suprimir), etc.
- Visual Basic proporciona un par de formas de identificar a los botones de función: por su número, o por su nombre de código. En esta rutina por facilidad didáctica para identificar que botón de función ha sido presionado se emplean nombres de código, aunque en realidad la variable *KeyCode* almacena números (números enteros con precisión de 2 bytes: tipo *Integer*) como identificadores.

[13] El *esqueleto* de un procedimiento lo constituyen su *cabecera* y *cierre* (sentencia *End Sub*). Y una *cabecera* es la primera línea de cualquier procedimiento, la cual lo define, indica su ámbito, tipo, y nombre, además de que puede tener o no ciertas variables declaradas.

- Cuando se presiona y suelta algún botón de función del teclado mientras se tiene seleccionada cualquier celda de *MSFlexGrid1*, se modifica el valor de la variable *KeyCode* y se pasa por referencia a este procedimiento. La variable *KeyCode* almacena un número entero que representa al botón de función utilizado.
- La sentencia de control *Case* inspecciona el contenido de la variable *KeyCode*.
- En caso de que el contenido de *KeyCode* indique que se presionó y soltó la tecla *Delete* o *Suprimir* (mediante el nombre de código *vbKeyDelete*) se borrará el contenido completo de la celda en uso (propiedad .*Text*) de *MSFlexGrid1*.
- En caso de que *KeyCode* indique que se presionó y soltó la tecla *BackSpace* o *Borrar o retroceder un espacio* (mediante el nombre de código *vbKeyBack*) se ejecutará una sentencia de control *If* que realiza lo siguiente: Con la ayuda de la función de cadena *Len()* se inspeccionará si la celda en uso de *MSFlexGrid1* no está vacía, si se cumple esa condición la función *Left()* devolverá la cadena izquierda de la celda en uso, constituida por todos sus caracteres originales (función *Len()*) menos uno, y asignará esa nueva cadena a la celda seleccionada (propiedad .*Text*). Es decir, cada vez que se presione y suelte el botón *BackSpace* se borrará un carácter (el último de izquierda a derecha) de la celda en uso de *MSFlexGrid1*, si es que la celda no está vacía.

Hasta aquí llega el apartado que permite hacer funcional a la interfaz de usuario para el análisis con la restricción de presupuesto (*Form1*), más en específico con este apartado se permitió hacer funcional al control cuadrícula de *Form1* (*MSFlexGrid1*).

La apariencia final de *Form1* en tiempo de ejecución se muestra en la Figura 6-21.

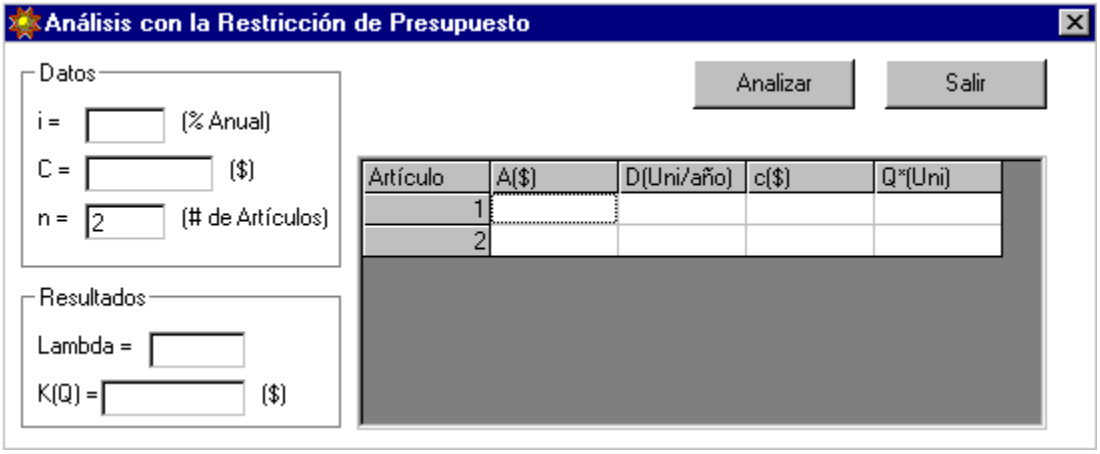


Figura 6-21
Apariencia final de *Form1* en tiempo de ejecución

2.3 Programación de los algoritmos de solución

En los apartados 1.2 y 2.1 del capítulo 2 se presentaron los algoritmos matemáticos de solución (*método de Multiplicadores de Lagrange modificado*) para el modelo de inventarios de artículos múltiples con restricción de recursos (***Presupuesto***), es ahora cuando se van a programar dichos algoritmos dando respuesta a la pregunta fundamental del modelo en estudio: ¿Cuál es la cantidad óptima de productos a ordenar (Q^*) ?

Hace algunas páginas, en el apartado 2.1 del presente capítulo se recordaron las variables que intervienen en el modelo en estudio, de aquí en adelante se van a utilizar intensivamente estas variables con algunos cambios en su nomenclatura para facilitar su programación. A continuación se presentan dichos cambios:

- $i =$ No sufre cambio su nomenclatura. Es el costo total anual de mantener el inventario; que debe ser suministrado en porcentaje anual.
- $C =$ No sufre cambio su nomenclatura. Es el límite de la inversión total en inventario (restricción de presupuesto), suministrada en $.
- $n =$ No sufre cambio su nomenclatura. Es el número de clases de artículos en el sistema.
- $A_i =$ No sufre cambio su nomenclatura. Es el costo de ordenar, es decir, todos los costos en que se incurre cuando se lanza una orden de compra (el costo de ordenar es independiente de la cantidad que se compra), suministrado en $. Para toda $i = 1,2,.....n$.
- $D_i =$ No sufre cambio su nomenclatura. Es la demanda por unidad de tiempo, suministrada en unidades por año. Para toda $i = 1,2,.....n$.
- $c_i =$ Su nomenclatura equivalente será *Cm*. Es el costo unitario para cada clase de artículo, suministrado en $ por unidad. Para toda $i = 1,2,.....n$.

- Q_i^* = Su nomenclatura equivalente será *Qop*. Es el tamaño de lote óptimo (cantidad económica a ordenar) para cada clase de artículo, desplegado en unidades. Para toda $i = 1,2,.....n$.
- λ = Su nomenclatura equivalente será *Lambda*. Es el multiplicador de Lagrange, es adimensional.
- $K(Q)$ = Su nomenclatura equivalente será *KT*. Es el costo total anual promedio del inventario, desplegado en $.
- *Cs* = No sufre cambio su nomenclatura. Es la inversión total en el inventario, es decir, $Cs = \sum_{i=1}^{n} c_i Q_i$.

Cualquier software de cierta complejidad tiene que desarrollarse bajo el concepto de *"Programación Estructurada"*, es decir, organizar su código en procedimientos principales y subprocedimientos con el objetivo de dar cierta coherencia a los algoritmos de solución, facilitar su comprensión y permitir reutilizar el código. El siguiente es un procedimiento principal, desde el cual se dispararán cada uno de los pasos (codificados en subprocedimientos) del método *de Multiplicadores de Lagrange* modificado para el modelo en estudio.

Al hacer *clic* en el botón de comando (*Command1*) que tiene la etiqueta *Analizar*, se va a desencadenar el procedimiento principal. Tal procedimiento lleva el nombre de *Command1_Click*, y es conducido por el evento *Click* de *Command1*. Se presenta a continuación:

```
Private Sub Command1_Click()

On Error GoTo ManipularError

i = Text1.Text
CP = Text2.Text
n = Text3.Text
```

- Procedimiento privado (tipo *Sub*) *Command1_Click* conducido por el evento *Click* del control *Command1*.
- La sentencia *On Error GoTo* intercepta cualquier tipo de error que se produzca en el procedimiento. Y la etiqueta *ManipularError* identifica la primera línea de la rutina que manipula el error.
- Asignación del contenido de *Text1* a la variable global *i*.
- Asignación del contenido de *Text2* a la variable *intermedia CP*. Una *variable intermedia* regularmente no se declara explícitamente, ya que sólo se emplea en el procedimiento en que fue creada (ámbito local), además de que Visual Basic la supone del tipo *Variant* (puede almacenar cualquier tipo de dato).
- Asignación del contenido de *Text3* a la variable global *n*.

```
If IsNumeric(i) And IsNumeric(CP) And IsNumeric(n) And (i > 0) And (i <= 100) And (CP
>= 0) And (n >= 1) And (i <> "") And (CP <> "") And (n <> "") Then
  If (n / CInt(n)) = 1 Then
```

- La sentencia de control *if* anidada va a inspeccionar si las variables *i*, *CP* y *n* contienen datos válidos, es decir, que no estén vacías, que contengan números dentro de un rango admitido, y adicionalmente en el caso de *n* que almacene un número entero. En caso afirmativo se continuará con el procedimiento, en caso contrario se abortará. Al momento de abortar el procedimiento se va a presentar al usuario una caja de diálogo con el siguiente mensaje: *Dato inválido ¡Revisar el frame de Datos¡*.

```
With MSFlexGrid1
 For j = 1 To n
   .Row = j
    For K = 1 To 3
     .Col = K
      If .Text = "" Then
      MsgBox "Alguna o algunas de las celdas de la tabla estan vacías", 16
      Exit Sub
      Else

      End If
    Next K
 Next j
End With
```

- Esta rutina permite inspeccionar todas las celdas de datos de *MSFlexGrid1* . En caso de que alguna o algunas de las celdas estén vacías se finalizará el procedimiento. Al momento de finalizar el procedimiento se va a presentar al usuario una caja de diálogo con el siguiente mensaje: *Alguna o algunas de las celdas de la tabla estan vacías*.
- El procedimiento *MSFlexGrid1_KeyPress* descrito en el apartado anterior, evita la introducción de letras, números negativos o cualquier otro símbolo no válido en las celdas de *MSFlexGrid1*, por lo que sólo es necesario el filtro que revisa si alguna o algunas de las celdas de datos están vacías.

C = CP

- Asignación del contenido de *CP* a la variable global *C* (restricción de presupuesto). Esta sentencia no hace otra cosa que asignar el contenido de *Text2* a *C*, pero, ¿Porqué no hacerlo directamente desde las asignaciones iniciales del procedimiento?.
- En el apartado anterior se definió a *C* como una variable del tipo *Double*, por lo que solo acepta que se le asignen números. Si se le asignara el contenido de *Text2* antes de pasar el filtro de la sentencia de control *if* anidada, se correría el riesgo que almacenara un tipo de dato no válido (letra o incluso vacío), lo cual generaría un error de ejecución. Es por ello que se utiliza una variable intermedia (*CP*) que garantiza después de pasar el *if* anidado contener un número válido, el cual ya es posible asignar a *C*.

i = i / 100

- Conversión de *i* (costo total anual de mantener el inventario) en un dato válido para el algoritmo de solución planteado. Inicialmente *i* es un dato que debe ser suministrado en porcentaje anual, es decir, un número de 0 a 100, pero para efectos de cálculo debe ser un número entre 0 y 1.

Call Paso1Calcular

- Esta sentencia llama al *procedimiento estándar*[14] *Paso1Calcular* que contiene el primer paso del método de *Multiplicadores de Lagrange* modificado para el modelo en estudio, es decir, en este paso se calculan los lotes óptimos (Q^*_i) para cada clase de artículo sin involucrar la restricción de presupuesto (*C*), para después obtener la inversión total en inventario (*Cs*) relacionada a esos óptimos.

[14] Para mayor referencia con respecto a los *procedimientos estándar* y la forma de llamarlos, dirigirse a el capítulo 3 apartado 5.1.7 de este documento.

```
If (Cs <= C) Then
Call Paso1Escribir
Text4.Text = ""
Else
Call Paso2Calcular
End If
```

- Si la inversión total en inventario (*Cs*) es menor o igual a la restricción de presupuesto (*C*) se ejecutarán un par de sentencias:
- La primera dispara el procedimiento *Paso1Escribir*, donde se recalculan y despliegan los óptimos no restringidos (Q^*_i) y el costo total anual promedio del inventario (*K(Q)*) relacionado.
- La segunda borra cualquier posible contenido de la caja de texto *Text4*, ya que en ella se despliega el valor de λ (*Multiplicador de Lagrange*), y como es lógico, si no se calculan los "óptimos restringidos", no existe λ.
- Si la inversión total en inventario (*Cs*) no es menor o igual a la restricción de presupuesto (*C*) se disparará el procedimiento *Paso2Calcular*, donde se calculan los lotes óptimos (Q^*_i) restringidos, es decir, se deduce λ, y con él se calculan y despliegan los óptimos restringidos (Q^*_i) y el costo total anual promedio del inventario (*K(Q)*) relacionado.

```
  Else
  MsgBox "Dato inválido ¡Revisar el frame de Datos!", 16
  End If


Else
MsgBox "Dato inválido ¡Revisar el frame de Datos!", 16
End If
```

```
ManipularError:
   If Err.Number = 0 Then
   Exit Sub
   Else
   MsgBox "Dato inválido:" & Err.Description, 16
   Err.Clear
   Exit Sub
   End If

End Sub
```

- Cuando se produce un error, el error se intercepta con la sentencia *On Error GoTo* y se manipula en esta rutina.

- Si no se presentan errores en tiempo de ejecución (*Err.Number = 0*), se finaliza el procedimiento (sentencia *Exit Sub*). En caso contrario, la función intrínseca *MsgBox* visualiza una caja de diálogo del tipo 16, es decir, que tiene que ser cerrada para poder continuar con el procedimiento (*Modal*) pulsando el botón aceptar. La caja de diálogo va a desplegar el mensaje *Dato inválido:* concatenado con la descripción del error que se produjo (propiedad *Description* del objeto *Err*), asimismo, después se van a iniciar las propiedades del objeto *Err* a cero o nulo, usando el método *Clear*, finalizando el procedimiento con la sentencia *Exit Sub*.

A diferencia de todos los procedimientos desarrollados hasta este momento (que han sido guiados por eventos), el siguiente es un procedimiento estándar llamado *Paso1Calcular*, el cual contiene el primer paso del algoritmo de solución para el modelo en estudio.

```
Private Sub Paso1Calcular()

Cs = 0
```

- Procedimiento privado (tipo *Sub*) *Paso1Calcular*.
- Inicialización de la variable global *Cs*. Como la inversión total en inventario es una variable global, puede almacenar datos calculados en corridas previas de la interfaz, es por ello que es necesario inicializarla, ya que debido a la naturaleza del algoritmo de solución esos datos se podrían sumar al calculado en la corrida en uso, arrojando información errónea al usuario.

```
With MSFlexGrid1

For j = 1 To n
   .Row = j
   For K = 1 To 4
      .Col = K
        Select Case K
          Case 1: A = .Text
          Case 2: D = .Text
          Case 3: Cm = .Text
          Case 4:
          Qop = Sqr((2 * A * D) / (i * Cm))
          cu = Cm * Qop
```

- Esta rutina inspecciona todas las celdas de datos de *MSFlexGrid1* , obteniendo los valores de *A, D* y *Cm* (*c*) para cada clase de artículo en el sistema (*n*), con estos valores y el de la variable global *i* (que fue leída en el procedimiento anterior) se procede a calcular los lotes óptimos *Qop* (Q^*_i) para cada clase de artículo, y con esos lotes óptimos se obtiene la inversión total en inventario (*Cs*).

```
        End Select
    Next K
    Cs = Cs + cu

Next j

End With

End Sub
```

El próximo, es el procedimiento estándar *Paso1Escribir*, donde se recalculan y despliegan los lotes óptimos no restringidos (Q^*_i) y el costo total anual promedio del inventario (*K(Q)*) relacionado.

```
Private Sub Paso1Escribir()

KT = 0
```

- Procedimiento privado (tipo *Sub*) *Paso1Escribir*.
- Inicialización de la variable *KT* (*K(Q)*).

```
With MSFlexGrid1

For j = 1 To n
    .Row = j
    For K = 1 To 4
        .Col = K
          Select Case K
            Case 1: A = .Text
            Case 2: D = .Text
            Case 3: Cm = .Text
            Case 4:
            Qop = Sqr((2 * A * D) / (i * Cm))
            .Text = CSng(Qop)
            Kp = (Cm * D) + ((A * D) / Qop) + ((Cm * i) * (Qop / 2))
          End Select
```

- Esta rutina inspecciona todas las celdas de datos de *MSFlexGrid1* , obteniendo los valores de *A, D* y *Cm* (*c*) para cada clase de artículo en el sistema (*n*), con estos valores y el de la variable global *i* (que fue leída en el procedimiento anterior) se procede a calcular y desplegar los lotes óptimos *Qop* (Q^*_i) para cada clase de artículo, y con esos lotes óptimos se obtiene y despliega el costo total anual promedio del inventario *KT* (*K(Q)*).

```
   Next K
   KT = KT + Kp
Next j
Text5.Text = KT
End With

End Sub
```

- Antes de escribir los valores de los lotes óptimos en la tabla *MSFlexGrid1* se deben convertir a un tipo de dato *Single*[15], mediante la función de conversión de tipo de dato *CSng()*, la cual convierte su argumento al tipo de dato *Single*. Lo anterior tiene por objeto reducir la precisión en los valores de los lotes óptimos (disminuir sus decimales), ya que con la precisión por omisión (tipo de dato *Variant*) no se visualizan por completo en las celdas de la tabla *MSFlexGrid1*.
- El costo total anual promedio del inventario *KT* (*K(Q*) se despliega en la caja de texto *Text5*.

En el procedimiento estándar *Paso2Calcular* se calculan los lotes óptimos (Q^*_i) restringidos, es decir, se deduce λ (*Multiplicador de Lagrange*), y con él se calculan y despliegan los óptimos restringidos y el costo total anual promedio del inventario (*K(Q)*) relacionado. A continuación se presenta dicho procedimiento:

```
Private Sub Paso2Calcular()

S = 0
KT = 0
```

- Procedimiento privado (tipo *Sub*) *Paso2Calcular*.
- Inicialización de la variable intermedia *S*.
- Inicialización de la variable *KT* (*K(Q)*).

```
With MSFlexGrid1

For j = 1 To n
   .Row = j
   For K = 1 To 3
      .Col = K
        Select Case K
          Case 1: A = .Text
          Case 2: D = .Text
          Case 3: Cm = .Text
          End Select
   Next K
   T = Sqr((2 * A * D * Cm))
   S = S + T
Next j

Lambda = ((1 / (2 * C * C)) * (S * S)) - (i / 2)
Text4.Text = Lambda
```

- Esta rutina lee de *MSFlexGrid1* todos los datos relacionados (*A, D, Cm*) a cada clase de artículo en el sistema (*n*), para que junto con el costo total anual de mantener el inventario (*i*) y la restricción de presupuesto (*C*) se proceda a calcular el Multiplicador de Lagrange *Lambda* (λ), de acuerdo con la Fórmula 2-1 que aparece en el capítulo 2 apartado 2.1 de este documento.
– El Multiplicador de Lagrange se despliega en la caja de texto *Text4*.

[15] Las variables del tipo *Single* almacenan sólo números de punto flotante con precisión simple (4 bytes).

```
For j = 1 To n
   .Row = j
   For K = 1 To 4
      .Col = K
        Select Case K
          Case 1: A = .Text
          Case 2: D = .Text
          Case 3: Cm = .Text
          Case 4:
          Qop = Sqr((2 * A * D) / ((i * Cm) + (2 * Lambda * Cm)))
          .Text = CSng(Qop)
          Kp = (Cm * D) + ((A * D) / Qop) + ((Cm * i) * (Qop / 2))
        End Select
   Next K
   KT = KT + Kp
Next j

Text5.Text = KT

End With

End Sub
```

- En esta rutina se vuelven a leer todos los datos relacionados a cada clase de artículo en el sistema (*n*), para que junto con el Multiplicador de Lagrange (que fue calculado en la rutina anterior) y la variable global *i* se proceda a calcular los lotes óptimos *Qop* (Q^*_i) para cada clase de artículo, y con esos lotes óptimos se obtenga el costo total anual promedio del inventario *KT* (*K(Q)*).

- Inmediatamente después de que se calculan los lotes óptimos se convierten a un tipo de dato *Single* (mediante la función de conversión de tipo de dato *CSng()*) y se despliegan en la última columna de *MSFlexGrid1*.
- Los lotes óptimos se deducen de acuerdo a la Ec. A del capítulo 2 apartado 2.1 de este documento.
- El costo total anual promedio del inventario se despliega en la caja de texto *Text5*.

Si se ha sido observador, el lector habrá advertido que en algunos procedimientos se han incluido sentencias (*On Error GoTo*) que interceptan cualquier tipo de error que se produzca en ellos. Tales sentencias también incluyen una etiqueta denominada *ManipularError*, que identifica la primera línea de la rutina que manipula el error (la rutina por lo regular se encuentra al final de los procedimientos).

Luego entonces, ¿Porqué no introducir sentencias que intercepten cualquier tipo de error, y rutinas que los manipulen en todos los procedimientos de la aplicación?. Pues bien, por las siguientes 2 razones:

- Porque ciertos procedimientos no tienen interacción directa con el usuario, es decir, no es posible que se produzcan errores en tiempo de ejecución, debido a que el usuario no puede introducir información inválida en tales procedimientos. En contraste, procedimientos en los cuales si se pueda introducir información inválida requieren rutinas de manipulación de errores. Por ejemplo, en un procedimiento que permita abrir un determinado archivo, es posible que el usuario especifique un camino incorrecto, el nombre del archivo sea equívoco sintácticamente hablando, o bien la unidad en la que se encuentra el archivo no esté disponible; en estos casos el procedimiento fallará (si es que no cuenta con una rutina de manipulación de errores), y por lo tanto también la aplicación.
- Porque cuando ocurre un error en un procedimiento que no tiene una rutina de manipulación del error, Visual Basic busca hacia atrás en la secuencia de llamadas

otra rutina de manipulación, es decir, no es necesario que un subprocedimiento incluya una rutina de manipulación de error, si es que el procedimiento que lo llamó (procedimiento principal) la contiene, ya que si en el subprocedimiento ocurre un error, Visual Basic va a buscar una rutina de manipulación del error en el subprocedimiento, y si no la encuentra la buscará en el procedimiento principal. El procedimiento principal *Command1_Click* y los subprocedimientos *Paso1Calcular, Paso1Escribir* y *Paso2Calcular* de este apartado, son un buen ejemplo de tal comportamiento.

2.4 Carga y descarga de la interfaz

Para concluir con el desarrollo de la interfaz de usuario para el análisis con la restricción de presupuesto (*Form1*), sólo hace falta escribir un par de procedimientos que permitan cargar y descargar dicha interfaz.

El primero de ellos permite cargar a *Form1* desde la interfaz principal de usuario (*MDIForm1*) cuando se hace *clic* en la orden *Presupuesto* del menú *Análisis con Restricción de..* El nombre de tal procedimiento es *MnuPresupuesto_Click*, y es conducido por el evento *Click* de *MnuPresupuesto*. Se presenta a continuación:

```
Private Sub MnuPresupuesto_Click()

Load Form1

End Sub
```

- Procedimiento privado (tipo *Sub*) *MnuPresupuesto_Click* conducido por el evento *Click* del menú *MnuPresupuesto*.
- La sentencia *Load* permite cargar al formulario *Form1* desde la interfaz principal de usuario (*MDIForm1*).

También es posible disparar el procedimiento anterior (desde *MDIForm1*) si se hace *clic* en el botón de la barra de herramientas etiquetado con la leyenda *Presup.*, ya que se encuentran vinculados (el botón y *MnuPresupuesto_Click*) gracias al procedimiento *Toolbar1_ButtonClick* del apartado 1.6.3 de el presente capítulo.

El segundo procedimiento permite descargar el formulario *Form1* cuando el usuario hace *clic* en el botón de comando (*Command2*) de dicho formulario que tiene la etiqueta *Salir*. El procedimiento es denominado *Command2_Click*, y es conducido por el evento *Click* de *Command2*. Se presenta a continuación:

```
Private Sub Command2_Click()

Unload Form1

End Sub
```

- Procedimiento privado (tipo *Sub*) *Command2_Click* conducido por el evento *Click* del control *Command2*.
- La sentencia *Unload* permite descargar al formulario *Form1*.

3 ANÁLISIS CON LA RESTRICCIÓN DE ESPACIO

Hasta este punto se ha explicado exhaustivamente cada detalle en el desarrollo de la aplicación (*AIRRAM*), por lo que el lector estará habituado a muchos de los conceptos y procedimientos que se han venido utilizando en la construcción de la misma. Gran cantidad de esos conceptos y procedimientos se seguirán empleando en lo que resta del desarrollo, tal vez no exactamente de la misma forma, pero si de un modo análogo o similar. En consecuencia, se obviará la descripción de numerosas partes en la programación restante de *AIRRAM* (por no ser necesaria), sólo en los casos en los que se le dé una solución sustancialmente diferente a una futura parte del desarrollo se ahondará en su explicación.

Es necesario añadir un par de variables a la colección que se ha venido utilizando, y redefinir una más:

- $F =$ Es el límite del espacio total en inventario (restricción de espacio), suministrado en unidades de espacio. Su nomenclatura de código será la misma (*F*).
- $f_i =$ Es el espacio requerido para una unidad del artículo clase *i* (espacio unitario para cada clase de artículo), suministrado en unidades de espacio. Para toda $i = 1,2,.....n$. Su nomenclatura de código será *Fm*.
- *Cs* = Su significado cambia de la inversión total en inventario, al espacio total en inventario, es decir, $Cs = \sum_{i=1}^{n} f_i Q_i$. Su nomenclatura de código será la misma (*Cs*).

Para la construcción de la interfaz de usuario para el análisis con la restricción de presupuesto (*Form1*), se aplicó el siguiente método:

- Diseño de la interfaz.
- Funcionalidad de la interfaz.
- Programación de los algoritmos de solución.
- Carga y descarga de la interfaz.

El cual es lógico repetir en el desarrollo de la *interfaz de usuario para el análisis con la restricción de espacio*.

3.1 Diseño de la interfaz

El diseño de la interfaz será muy similar al de *Form1*. Para empezar hay que insertar un nuevo formulario del tipo *SDI* en el proyecto, que tendrá el nombre de código *Form2*.

Agregando y especificando controles y propiedades en *Form2* de la misma forma que se hizo en *Form1*, se obtiene el resultado que se exhibe en la Figura 6-22.

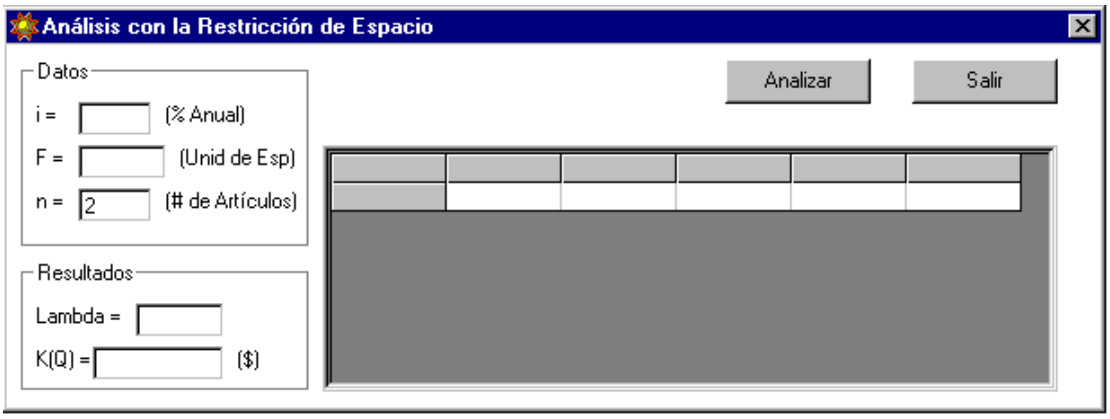


Figura 6-22
Apariencia de *Form2* en tiempo de diseño

Como es evidente los formularios *Form2* y *Form1* son muy parecidos entre si, sólo difieren en lo siguiente:

- La barra de título. En este caso a la propiedad ***Caption*** de *Form2* se le asigna el título *Análisis con la Restricción de Espacio.*
- El ancho del formulario. ***Width:*** *9225 twips*.
- En el *frame* de *Datos* se intercambia la restricción de presupuesto (*C*) por la de espacio (*F*). ***Label2.Caption:*** *F =* , ***Label5.Caption:*** *(Unid de Esp)*.
- Se añade una columna a la tabla *MSFlexGrid1*. ***MSFlexGrid1.Cols:*** *6*. En tal columna se va a introducir el espacio unitario para cada clase de artículo (*f*).

3.2 Funcionalidad de la interfaz

La funcionalidad de *Form2* se alcanza valiéndose de los mismos 5 procedimientos descritos para *Form1*, sólo se insertan un par de líneas mas de código, las cuales permiten etiquetar el título de la nueva columna de *MSFlexGrid1*, y declarar a la variable global *F*.

La apariencia de *Form2* en tiempo de ejecución se muestra en la Figura 6-23.

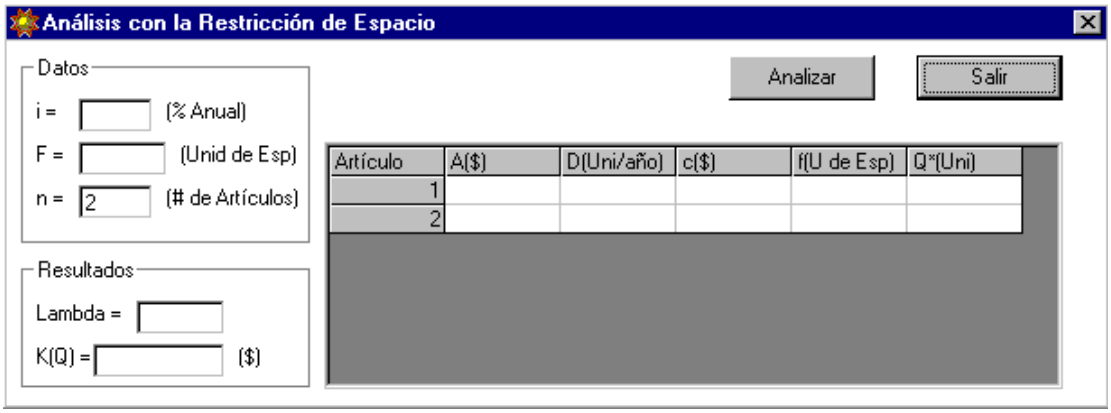


Figura 6-23
Apariencia de *Form2* en tiempo de ejecución

Si se requiere consultar el código completo de los procedimientos relacionados a la interfaz de usuario para el análisis con la restricción de espacio (*Form2*), dirigirse a los **Anexos** de este documento.

3.3 Programación de los algoritmos de solución

En este apartado se va a programar el *método de Multiplicadores de Lagrange modificado* para el modelo de inventarios de artículos múltiples con restricción de recursos (***Espacio***)[16]. El algoritmo de solución para el modelo con la restricción de espacio (*Form2*), es análogo al que se usó para el modelo con la restricción de presupuesto (*Form1*), es por ello que los procedimientos en los que se programan dichos algoritmos son muy similares para ambos formularios, de hecho *Form2* solo cuenta con un par de procedimientos sustancialmente diferentes, que son los únicos que se van a describir en este apartado.

El primero de ellos es el procedimiento estándar *Paso2Calcular*, en donde se calculan los lotes óptimos (Q^*_i) restringidos, es decir, se deduce λ (*Multiplicador de Lagrange*), y con él se calculan y despliegan los óptimos restringidos y el costo total anual promedio del inventario ($K(Q)$) relacionado. La diferencia fundamental entre el procedimiento *Paso2Calcular* de este apartado, y su correspondiente del apartado 2.3 de este capítulo radica en la forma en la que se deduce λ, pues mientras en el apartado 2.3 se obtiene simplemente sustituyendo valores en una fórmula, en este apartado hay que resolver una ecuación no lineal de la cual λ es la raíz.[17]

El método numérico que se ha elegido para dar solución a la ecuación no lineal antes citada es el de la **Bisectriz**, por lo que es conveniente hacer un pequeño paréntesis en este apartado para describirlo.

[16] Para mayor referencia con respecto a el *método de Multiplicadores de Lagrange modificado* para el modelo de inventarios de artículos múltiples con restricción de recursos (***Espacio***), dirigirse a el capítulo 2 apartados 1.2 y 2.2 de este documento.
[17] La ecuación no lineal a la cual se hace referencia, es la Fórmula 2-2 que aparece en el capítulo 2 apartado 2.2 de este documento.

3.3.1 Método de la Bisectriz

El método de la bisectriz es un método numérico sencillo pero muy versátil para encontrar una raíz real en un intervalo dado en el que se sabe que existe una raíz.

Supóngase que hay una raíz de *f(x) = 0* en un intervalo entre *x = LI* (donde *LI* es el límite inferior) y *x = LS* (donde *LS* es el límite superior), denotado por [*LI*, *LS*] o, de forma equivalente, por $LI \leq x \leq LS$. El método de la bisectriz se basa en el hecho de que, cuando un intervalo [*LI*, *LS*] tiene una raíz, el signo de *f(x)* en los dos extremos es distinto, o sea, $f(LI)f(LS) < 0$. El primer paso de este método consiste en bisecar el intervalo [*LI*, *LS*] en dos mitades: [*LI*, *R*] y [*R*, *LS*], donde *R = (LI + LS)/2* (*R* es una estimación de la raíz). Si se verifican los signos de *f(LI)f(R)* y *f(R)f(LS)*, se sabrá en que mitad del intervalo se encuentra la raíz. De hecho, si $f(LI)f(R) \leq 0$, se sabrá que el intervalo [*LI*, *R*], que incluye *x = LI* y *x = R*, contiene a la raíz; de lo contrario, la raíz estará en el otro intervalo, [*R*, *LS*]. A continuación, se biseca de nuevo el intervalo que contiene a la raíz. Al repetir este procedimiento, el tamaño del intervalo que contiene a la raíz se hará cada vez más pequeño. En cada paso se toma el punto medio del intervalo como la aproximación más actualizada a la raíz.

El procedimiento puede continuarse hasta que el resultado sea lo suficientemente exacto para llenar los requerimientos. Y el criterio que decide cuándo se cumple esta premisa, es decir, en qué momento terminar el método, es el *error relativo porcentual* (que debe ser estipulado por el usuario), el cual puede designarse como *ES*. Así, después de cada iteración puede calcularse un *error relativo porcentual aproximado EA* como

$$EA = \left| \frac{LS - LI}{2R} \right| \times 100\% = \left| \frac{LS - LI}{LI + LS} \right| \times 100\%$$

Cuando *EA* cae abajo de *ES*, el cálculo puede terminarse con la seguridad de que la raíz se conoce por lo menos al nivel especificado por *ES*.

Ahora que ya se conoce el método de la bisectriz será factible comprender el procedimiento *Paso2Calcular*. A continuación se presenta dicho procedimiento:

```
Private Sub Paso2Calcular()

KT = 0
Contador = 0
Sum = 0


LI = 0
LS = 10
```

- Procedimiento privado (tipo *Sub*) *Paso2Calcular*.

```
While (FNF(LI) * FNF(LS)) >= 0
  LS = LS * 10
  Contador = Contador + 1
  If Contador >= 10 Then
  MsgBox "No existe Lambda", 16
  Exit Sub
  End If
Wend
```

- Esta rutina permite definir los límites entre los cuales posiblemente se encontrará *Lambda* (λ).
- Para el modelo en estudio λ toma regularmente valores de 0 a 3, y en ocasiones mayores a 5. Es por ello que el intervalo inicial es de 0 a 10.
- Cuando la restricción de espacio (F) adquiere una relación francamente desproporcionada con respecto a los espacios unitarios para cada clase de artículo en el sistema Fm (f_i) los valores de λ se disparan. La rutina responde a tales circunstancias incrementando gradualmente el intervalo en análisis hasta un límite superior cercano a un billón, aunque posiblemente jamás se tropezará con un problema real en el que λ adquiera valores superiores a 10.

```
30:
varAntes = Timer
X = (LI + LS) / 2
40: If FNF(X) = 0 Then GoTo 100
50: If (FNF(LI) * FNF(X)) <= 0 Then GoTo 70
60: LI = X
65: GoTo 80
70: LS = X
80: If (Abs(LS - LI) / (Abs(LI) + Abs(LS))) < 0.0001 Then GoTo 100
90:
varDespues = Timer
Dif = varDespues - varAntes
Sum = Sum + Dif
If Sum >= 5 Then
Sum = 0
Resp = MsgBox("El análisis tomará algunos segundos más, ¿Desea continuar?", vbYesNo)
  If Resp = 7 Then Exit Sub
End If
GoTo 30
100: Lambda = X
```

- Esta pequeña rutina se apoya en el método de la bisectriz para encontrar la raíz (λ) de la Fórmula 2-2, que aparece en el capitulo 2 apartado 2.2 de este documento. El error relativo porcentual es de 0.01%.

- El procedimiento *FNF* es un procedimiento estándar tipo *función*,[18] en el cual se construye y evalúa la ecuación no lineal mencionada en el párrafo anterior (Fórmula 2-2). Cuando la función *FNF* es invocada se le pasa un argumento, por ejemplo *X*, el cual es una estimación de *Lambda* (λ), y *FNF* retorna su valor a través de su nombre, es decir, se sustituye la estimación de λ en la ecuación que construye *FNF* y su valor se devuelve al procedimiento *Paso2Calcular*.

[18] Para mayor referencia con respecto al tópico *funciones*, dirigirse a el capítulo 3 apartado 5.1.7.3 de este documento.

```
With MSFlexGrid1

Text4.Text = Lambda

For j = 1 To n
   .Row = j
   For K = 1 To 5
      .Col = K
        Select Case K
          Case 1: A = .Text
          Case 2: D = .Text
          Case 3: Cm = .Text
          Case 4: Fm = .Text
          Case 5:
          Qop = Sqr((2 * A * D) / ((i * Cm) + (2 * Lambda * Fm)))
          .Text = CSng(Qop)
          Kp = (Cm * D) + ((A * D) / Qop) + ((Cm * i) * (Qop / 2))
        End Select
   Next K
   KT = KT + Kp
Next j

Text5.Text = KT

End With
End Sub
```

- El código restante es muy similar al del procedimiento *Paso2Calcular* del apartado 2.3 de este capítulo.

En la función *FNF* se construye y evalúa la ecuación no lineal de la cual se habló en el procedimiento anterior. Dicha función es:

```
Public Function FNF(Lambda As Variant)

S = 0

With MSFlexGrid1

For j = 1 To n
   .Row = j
   For K = 1 To 4
      .Col = K
        Select Case K
          Case 1: A = .Text
          Case 2: D = .Text
          Case 3: Cm = .Text
```

```
        Case 4: Fm = .Text
        End Select
    Next K
    T = (Sqr((2 * A * D) / ((i * Cm) + (2 * Lambda * Fm)))) * (Fm)
    S = S + T
Next j

FNF = S - F

End With

End Function
```

3.4 Carga y descarga de la interfaz

Como en el caso de *Form1*, solo hace falta presentar un par de procedimientos que permitan cargar y descargar al formulario *Form2* para concluir con su desarrollo.

El primero de ellos permite cargar a *Form2* desde la interfaz principal de usuario (*MDIForm1*) cuando se hace *clic* en la orden *Espacio* del menú *Análisis con Restricción de.*. El nombre de tal procedimiento es *MnuEspacio_Click*, y es conducido por el evento *Click* de *MnuEspacio*. Se presenta a continuación:

```
Private Sub MnuEspacio_Click()
Load Form2
End Sub
```

También es posible disparar el procedimiento anterior (desde *MDIForm1*) si se hace *clic* en el botón de la barra de herramientas etiquetado con la leyenda *Espacio*, ya que se encuentran vinculados (el botón y *MnuEspacio_Click*) gracias al procedimiento *Toolbar1_ButtonClick* del apartado 1.6.3 de el presente capítulo.

El segundo procedimiento permite descargar el formulario *Form2* cuando el usuario hace *clic* en el botón de comando (*Command2*) de dicho formulario que tiene la etiqueta *Salir*. El procedimiento es denominado *Command2_Click*, y es conducido por el evento *Click* de *Command2*. Se presenta a continuación:

```
Private Sub Command2_Click()
Unload Form2
End Sub
```

---

4 ANÁLISIS CON AMBAS RESTRICCIONES

En este apartado se va a desarrollar la *interfaz de usuario para el análisis con ambas restricciones* (***Presupuesto y Espacio***). Antes de empezar con el desarrollo es necesario añadir tres variables a la colección que se ha venido utilizando:

- *Ks* = Es la inversión total en el inventario, es decir, $Ks = \sum_{i=1}^{n} c_i Q_i$. Su nomenclatura de código será *Ks*. Hasta el apartado 2 del presente capítulo esta variable se manejaba con el símbolo *Cs*, pero hay que recordar que en el apartado 3 se le asignó a *Cs* un nuevo significado (el espacio total en inventario), por lo que fue necesario dar un nuevo símbolo a la inversión total en el inventario (*Ks*).
- $\lambda_1 = \lambda 1$ = Es el primer multiplicador de Lagrange, es adimensional. Su nomenclatura equivalente será *Lambda1*.
- $\lambda_2 = \lambda 2$ = Es el segundo multiplicador de Lagrange, es adimensional. Su nomenclatura equivalente será *Lambda2*.

Para la construcción de esta interfaz se aplicará el mismo método que para *Form1* y *Form2*, el cual es:

- Diseño de la interfaz.
- Funcionalidad de la interfaz.
- Programación de los algoritmos de solución.
- Carga y descarga de la interfaz.

4.1 Diseño de la interfaz

Para empezar hay que insertar un nuevo formulario del tipo *SDI* en el proyecto, que tendrá el nombre de código *Form3*.

Agregando y especificando controles y propiedades en *Form3* de la misma forma que se hizo en *Form1* y *Form2*, se obtiene el resultado que se exhibe en la Figura 6-24.

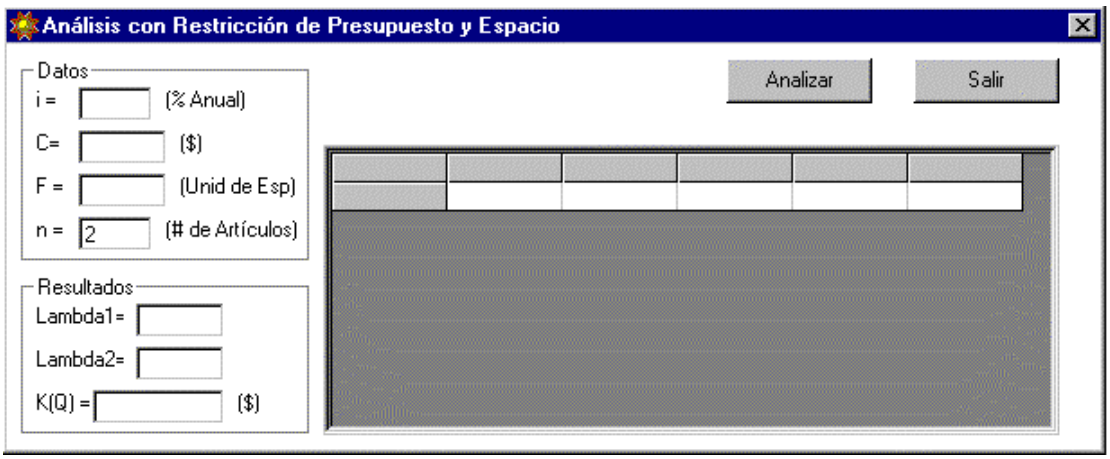


Figura 6-24
Apariencia de *Form3* en tiempo de diseño

Form3 solo difiere de *Form2* en lo siguiente:

- La barra de título. En este caso a la propiedad ***Caption*** de *Form3* se le asigna el título *Análisis con Restricción de Presupuesto y Espacio.*
- En el *frame* de *Datos* se introduce una nueva caja de texto (con nombre de código *Text6*), en la cual se va a introducir la restricción extra.
- En el *frame* de *Resultados* se introduce una nueva caja de texto (con nombre de código *Text7*), en la cual se va a desplegar el multiplicador de Lagrange extra.

4.2 Funcionalidad de la interfaz

La funcionalidad de *Form3* se alcanza valiéndose de los mismos 5 procedimientos descritos para *Form1* y *Form2*, sólo se inserta una línea más de código que permite declarar a la variable global *Ks*.

La apariencia de *Form3* en tiempo de ejecución se muestra en la Figura 6-25.

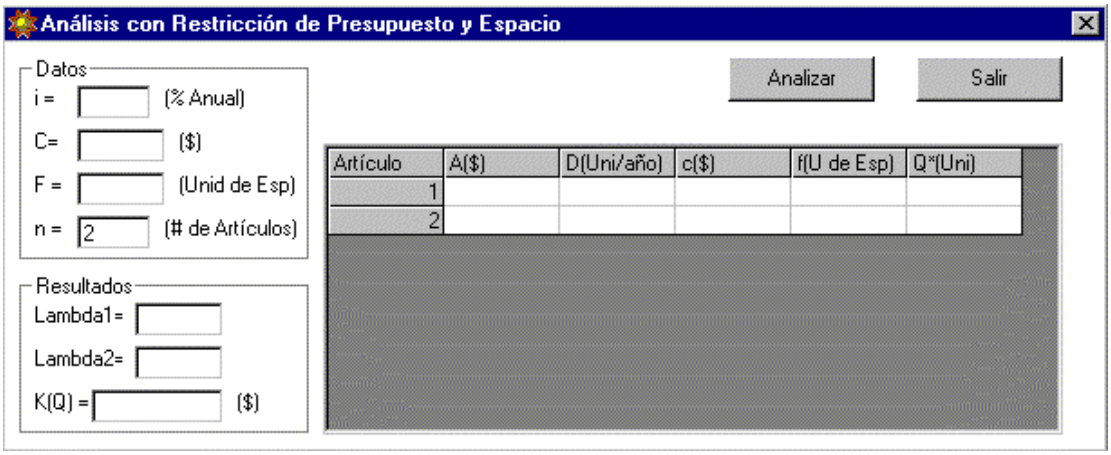


Figura 6-25
Apariencia de *Form3* en tiempo de ejecución

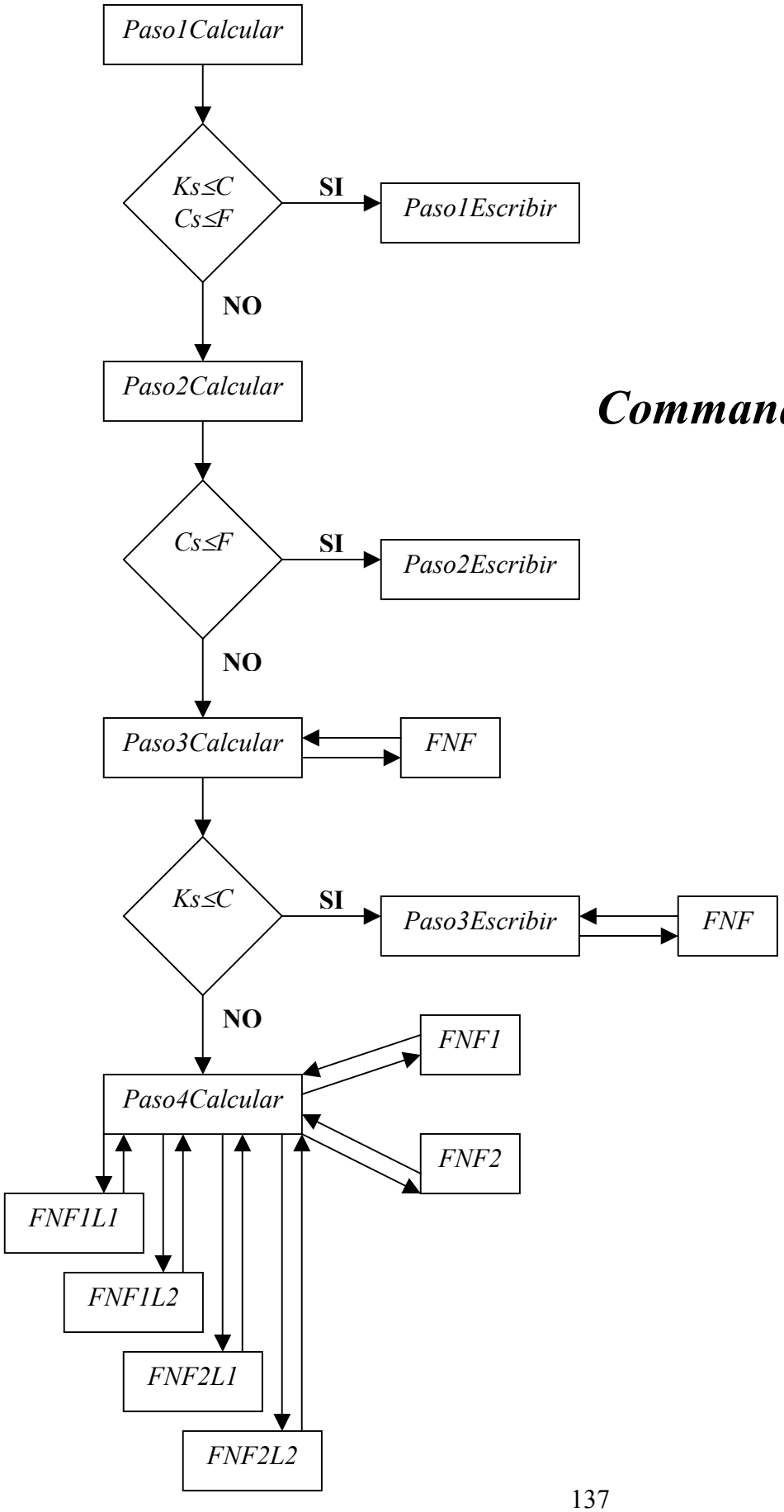
Si se requiere consultar el código completo de los procedimientos relacionados a la interfaz de usuario para el análisis con ambas restricciones (*Form3*), dirigirse a los **Anexos** de este documento.

4.3 Programación de los algoritmos de solución

En este apartado se va a programar el *método de Multiplicadores de Lagrange modificado* para el modelo de inventarios de artículos múltiples con restricción de recursos (***Presupuesto y Espacio***)[19].

El algoritmo de solución para el modelo en estudio es el más complejo y completo de los que se ha utilizado, incluso se puede considerar que los algoritmos anteriores (apartados 2.3 y 3.3 de este capítulo) son parte de el. Es por ello que a continuación se presenta el diagrama de flujo del procedimiento *Command1_Click*, procedimiento en el cual se programa el algoritmo de solución para el modelo en estudio.

[19] Para mayor referencia con respecto a el *método de Multiplicadores de Lagrange modificado* para el modelo de inventarios de artículos múltiples con restricción de recursos (***Presupuesto y Espacio***), dirigirse a el capítulo 2 apartados 1.3 y 2.3 de este documento.


Paso1Calcular
Ks≤C
Cs≤F
SI
Paso1Escribir
NO
Paso2Calcular
Command1_Click
Cs≤F
SI
Paso2Escribir
NO
Paso3Calcular
FNF
Ks≤C
SI
Paso3Escribir
FNF
NO
FNF1
Paso4Calcular
FNF2
FNF1L1
FNF1L2
FNF2L1
FNF2L2

Cuando el usuario final hace *clic* en el botón de comando de *Form3* que tiene la etiqueta *Analizar*, se desencadena el procedimiento *Command1_Click*, que a su vez llama a los siguientes subprocedimientos.

- *Paso1Calcular*. En este procedimiento se calculan los lotes óptimos (Q^*_i) no restringidos. Si se respetan ambas restricciones ($Ks \leq C$ y $Cs \leq F$) se dispara el procedimiento *Paso1Escribir*, de lo contrario se dispara el procedimiento *Paso2Calcular*. El procedimiento *Paso1Calcular* de este apartado es muy similar al del mismo nombre del apartado 2.3 de este capítulo.
- *Paso1Escribir*. Aquí se recalculan y despliegan los óptimos no restringidos (Q^*_i) y el costo total anual promedio del inventario ($K(Q)$) relacionado. El procedimiento *Paso1Escribir* de este apartado es muy similar al del mismo nombre del apartado 2.3 de este capítulo.
- *Paso2Calcular*. Aquí se calculan los óptimos (Q^*_i) bajo la restricción de presupuesto (C), es decir, se deduce el multiplicador de Lagrange (λ) y con el se calculan los óptimos restringidos. Si se respeta la restricción de espacio ($Cs \leq F$) se dispara el procedimiento *Paso2Escribir*, de lo contrario se dispara el procedimiento *Paso3Calcular*. El procedimiento *Paso2Calcular* de este apartado es muy similar al del mismo nombre del apartado 2.3 de este capítulo.
- *Paso2Escribir*. Aquí se recalculan y despliegan los óptimos (Q^*_i) bajo la restricción de presupuesto y el costo total anual promedio del inventario ($K(Q)$) relacionado. El procedimiento *Paso2Escribir* de este apartado es muy similar al *Paso2Calcular* del apartado 2.3 de este capítulo.
- *Paso3Calcular*. En el se calculan los óptimos (Q^*_i) bajo la restricción de espacio (F), es decir, se deduce el multiplicador de Lagrange (λ) mediante el método de la **Bisectriz** y con el se calculan los óptimos restringidos. Si se respeta la restricción de presupuesto ($Ks \leq C$) se dispara el procedimiento *Paso3Escribir*, de lo contrario se dispara el procedimiento *Paso4Calcular*. En este procedimiento el multiplicador de Lagrange (λ) es la raíz de una ecuación no lineal que se resuelve mediante el método de la Bisectriz, tal ecuación no lineal se construye y evalúa en el procedimiento tipo función *FNF*. El procedimiento *Paso3Calcular* de este apartado es muy similar al *Paso2Calcular* del apartado 3.3 de este capítulo.
- *Paso3Escribir*. Aquí se recalculan y despliegan los óptimos (Q^*_i) bajo la restricción de espacio y el costo total anual promedio del inventario ($K(Q)$) relacionado. En este procedimiento se sigue utilizando la función *FNF*. El procedimiento *Paso3Escribir* de este apartado es muy similar al *Paso2Calcular* del apartado 3.3 de este capítulo.
- *Paso4Calcular*. En este procedimiento se calculan los lotes óptimos (Q^*_i) bajo ambas restricciones (*Presupuesto y Espacio*), es decir, se deducen dos multiplicadores de Lagrange ($\lambda 1$ y $\lambda 2$) y con ellos se calculan los óptimos restringidos. En este procedimiento los multiplicadores de Lagrange ($\lambda 1$ y $\lambda 2$) son las raíces de un sistema de ecuaciones no lineales[20] que se resuelve mediante el método de **Newton**. También se utilizan seis funciones más, dos de las cuales se

[20] El sistema de ecuaciones no lineales al cual se hace referencia, es el sistema de ecuaciones 2-1 que aparece en el capítulo 2 apartado 2.3 de este documento.

usan para construir y evaluar el sistema de ecuaciones no lineales antes mencionado (*FNF1* y *FNF2*), y las cuatro restantes para construir y evaluar las derivadas parciales con respecto a *λ1* y *λ2* del mismo sistema.(*FNF1L1*, *FNF1L2*, *FNF2L1* y *FNF2L2*).

Sólo el procedimiento *Paso4Calcular* es sustancialmente nuevo o diferente (y sus seis subprocedimientos tipo función), es decir, no se parece a algún otro procedimiento que se haya descrito, es por ello que será el único que se exponga en este apartado.

Antes de presentar el procedimiento *Paso4Calcular*, resulta conveniente hacer un pequeño paréntesis para explicar el método numérico (método de Newton) que emplea para dar solución al sistema de ecuaciones no lineales antes citado.

4.3.1 Método de Newton

El método de Newton es un método numérico eficiente para encontrar la solución de un sistema de ecuaciones no lineales. Se comienza con las formas:

$$f(x,y)=0,\quad g(x,y)=0$$

Sea $x=r$, $y=s$ una raíz, desarróllense ambas funciones como una serie de Taylor, alrededor del punto (x_1,y_1) en términos de $(r-x_1)$, $(s-y_1)$ en donde (x_1,y_1) es un punto cercano a la raíz:

$$f(r,s)=0=f(x_1,y_1)+f_x(x_1,y_1)(r-x_1)+f_y(x_1,y_1)(s-y_1)+.....$$

$$g(r,s)=0=g(x_1,y_1)+g_x(x_1,y_1)(r-x_1)+g_y(x_1,y_1)(s-y_1)+.....$$

En las ecuaciones anteriores, la notación de subíndices designa derivadas parciales. Si (x_1,y_1) está lo suficientemente cercana a (r,s) entonces se puede truncar después de los primeros términos derivados, y resolver para las incógnitas $(r-x_1)$, $(s-y_1)$. Se presentará la solución en forma de determinante en el siguiente par de ecuaciones. Debido a que se ha truncado la serie, sólo se obtiene una aproximación:

$$r-x_1=\frac{\begin{vmatrix}-f & f_y\\ -g & g_y\end{vmatrix}}{\begin{vmatrix}f_x & f_y\\ g_x & g_y\end{vmatrix}}\qquad\qquad s-y_1=\frac{\begin{vmatrix}f_x & -f\\ g_x & -g\end{vmatrix}}{\begin{vmatrix}f_x & f_y\\ g_x & g_y\end{vmatrix}}$$

Nota: Todas las funciones son evaluadas en (x_1,y_1).

Los segundos miembros del par de ecuaciones anteriores son los incrementos a sumar a x_1, y_1, para obtener una mejor estimación de la raíz (r, s). La extensión a más de dos ecuaciones simultáneas es directa en principio, pero el esfuerzo pronto se hace enorme.

Obsérvese que se ha reducido el problema de resolver un sistema de dos ecuaciones no lineales, a resolver un sistema de dos ecuaciones lineales. Las incógnitas se mejoran en cada variable estimada.

El método de Newton tiene la ventaja de converger cuadráticamente, al menos cuando está cercano a la raíz, pero resulta costoso en términos de evaluaciones de función. Para el anterior sistema de 2×2, hay seis evaluaciones de función en cada paso, mientras que para un sistema de 3×3 hay doce.

Ahora que ya se conoce el método de Newton será factible comprender el procedimiento *Paso4Calcular*, en el que se calculan los lotes óptimos (Q^*_i) bajo ambas restricciones (*Presupuesto y Espacio*), es decir, se deducen dos multiplicadores de Lagrange ($\lambda 1$ y $\lambda 2$), y con ellos se calculan y despliegan los óptimos restringidos y el costo total anual promedio del inventario (*K(Q)*) relacionado. A continuación se presenta dicho procedimiento:

- Procedimiento privado (tipo *Sub*) *Paso4Calcular*.

```
Private Sub Paso4Calcular()

KT = 0
Sum = 0

L1 = 0
L2 = 0


10:
varAntes = Timer
L1N = L1 + (((FNF2(L1, L2) * FNF1L2(L1, L2)) - (FNF1(L1, L2) * FNF2L2(L1, L2))) /
((FNF1L1(L1, L2) * FNF2L2(L1, L2)) - (FNF2L1(L1, L2) * FNF1L2(L1, L2))))
L2N = L2 + (((FNF2L1(L1, L2) * FNF1(L1, L2)) - (FNF1L1(L1, L2) * FNF2(L1, L2))) /
((FNF1L1(L1, L2) * FNF2L2(L1, L2)) - (FNF2L1(L1, L2) * FNF1L2(L1, L2))))
varDespues = Timer
Dif = varDespues - varAntes
Sum = Sum + Dif
If Sum >= 5 Then
Sum = 0
Resp = MsgBox("El análisis tomará algunos segundos más, ¿Desea continuar?", vbYesNo)
  If Resp = 7 Then Exit Sub
End If
If (L1N = L1) And (L2N = L2) Then GoTo 20
```

```
L1 = L1N
L2 = L2N
GoTo 10
20:
Lambda1 = L1N
Lambda2 = L2N
```

- Esta rutina se apoya en el método de Newton para encontrar las dos raíces ($\lambda 1$ y $\lambda 2$) del Sist. de Ec. 2-1, que aparece en el capitulo 2 apartado 2.3 de este documento.
- En caso de que la rutina tarde 5 segundos o más en encontrar las dos raíces, se le preguntará al usuario si desea continuar o interrumpir el análisis (la misma pregunta se repetirá 5 segundos después si el análisis aun no ha concluido, y así sucesivamente).
- Las iteraciones se detienen en el momento que las nuevas estimaciones de las raíces tienen el mismo valor de las estimaciones previas, es decir, cuando el método de Newton converge por completo.
- En los procedimientos estándar tipo función *FNF1* y *FNF2* se construye y evalúa el sistema de ecuaciones no lineales antes mencionado (Sist. de Ec. 2-1), y en las funciones *FNF1L1*, *FNF1L2*, *FNF2L1* y *FNF2L2* se construyen y evalúan las derivadas parciales con respecto a $\lambda 1$ y $\lambda 2$ del mismo sistema.

```
With MSFlexGrid1

Text4.Text = CCur(Lambda1)
Text7.Text = CCur(Lambda2)
```

- Este par de sentencias permiten desplegar el contenido de las variables $\lambda 1$ y $\lambda 2$ (*Lambda1* y *Lambda2*) en las cajas de texto *Text4* y *Text7*, después de convertir el contenido de ambas variables al tipo de dato *Currency*[21], mediante la función de conversión de tipo de dato *CCur()*. La conversión tiene por objeto reducir la precisión en los valores de $\lambda 1$ y $\lambda 2$, ya que con la precisión por omisión (tipo de dato *Variant*) no se visualizan por completo en las cajas de texto *Text4* y *Text7*.

```
For j = 1 To n
   .Row = j
   For K = 1 To 5
      .Col = K
        Select Case K
          Case 1: A = .Text
          Case 2: D = .Text
          Case 3: Cm = .Text
          Case 4: Fm = .Text
```

- El código restante permite calcular y desplegar los óptimos restringidos (Q^*_i) y el costo total anual promedio del inventario ($K(Q)$) relacionado.

[21] Las variables del tipo *Currency* almacenan sólo números con punto decimal fijo (8 bytes).

```
          Case 5:
          Qop = Sqr((2 * A * D) / ((i * Cm) + (2 * Lambda1 * Cm) + (2 * Lambda2 * Fm)))
          .Text = CSng(Qop)
          Kp = (Cm * D) + ((A * D) / Qop) + ((Cm * i) * (Qop / 2))
        End Select
   Next K
   KT = KT + Kp
Next j

Text5.Text = KT

End With



End Sub
```

En la función *FNF1* se construye y evalúa la primera ecuación del sistema no lineal al cual se ha venido haciendo referencia (Sist. de Ec. 2-1 que aparece en el capitulo 2 apartado 2.3 de este documento). Dicha función es:

```
Public Function FNF1(Lambda1 As Variant, Lambda2 As Variant)
S = 0
With MSFlexGrid1
For j = 1 To n
   .Row = j
   For K = 1 To 4
      .Col = K
        Select Case K
          Case 1: A = .Text
          Case 2: D = .Text
          Case 3: Cm = .Text
          Case 4: Fm = .Text
          End Select
   Next K
   T = (Sqr((2 * A * D) / ((i * Cm) + ((2) * ((Lambda1 * Cm) + (Lambda2 * Fm)))))) *
(Cm)
   S = S + T
Next j

FNF1 = S - C

End With
```

```
End Function
```

En la función *FNF2* se construye y evalúa la segunda ecuación del sistema no lineal. Dicha función es:

```
Public Function FNF2(Lambda1 As Variant, Lambda2 As Variant)
S = 0
With MSFlexGrid1
For j = 1 To n
   .Row = j
   For K = 1 To 4
      .Col = K
        Select Case K
          Case 1: A = .Text
          Case 2: D = .Text
          Case 3: Cm = .Text
          Case 4: Fm = .Text
          End Select
   Next K
   T = (Sqr((2 * A * D) / ((i * Cm) + ((2) * ((Lambda1 * Cm) + (Lambda2 * Fm)))))) *
(Fm)
   S = S + T
Next j

FNF2 = S - F

End With

End Function
```

En la función *FNF1L1* se construye y evalúa la derivada parcial con respecto a $\lambda 1$ de la primera ecuación del sistema no lineal. Dicha función es:

```
Public Function FNF1L1(Lambda1 As Variant, Lambda2 As Variant)
S = 0
With MSFlexGrid1
For j = 1 To n
   .Row = j
   For K = 1 To 4
      .Col = K
        Select Case K
```

```
        Case 1: A = .Text
        Case 2: D = .Text
        Case 3: Cm = .Text
        Case 4: Fm = .Text
        End Select
   Next K
   T = (Cm ^ 2) * ((2 * A * D) ^ (1 / 2)) * (((i * Cm) + ((2) * ((Lambda1 * Cm) +
(Lambda2 * Fm)))) ^ (-3 / 2))
   S = S + T
Next j

FNF1L1 = -S

End With

End Function
```

En la función *FNF1L2* se construye y evalúa la derivada parcial con respecto a $\lambda 2$ de la primera ecuación del sistema no lineal. Dicha función es:

```
Public Function FNF1L2(Lambda1 As Variant, Lambda2 As Variant)
S = 0
With MSFlexGrid1
For j = 1 To n
   .Row = j
   For K = 1 To 4
      .Col = K
        Select Case K
          Case 1: A = .Text
          Case 2: D = .Text
          Case 3: Cm = .Text
          Case 4: Fm = .Text
          End Select
   Next K
   T = (Cm * Fm) * ((2 * A * D) ^ (1 / 2)) * (((i * Cm) + ((2) * ((Lambda1 * Cm) +
(Lambda2 * Fm)))) ^ (-3 / 2))
   S = S + T
Next j

FNF1L2 = -S

End With
```

```
End Function
```

En la función *FNF2L1* se construye y evalúa la derivada parcial con respecto a $\lambda 1$ de la segunda ecuación del sistema no lineal. Dicha función es:

```
Public Function FNF2L1(Lambda1 As Variant, Lambda2 As Variant)
S = 0
With MSFlexGrid1
For j = 1 To n
   .Row = j
   For K = 1 To 4
      .Col = K
        Select Case K
          Case 1: A = .Text
          Case 2: D = .Text
          Case 3: Cm = .Text
          Case 4: Fm = .Text
          End Select
   Next K
   T = (Cm * Fm) * ((2 * A * D) ^ (1 / 2)) * (((i * Cm) + ((2) * ((Lambda1 * Cm) +
(Lambda2 * Fm)))) ^ (-3 / 2))
   S = S + T
Next j

FNF2L1 = -S

End With

End Function
```

En la función *FNF2L2* se construye y evalúa la derivada parcial con respecto a $\lambda 2$ de la segunda ecuación del sistema no lineal. Dicha función es:

```
Public Function FNF2L2(Lambda1 As Variant, Lambda2 As Variant)
S = 0
With MSFlexGrid1
For j = 1 To n
   .Row = j
   For K = 1 To 4
      .Col = K
        Select Case K
```

```
        Case 1: A = .Text
        Case 2: D = .Text
        Case 3: Cm = .Text
        Case 4: Fm = .Text
        End Select
    Next K
    T = (Fm ^ 2) * ((2 * A * D) ^ (1 / 2)) * (((i * Cm) + ((2) * ((Lambda1 * Cm) +
(Lambda2 * Fm)))) ^ (-3 / 2))
    S = S + T
Next j

FNF2L2 = -S

End With

End Function
```

4.4 Carga y descarga de la interfaz

Como en el caso de *Form1* y *Form2*, solo hace falta presentar un par de procedimientos que permitan cargar y descargar al formulario *Form3* para concluir con su desarrollo.

El primero de ellos permite cargar a *Form3* desde la interfaz principal de usuario (*MDIForm1*) cuando se hace *clic* en la orden *Presupuesto y Espacio* del menú *Análisis con Restricción de.*. El nombre de tal procedimiento es *MnuPyE_Click*, y es conducido por el evento *Click* de *MnuPyE*. Se presenta a continuación:

```
Private Sub MnuPyE_Click()
Load Form3
End Sub
```

También es posible disparar el procedimiento anterior (desde *MDIForm1*) si se hace *clic* en el botón de la barra de herramientas etiquetado con la leyenda *P y E*, ya que se encuentran vinculados (el botón y *MnuPyE_Click*) gracias al procedimiento *Toolbar1_ButtonClick* del apartado 1.6.3 de el presente capítulo.

El segundo procedimiento permite descargar el formulario *Form3* cuando el usuario hace *clic* en el botón de comando (*Command2*) de dicho formulario que tiene la etiqueta *Salir*. El procedimiento es denominado *Command2_Click*, y es conducido por el evento *Click* de *Command2*. Se presenta a continuación:

```
Private Sub Command2_Click()
Unload Form3
End Sub
```

Con el desarrollo de los apartados 2, 3 y 4 de este capítulo, se ha dado respuesta al objetivo central del presente documento, el cual es: "*desarrollar un software capaz de analizar el problema del modelo de inventarios de artículos múltiples con restricción de recursos (Presupuesto y Espacio)"*. Lo anterior justifica la afirmación de que aquí termina la parte cardinal de la tesis, y empieza el desarrollo de herramientas de soporte e integración de software para el análisis de modelos generales de inventarios en **AIRRAM**.

5 SOPORTE PARA LAS VENTANAS DE ANÁLISIS

Es posible que el usuario final de la aplicación (*AIRRAM*) necesite almacenar y recuperar información (*datos de entrada* e *información de salida* que se recolecta y genera en *Form1*, *Form2* y *Form3*) de una sesión a otra, o tal vez imprimirla. Para ello se valdrá de las órdenes del menú *Archivo* que se encuentra en la interfaz principal de usuario (*MDIForm1*). La estructura del menú *Archivo* es:

- Archivo
  - Abrir... Ctrl+A
  - Guardar como... Ctrl+G
  - Imprimir Ctrl+I
  - Salir Ctrl+S

También es posible que se necesite editar las celdas de la tabla *MSFlexGrid1* que se localiza en cada interfaz de análisis (*Form1*, *Form2* y *Form3*), mediante las órdenes del menú *Editar Tabla* que se encuentra en *MDIForm1*. La información en las cajas de texto se edita con el menú contextual flotante de Windows. La estructura del menú *Editar Tabla* es:

- Editar Tabla
  - Copiar Ctrl+C
  - Pegar Crtl+P
  - Borrar Del

Antes de escribir el código que permitirá funcionar a los menús antes citados, es necesario declarar una variable y desarrollar algunos procedimientos más en los formularios *Form1*, *Form2* y *Form3*.

5.1 Variable Bandera

Resulta al menos curioso dedicar un apartado completo sólo a la declaración de una variable (*Bandera*), pero no es cualquier variable, es una variable que va a determinar cual de los formularios (*Form1*, *Form2* o *Form3*) esta cargado y activo. Imagínese que el usuario final de la aplicación desea guardar los datos de un determinado formulario, pero, ¿Cómo puede saber *AIRRAM* donde leer esos datos?, sería molesto para el usuario indicarle explícitamente que los datos se encuentran en *Form1*, *Form2* o *Form3*. Es por ello que tal labor se confía a la variable *Bandera*.

Bandera puede almacenar los números 1, 2, 3 y 0; cada número indica lo siguiente:

- Si *Bandera* = 1, el formulario *Form1* está cargado y activo.
- Si *Bandera* = 2, el formulario *Form2* está cargado y activo.
- Si *Bandera* = 3, el formulario *Form3* está cargado y activo.
- Si *Bandera* = 0, ningún formulario de los tres anteriores está activo, aunque si pueden estar cargados.

Cuando en un módulo del tipo formulario se declara una variable (incluso *Global*), esta variable se destruye cuando el formulario se cierra (descarga), y *Bandera* no se puede destruir. Luego entonces habría que declararla en un módulo que no se cierre mientras la aplicación esta corriendo, y no dependa del formulario principal.

Existe un módulo que cumple con los requerimientos anteriores, el estándar, sólo hay que insertarlo y declarar a la variable *Bandera* como global (el nombre de código del módulo estándar será *Module1*). Con la siguiente línea de código se declara a *Bandera* como una variable global en el modulo estándar *Module1*.

```
Public Bandera
```

5.2 Habilitación y bloqueo de elementos en las barras de menús y herramientas

El que ciertas órdenes (y sus botones asociados) de los menús *Archivo* y *Editar Tabla* estén habilitadas o no, va a depender de la condición de las ventanas o formularios de análisis (*Form1*, *Form2* y *Form3*) y sus respectivas tablas (*MSFlexGrid1*).

Si en un determinado momento no se encuentra cargada y activa alguna interfaz de análisis (*Form1*, *Form2* o *Form3*), no tiene caso que estén habilitadas las órdenes (y sus botones asociados) que permiten guardar e imprimir la información de dichas interfaces, puesto que no existe la fuente de tal información. Por otro lado, si no está seleccionada parte o toda la tabla (*MSFlexGrid1*) de algún formulario de análisis, tampoco tiene caso que estén habilitadas las órdenes que permiten editar la información de dichas tablas, puesto que no se ha seleccionado la fuente o el depósito de tal información.

El primer procedimiento de este apartado tiene que ser escrito en la interfaz principal de usuario (*MDIForm1*), ya que se va a disparar cuando se cargue o abra dicha interfaz. El procedimiento va a permitir inhabilitar o bloquear las órdenes *Guardar como...* e *Imprimir* (y sus botones asociados) del menú *Archivo*, así como todas las órdenes del menú *Editar Tabla*. Se bloquean todas estas órdenes porque no tienen donde aplicar cuando se carga *MDIForm1*, es decir, instantes después de que se carga *MDIForm1* todavía no se ha abierto algún formulario de análisis (*Form1*, *Form2* o *Form3*), por lo que las órdenes antes citadas no tienen donde adquirir y depositar la información que manipulan y deben estar bloqueadas. El procedimiento del cual se ha venido hablando en este párrafo lleva el nombre de *MDIForm_Load*, y se presenta a continuación:

```
Private Sub MDIForm_Load()

MDIForm1.MnuGuardarComo.Enabled = False
Toolbar1.Buttons(2).Enabled = False
MDIForm1.MnuImprimir.Enabled = False
```

```
Toolbar1.Buttons(3).Enabled = False

MDIForm1.MnuCopiar.Enabled = False
MDIForm1.MnuBorrar.Enabled = False
MDIForm1.MnuPegar.Enabled = False

Picture1.Left = MDIForm1.Width - 1275
```

- Esta línea de código se explicó en el procedimiento *MDIForm_Load* del apartado 1.8 del presente capítulo.

```
End Sub
```

Los siguientes cinco procedimientos se van a repetir en cada formulario de análisis (*Form1*, *Form2* y *Form3*), ya que las órdenes (y sus botones asociados) que van a habilitar o bloquear toman y vacían información en cada uno de ellos.

Con el siguiente procedimiento (*Form_Activate*) se habilitan las órdenes *Guardar como...* e *Imprimir* (y sus botones asociados) del menú *Archivo* cuando se carga y activa algún formulario de análisis (*Form1*, *Form2* o *Form3*). Si algún formulario de análisis está activo (y por lo tanto cargado) las órdenes antes citadas tienen donde leer la información que manipulan, y por lo tanto resulta viable que estén habilitadas. En este procedimiento la variable *Bandera* puede almacenar los números 1, 2 o 3, y el número que almacene dependerá de que formulario está cargado y activo (*Form1*, *Form2* o *Form3*). A continuación se presenta *Form_Activate*:

```
Private Sub Form_Activate()

MDIForm1.MnuGuardarComo.Enabled = True
MDIForm1.Toolbar1.Buttons(2).Enabled = True


MDIForm1.MnuImprimir.Enabled = True
MDIForm1.Toolbar1.Buttons(3).Enabled = True

Module1.Bandera = 1

End Sub
```

En el procedimiento *Form_Deactivate* se bloquean las órdenes *Guardar como...* e *Imprimir* (y sus botones asociados) del menú *Archivo* cuando se desactiva algún formulario de análisis (*Form1*, *Form2* o *Form3*). Si algún formulario de análisis se desactiva y no se

activa otro inmediatamente, las órdenes antes citadas no tienen donde leer la información que manipulan, y por lo tanto resultaría inútil que estuviesen habilitadas. En este procedimiento la variable *Bandera* almacena el número cero, lo que indica que el formulario (*Form1*, *Form2* o *Form3*) en el cual se programa el procedimiento está inactivo. A continuación se presenta *Form_Deactivate*:

```
Private Sub Form_Deactivate()

MDIForm1.MnuGuardarComo.Enabled = False
MDIForm1.Toolbar1.Buttons(2).Enabled = False

MDIForm1.MnuImprimir.Enabled = False
MDIForm1.Toolbar1.Buttons(3).Enabled = False

Module1.Bandera = 0

End Sub
```

En el procedimiento *Form_Unload* se bloquean las órdenes *Guardar como...* e *Imprimir* (y sus botones asociados) del menú *Archivo*, así como todas las órdenes del menú *Editar Tabla* cuando se descarga algún formulario de análisis (*Form1*, *Form2* o *Form3*). Si algún formulario de análisis se descarga y no se carga y activa otro inmediatamente, las órdenes antes citadas no tienen donde leer y depositar la información que manipulan, por lo que resultaría inútil que estuviesen habilitadas. En este procedimiento la variable *Bandera* almacena el número cero, lo que indica que el formulario (*Form1*, *Form2* o *Form3*) en el cual se programa el procedimiento se descarga instantes después de que *Form_Unload* es disparado. A continuación se presenta *Form_Unload*:

```
Private Sub Form_Unload(Cancel As Integer)

MDIForm1.MnuGuardarComo.Enabled = False
MDIForm1.Toolbar1.Buttons(2).Enabled = False


MDIForm1.MnuImprimir.Enabled = False
MDIForm1.Toolbar1.Buttons(3).Enabled = False


Module1.Bandera = 0

MDIForm1.MnuCopiar.Enabled = False
MDIForm1.MnuBorrar.Enabled = False
MDIForm1.MnuPegar.Enabled = False
```

```
End Sub
```

En el procedimiento *MSFlexGrid1_GotFocus* se habilitan todas las órdenes del menú *Editar Tabla* cuando se selecciona parte o toda la tabla (*MSFlexGrid1*) de algún formulario de análisis (*Form1*, *Form2* o *Form3*). Si se selecciona parte o toda la tabla de algún formulario de análisis las órdenes antes citadas tienen donde leer y depositar la información que manipulan, y por lo tanto resulta viable que estén habilitadas. A continuación se presenta *MSFlexGrid1_GotFocus*:

```
Private Sub MSFlexGrid1_GotFocus()

MDIForm1.MnuCopiar.Enabled = True
MDIForm1.MnuBorrar.Enabled = True
MDIForm1.MnuPegar.Enabled = True

End Sub
```

En el procedimiento *MSFlexGrid1_LostFocus* se bloquean todas las órdenes del menú *Editar Tabla* cuando se pierde la condición de selección de parte o toda la tabla (*MSFlexGrid1*) de algún formulario de análisis (*Form1*, *Form2* o *Form3*). Si se pierde la condición de selección de parte o toda la tabla de algún formulario de análisis y no se recupera inmediatamente, las órdenes antes citadas no tienen donde leer y depositar la información que manipulan, por lo que resultaría inútil que estuviesen habilitadas. A continuación se presenta el procedimiento *MSFlexGrid1_LostFocus*:

```
Private Sub MSFlexGrid1_LostFocus()

MDIForm1.MnuCopiar.Enabled = False
MDIForm1.MnuBorrar.Enabled = False
MDIForm1.MnuPegar.Enabled = False

End Sub
```

5.3 Menú Archivo

La mayoría de los procedimientos del menú *Archivo* emplearán intensivamente un nuevo tipo control, el control *CommonDialog* (diálogo común), el cual permite visualizar las cajas de diálogo más comúnmente empleadas en el diseño de aplicaciones, tales como *Abrir* (*Open*), *Guardar como* (*Save As*), *Imprimir* (*Print*), *Fuente* (*Font*) y *Color* (*Color*). Debido

a que *CommonDialog* no se encuentra dentro de los controles intrínsecos incluidos en la *caja de herramientas* de Visual Basic, hay que añadirlo ejecutando la orden *Componentes* del menú *Proyecto* y agregar el componente *ActiveX* denominado *Microsoft Common Dialog Control 6.0.*

Después de insertar el control *CommonDialog* en la interfaz principal de usuario (*MDIForm1*) ya podrá ser utilizado en los procedimientos del menú *Archivo*. El nombre de código del control *CommonDialog* será *CommonDialog1*.

Todos los procedimientos que permitirán funcionar a las órdenes de soporte (órdenes de los menús *Archivo* y *Editar Tabla*) para las ventanas de análisis (*Form1*, *Form2* y *Form3*) serán escritos en la interfaz principal de usuario (*MDIForm1*).

5.3.1 Abrir...

Cuando el usuario final haga clic en la orden *Abrir...* (o en su botón asociado) del menú *Archivo* se disparará el procedimiento *MnuAbrir_Click*. Este procedimiento abre ficheros de datos generados en *AIRRAM* (**.AIRR1*, **.AIRR2* o **.AIRR3*), y deposita la información que contienen en su respectivo formulario de análisis (*Form1*, *Form2* o *Form3*).

Los ficheros de datos generados en *AIRRAM* pueden tener tres diferentes extensiones; cada extensión indica lo siguiente:

- Si el fichero tiene una extensión **.AIRR1*, es un fichero que fue creado a partir de información perteneciente a un formulario de análisis *Form1*, y por lo tanto, en el momento en que se abra, su información debe de ser depositada en un formulario *Form1*, en otras palabras, un fichero con extensión **.AIRR1* corresponde a un formulario *Form1*.
- Un fichero con extensión **.AIRR2* corresponde a un formulario *Form2*.
- Un fichero con extensión **.AIRR3* corresponde a un formulario *Form3*.

A continuación se presenta el procedimiento *MnuAbrir_Click*:

```
Private Sub MnuAbrir_Click()

On Error GoTo ManipularErrorAbrir
```

- Esta sentencia intercepta cualquier tipo de error que se produzca en el procedimiento. Y la etiqueta *ManipularErrorAbrir* identifica la primera línea de la rutina que manipula el error.

```
CommonDialog1.CancelError = True
```

- Aquí se genera un error interceptable justo cuando el usuario final pulsa el botón *Cancelar* (en tiempo de ejecución) de la caja de diálogo *CommonDialog1*.

```
CommonDialog1.Filter = "Restricción de Presupuesto (*.AIRR1)|*.AIRR1|" & _
                       "Restricción de Espacio (*.AIRR2)|*.AIRR2|" & _
                       "Restricción de Presupuesto y Espacio (*.AIRR3)|*.AIRR3|" & _
                       "Todos (*.*)|*.*"
```

- Con esta rutina se especifica la lista de tipos de ficheros que se desea abrir.

```
CommonDialog1.FilterIndex = 1
```

- Esta sentencia asigna el filtro por omisión de los de la lista especificada anteriormente, es decir, asigna el tipo de fichero por omisión que se desea abrir (el primero es el 1).

```
CommonDialog1.ShowOpen
```

- Con esta sentencia se visualiza la caja de diálogo *Abrir* y se detiene el procedimiento. El procedimiento se reanuda hasta que el usuario final selecciona un fichero y pulsa el botón *Abrir*.

```
If CommonDialog1.FileTitle Like "*.AIRR1" Then
AIRR1 = CommonDialog1.FileName
AbrirFichero1
End If
```

- Si el fichero elegido corresponde a un formulario de análisis *Form1*, se asigna el camino y el nombre del fichero a la variable global *AIRR1*, y se dispara el procedimiento *AbrirFichero1*. Este procedimiento abre el fichero especificado en la variable *AIRR1* para *acceso secuencial*[22], y deposita la información que contiene en el formulario de análisis *Form1*.

```
If CommonDialog1.FileTitle Like "*.AIRR2" Then
AIRR2 = CommonDialog1.FileName
```

[22] Visual Basic permite utilizar tres tipos diferentes de acceso a ficheros de datos: secuencial, aleatorio y binario. Un fichero abierto para *acceso secuencial* es un fichero que puede contener registros de cualquier longitud, y como su nombre lo dice los registros se guardan y leen secuenciálmente.

```
AbrirFichero2
End If
```

- Si el fichero elegido corresponde a un formulario de análisis *Form2*, se asigna el camino y el nombre del fichero a la variable global *AIRR2*, y se dispara el procedimiento *AbrirFichero2*. Este procedimiento abre el fichero especificado en la variable *AIRR2* para *acceso secuencial*, y deposita la información que contiene en el formulario de análisis *Form2*.

```
If CommonDialog1.FileTitle Like "*.AIRR3" Then
AIRR3 = CommonDialog1.FileName
AbrirFichero3
End If
```

- Si el fichero elegido corresponde a un formulario de análisis *Form3*, se asigna el camino y el nombre del fichero a la variable global *AIRR3*, y se dispara el procedimiento *AbrirFichero3*. Este procedimiento abre el fichero especificado en la variable *AIRR3* para *acceso secuencial*, y deposita la información que contiene en el formulario de análisis *Form3*.

- Las variables *AIRR1*, *AIRR2* y *AIRR3* están definidas como variables globales del tipo *String*[23] en la sección de declaraciones del módulo (*MDIForm1*), ya que almacenan cadenas y son empleadas por varios procedimientos.

```
SalirAbrir:
  Exit Sub
```

- Si en el procedimiento se origina algún error en tiempo de ejecución se manipula en esta rutina.

```
ManipularErrorAbrir:
  If Err.Number = cdlCancel Then Exit Sub
  MsgBox Err.Description
  Resume SalirAbrir


End Sub
```

- En caso de que el error corresponda al que se genera cuando el usuario final pulsa el botón *Cancelar* de la caja de diálogo *Abrir* (*CommonDialog1*) se finaliza el procedimiento, en caso contrario, se visualiza una caja de diálogo con la descripción del error que se produjo y se finaliza el procedimiento.

---

[23] Las variables del tipo *String* guardan cadenas de caracteres de longitud variable (10 bytes + 1 byte por caracter).

El procedimiento anterior a su vez dispara otros tres, el primero de ellos es *AbrirFichero1*, el cual abre el fichero especificado en la variable *AIRR1* para *acceso secuencial*, y deposita la información que contiene en el formulario de análisis *Form1*. A continuación se presenta *AbrirFichero1*:

```
Private Sub AbrirFichero1()

Dim Cadena As String
Dim NumFichero As Integer

NumFichero = FreeFile
```

- El fichero especificado en la variable *AIRR1* tiene que ser referenciado con un número entre 1 y 511 al ser abierto. La función *FreeFile* obtiene de forma segura, un número de fichero libre y lo asigna a la variable *NumFichero*.

```
On Error GoTo RutinaErrorAbrir
```

- Esta sentencia intercepta cualquier tipo de error que se produzca en el procedimiento. Y la etiqueta *RutinaErrorAbrir* identifica la primera línea de la rutina que manipula el error.

```
Open AIRR1 For Input As #NumFichero
```

- Con esta sentencia se abre el fichero especificado en la variable *AIRR1* para *acceso secuencial* en modo *Input* (lectura), con el número de fichero asignado a la variable *NumFichero*.

```
Load Form1
Form1.SetFocus
```

- Se carga el formulario de análisis *Form1* y se enfoca (se activa).

```
  Input #NumFichero, a
  Form1.Text1.Text = a
  Input #NumFichero, a
  Form1.Text2.Text = a
  Input #NumFichero, a
  Form1.Text3.Text = a
  Input #NumFichero, a
  Form1.Text4.Text = a
  Input #NumFichero, a
  Form1.Text5.Text = a
```

- En esta rutina se leen los primeros cinco registros del fichero, y se depositan en las cajas de texto de *Form1*.

```
a = Form1.Text3.Text
If IsNumeric(a) And (a >= 1) And (a <> "") Then
  If (a / CInt(a)) = 1 Then
    With Form1.MSFlexGrid1
      For j = 1 To Form1.Text3.Text
        .Row = j
        For K = 1 To 4
          .Col = K
          Input #NumFichero, a
          .Text = a
        Next K
      Next j
    End With
  End If
End If
```

- Con esta rutina se leen los registros restantes, y se depositan en la tabla *MSFlexGrid1* de *Form1*.

```
  Close #NumFichero
```

- Se cierra el fichero.

```
SalirAbrir:
  Exit Sub
RutinaErrorAbrir:
  MsgBox "Error al abrir el fichero", 48
  Resume SalirAbrir

End Sub
```

- Si en el procedimiento se origina algún error en tiempo de ejecución se manipula en esta rutina.

A excepción de los formularios en donde depositan la información del fichero elegido , los procedimientos *AbrirFichero2* y *AbrirFichero3* son idénticos al anterior, por lo que no tiene caso que se expliquen.

El código completo de los procedimientos relacionados a la orden *Abrir...* del menú *Archivo* se encuentra en los **Anexos** de este documento.

5.3.2 Guardar como...

Cuando el usuario final haga clic en la orden *Guardar como...* (o en su botón asociado) del menú *Archivo* se disparará el procedimiento *MnuGuardarComo_Click*. Este procedimiento escribe la información que contienen los formularios de análisis de AIRRAM (*Form1*, *Form2* o *Form3*), en su respectivo fichero de datos (**.AIRR1*, **.AIRR2* o **.AIRR3*).

Antes de escribir o guardar información en un fichero tiene que ser abierto, ya sea en modo *Output* o *Append*. En el modo *Output* si el fichero no existe, se crea, y si existe, se destruye y se crea de nuevo. Mientras que en el modo *Append* si el fichero ya existe, no se destruye. El modo *Append* permite añadir la nueva información al final del fichero.

A continuación se presenta el procedimiento *MnuGuardarComo_Click*:

```
Private Sub MnuGuardarComo_Click()

If Module1.Bandera = 1 Then

On Error GoTo ManipularErrorGuardar

  CommonDialog1.CancelError = True

  CommonDialog1.Filter = "Restricción de Presupuesto (*.AIRR1)|*.AIRR1|" & _
                         "Todos (*.*)|*.*"

  CommonDialog1.FilterIndex = 1

  CommonDialog1.ShowSave

  AIRR1 = CommonDialog1.FileName

  GuardarFichero1

End If
```

- Si la variable *Bandera* almacena el número 1, es decir, el formulario *Form1* está cargado y activo, se realizan las siguientes tareas:
1. La sentencia *On Error GoTo* intercepta cualquier tipo de error que se produzca en el procedimiento. Y la etiqueta *ManipularErrorGuardar* identifica la primera línea de la rutina que manipula el error.
2. Aquí se genera un error interceptable justo cuando el usuario final pulsa el botón *Cancelar* (en tiempo de ejecución) de la caja de diálogo *CommonDialog1*.

3. Con esta rutina se especifica la lista de tipos de ficheros que se desea guardar.
4. Esta sentencia asigna el filtro por omisión de los de la lista especificada anteriormente, es decir, asigna el tipo de fichero por omisión que se desea guardar (**.AIRR1*).

5. Con esta sentencia se visualiza la caja de diálogo *Guardar como* y se detiene el procedimiento. El procedimiento se reanuda hasta que el usuario final escribe un nombre de fichero, y pulsa el botón *Guardar*.
6. Por último, se asigna el camino y el nombre del fichero a la variable global *AIRR1*, y se dispara el procedimiento *GuardarFichero1*. Este procedimiento abre el fichero especificado en la variable *AIRR1* para *acceso secuencial*, y escribe en él la información que contiene el formulario de análisis *Form1*.

```
If Module1.Bandera = 2 Then

On Error GoTo ManipularErrorGuardar

  CommonDialog1.CancelError = True

  CommonDialog1.Filter = "Restricción de Espacio (*.AIRR2)|*.AIRR2|" & _
                         "Todos (*.*)|*.*"

  CommonDialog1.FilterIndex = 1
  '
  CommonDialog1.ShowSave

  AIRR2 = CommonDialog1.FileName

  GuardarFichero2

End If
```

- Si la variable *Bandera* almacena el número 2, es decir, el formulario *Form2* está cargado y activo, se realizan las mismas tareas que en el caso anterior, a excepción de la siguiente:
- Se asigna el camino y el nombre del fichero a la variable global *AIRR2*, y se dispara el procedimiento *GuardarFichero2*. Este procedimiento abre el fichero especificado en la variable *AIRR2* para *acceso secuencial*, y escribe en él la información que contiene el formulario de análisis *Form2*.

```
If Module1.Bandera = 3 Then

On Error GoTo ManipularErrorGuardar

  CommonDialog1.CancelError = True

  CommonDialog1.Filter = "Restricción de Presupuesto y Espacio (*.AIRR3)|*.AIRR3|" &
_"Todos (*.*)|*.*"

  CommonDialog1.FilterIndex = 1

  CommonDialog1.ShowSave

  AIRR3 = CommonDialog1.FileName

  GuardarFichero3

End If
```

- Si la variable *Bandera* almacena el número 3, es decir, el formulario *Form3* está cargado y activo, se realizan las mismas tareas que en el caso anterior, a excepción de la siguiente:
- Se asigna el camino y el nombre del fichero a la variable global *AIRR3*, y se dispara el procedimiento *GuardarFichero3*. Este procedimiento abre el fichero especificado en la variable *AIRR3* para *acceso secuencial*, y escribe en él la información que contiene el formulario de análisis *Form3*.

```
SalirGuardar:
  Exit Sub
ManipularErrorGuardar:
  If Err.Number = cdlCancel Then Exit Sub
  MsgBox Err.Description
  Resume SalirGuardar

End Sub
```

- Si en el procedimiento se origina algún error en tiempo de ejecución se manipula en esta rutina.

El procedimiento anterior a su vez dispara otros tres, el primero de ellos es *GuardarFichero1*, el cual abre el fichero especificado en la variable *AIRR1* para *acceso secuencial*, y escribe en él la información que contiene el formulario de análisis *Form1*. A continuación se presenta *GuardarFichero1*:

```
Private Sub GuardarFichero1()

Dim Cadena As String
Dim NumFichero As Integer

  NumFichero = FreeFile
```

- El fichero especificado en la variable *AIRR1* tiene que ser referenciado con un número entre 1 y 511 al ser abierto. La función *FreeFile* obtiene de forma segura, un número de fichero libre y lo asigna a la variable *NumFichero*.

```
  On Error GoTo RutinaErrorGuardar
```

- Esta sentencia intercepta cualquier tipo de error que se produzca en el procedimiento. Y la etiqueta *RutinaErrorGuardar* identifica la primera línea de la rutina que manipula el error.

```
  Open AIRR1 For Append As #NumFichero
```

- Con esta sentencia se abre el fichero especificado en la variable *AIRR1* para *acceso secuencial* en modo *Append* (escritura), con el número de fichero asignado a la variable *NumFichero*.

```
 If (LOF(NumFichero) <> 0) Then
   Cadena = "El fichero ya existe" & vbCrLf
   Cadena = Cadena & "¿desea sobreescribirlo?"
   If MsgBox(Cadena, 36) = vbYes Then
     Close #NumFichero
     Kill AIRR1
     Open AIRR1 For Output As #NumFichero
   Else
     Close #NumFichero
     Exit Sub
   End If
 End If
```

- Si el fichero no está vacío (su longitud no es cero), mediante una caja de diálogo se informa al usuario que existe, y se le pregunta si desea sobrescribirlo. En caso afirmativo, la sentencia *Kill* borra el contenido del fichero, el cual tiene que estar cerrado, y la sentencia *Open* abre de nuevo el fichero para *acceso secuencial* en modo *Output* (escritura) con el número de fichero asignado a la variable *NumFichero*; en caso contrario, se cierra el fichero y se abandona el procedimiento.

```
   Print #NumFichero, Form1.Text1.Text
   Print #NumFichero, Form1.Text2.Text
   Print #NumFichero, Form1.Text3.Text
   Print #NumFichero, Form1.Text4.Text
   Print #NumFichero, Form1.Text5.Text
```

- En esta rutina se escribe el contenido de las cajas de texto de *Form1* en el fichero.

```
a = Form1.Text3.Text
If IsNumeric(a) And (a >= 1) And (a <> "") Then
 If (a / CInt(a)) = 1 Then
   With Form1.MSFlexGrid1
     For j = 1 To Form1.n
       .Row = j
       For K = 1 To 4
         .Col = K
         Print #NumFichero, .Text
       Next K
     Next j
   End With
 End If
End If
```

- Con esta rutina se escribe el contenido de la tabla *MSFlexGrid1* de *Form1* en el fichero.

```
  Close #NumFichero
```

- Se cierra el fichero.

```
SalirGuardar:
  Exit Sub
RutinaErrorGuardar:
  MsgBox "Error al abrir el fichero", 48
  Resume SalirGuardar

End Sub
```

- Si en el procedimiento se origina algún error en tiempo de ejecución se manipula en esta rutina.

A excepción de los formularios de donde leen la información que escriben en el fichero elegido, los procedimientos *GuardarFichero2* y *GuardarFichero3* son idénticos al anterior, por lo que no tiene caso que se expliquen.

El código completo de los procedimientos relacionados a la orden *Guardar como...* del menú *Archivo* se encuentra en los **Anexos** de este documento.

5.3.3 Imprimir

Cuando el usuario final haga clic en la orden *Imprimir* (o en su botón asociado) del menú *Archivo* se disparará el procedimiento *MnuImprimir_Click*. Este procedimiento imprime los formularios de análisis de *AIRRAM* (*Form1*, *Form2* y *Form3*).

El procedimiento *MnuImprimir_Click* empleará un control *CommonDialog* adicional para visualizar la caja de diálogo *Imprimir* (*Print*). Después de insertar el control en la interfaz principal de usuario (*MDIForm1*) ya podrá ser utilizado en *MnuImprimir_Click*. El nombre de código del control *CommonDialog* será *CommonDialog2*.

A continuación se presenta el procedimiento *MnuImprimir_Click*:

```
Private Sub MnuImprimir_Click()

  On Error GoTo ManipularErrorImprimir
```

- Esta sentencia intercepta cualquier tipo de error que se produzca en el procedimiento. Y la etiqueta *ManipularErrorImprimir* identifica la primera línea de la rutina que manipula el error.

```
CommonDialog2.CancelError = True
```

- Aquí se genera un error interceptable justo cuando el usuario final pulsa el botón *Cancelar* (en tiempo de ejecución) de la caja de diálogo *CommonDialog2*.

```
CommonDialog2.ShowPrinter
```

- Con esta sentencia se visualiza la caja de diálogo *Imprimir* y se detiene el procedimiento. El procedimiento se reanuda hasta que el usuario final fija las propiedades de impresión y pulsa el botón *Aceptar*.

```
    If Module1.Bandera = 1 Then
    Form1.PrintForm
    End If
    If Module1.Bandera = 2 Then
    Form2.PrintForm
    End If
    If Module1.Bandera = 3 Then
    Form3.PrintForm
    End If
```

- Esta rutina imprime el formulario de análisis (*Form1*, *Form2* o *Form3*) que está cargado y activo.

```
SalirImprimir:
  Exit Sub
ManipularErrorImprimir:
  If Err.Number = cdlCancel Then Exit Sub
  MsgBox Err.Description
  Resume SalirImprimir

End Sub
```

- Si en el procedimiento se origina algún error en tiempo de ejecución se manipula en esta rutina.

5.3.4 Salir

Cuando el usuario final haga clic en la orden *Salir* del menú *Archivo* se disparará el procedimiento *MnuSalir_Click*. Este procedimiento cierra la aplicación (*AIRRAM*) mediante la sentencia *End*. Se presenta a continuación:

```
Private Sub MnuSalir_Click()
End
```

End Sub

5.4 Menú Editar Tabla

Las órdenes del menú *Editar Tabla* permiten copiar, borrar y pegar información en las celdas seleccionadas de la tabla *MSFlexGrid1* que se localiza en cada formulario de análisis (*Form1*, *Form2* y *Form3*).

5.4.1 Copiar

Cuando el usuario final haga clic en la orden *Copiar* del menú *Editar Tabla* se disparará el procedimiento *MnuCopiar_Click*. Este procedimiento copia las celdas seleccionadas de la tabla *MSFlexGrid1* en uso (la tabla de *Form1*, *Form2* o *Form3*) en el *Portapapeles*. A continuación se presenta *MnuCopiar_Click*:

```
Private Sub MnuCopiar_Click()

On Error GoTo ManipularError
```

- Esta sentencia intercepta cualquier tipo de error que se produzca en el procedimiento. Y la etiqueta *ManipularError* identifica la primera línea de la rutina que manipula el error.

```
If Module1.Bandera = 1 Then
CopiarTabla1
End If
```

- Si la variable *Bandera* almacena el número 1, es decir, el formulario *Form1* está cargado y activo, se dispara el procedimiento *CopiarTabla1*. Este procedimiento copia las celdas seleccionadas de la tabla *MSFlexGrid1* en uso (en este caso la tabla del formulario de análisis *Form1*) en el *Portapapeles*.

```
If Module1.Bandera = 2 Then
CopiarTabla2
End If
```

- Si la variable *Bandera* almacena el número 2, es decir, el formulario *Form2* está cargado y activo, se dispara el procedimiento *CopiarTabla2*. Este procedimiento copia las celdas seleccionadas de la tabla *MSFlexGrid1* en uso (en este caso la tabla del formulario de análisis *Form2*) en el *Portapapeles*.

```
If Module1.Bandera = 3 Then
CopiarTabla3
End If
```

- Si la variable *Bandera* almacena el número 3, es decir, el formulario *Form3* está cargado y activo, se dispara el procedimiento *CopiarTabla3*. Este procedimiento copia las celdas seleccionadas de la tabla *MSFlexGrid1* en uso (en este caso la tabla del formulario de análisis *Form3*) en el *Portapapeles*.

```
Salir:
  Exit Sub
ManipularError:
  MsgBox Err.Description
  Resume Salir

End Sub
```

- Si en el procedimiento se origina algún error en tiempo de ejecución se manipula en esta rutina.

El procedimiento anterior a su vez dispara otros tres, el primero de ellos es *CopiarTabla1*, el cual copia las celdas seleccionadas de la tabla *MSFlexGrid1* en uso (en este caso la tabla del formulario de análisis *Form1*) en el *Portapapeles*. A continuación se presenta *CopiarTabla1*:

```
Private Sub CopiarTabla1()

With Form1.MSFlexGrid1

  Dim TextoSelec As String

  TextoSelec = .Clip
```

- Con esta sentencia se copia el contenido de las celdas seleccionadas de la tabla *MSFlexGrid1* en la variable *TextoSelec*.

```
  Clipboard.Clear
```

- Aquí se vacía el *Portapapeles*, ya que éste no lo hace automáticamente porque está diseñado para almacenar varias unidades de datos simultáneamente con formatos diferentes.

```
  Clipboard.SetText TextoSelec

.Row = 1
.Col = 1
```

- Esta sentencia asigna el contenido de la variable *TextoSelec* al *Portapapeles*.

```
End With

End Sub
```

A excepción de los formularios de donde adquieren la información que copian en el *Portapapeles*, los procedimientos *CopiarTabla2* y *CopiarTabla3* son idénticos al anterior, por lo que no tiene caso que se expliquen.

El código completo de los procedimientos relacionados a la orden *Copiar* del menú *Editar Tabla* se encuentra en los **Anexos** de este documento.

5.4.2 Pegar

Cuando el usuario final haga clic en la orden *Pegar* del menú *Editar Tabla* se disparará el procedimiento *MnuPegar_Click*. Este procedimiento pega el contenido del *Portapapeles* (que debe ser una celda o un grupo de celdas) en la tabla *MSFlexGrid1* en uso (la tabla de *Form1*, *Form2* o *Form3*). A continuación se presenta *MnuPegar_Click*:

```
Private Sub MnuPegar_Click()

On Error GoTo ManipularError
```

- Esta sentencia intercepta cualquier tipo de error que se produzca en el procedimiento. Y la etiqueta *ManipularError* identifica la primera línea de la rutina que manipula el error.

```
If Module1.Bandera = 1 Then
PegarTabla1
End If
```

- Si la variable *Bandera* almacena el número 1, es decir, el formulario *Form1* está cargado y activo, se dispara el procedimiento *PegarTabla1*. Este procedimiento pega el contenido del *Portapapeles* (que debe ser una celda o un grupo de celdas) en la tabla *MSFlexGrid1* en uso (en este caso la tabla del formulario de análisis *Form1*).

```
If Module1.Bandera = 2 Then
PegarTabla2
End If
```

- Si la variable *Bandera* almacena el número 2, es decir, el formulario *Form2* está cargado y activo, se dispara el procedimiento *PegarTabla2*. Este procedimiento pega el contenido del *Portapapeles* (que debe ser una celda o un grupo de celdas) en la tabla *MSFlexGrid1* en uso (en este caso la tabla del formulario de análisis *Form2*).

```
If Module1.Bandera = 3 Then
PegarTabla3
End If
```

- Si la variable *Bandera* almacena el número 3, es decir, el formulario *Form3* está cargado y activo, se dispara el procedimiento *PegarTabla3*. Este procedimiento pega el contenido del *Portapapeles* (que debe ser una celda o un grupo de celdas) en la tabla *MSFlexGrid1* en uso (en este caso la tabla del formulario de análisis *Form3*).

```
Salir:
  Exit Sub
ManipularError:
  MsgBox Err.Description
  Resume Salir

End Sub
```

- Si en el procedimiento se origina algún error en tiempo de ejecución se manipula en esta rutina.

El procedimiento anterior a su vez dispara otros tres, el primero de ellos es *PegarTabla1*, el cual pega el contenido del *Portapapeles* (que debe ser una celda o un grupo de celdas) en la tabla *MSFlexGrid1* en uso (en este caso la tabla del formulario de análisis *Form1*). A continuación se presenta *PegarTabla1*:

```
Private Sub PegarTabla1()

With Form1.MSFlexGrid1

Original = .Col

Tabla = Clipboard.GetText
```

- Con esta sentencia se copia el contenido del *Portapapeles* (que debe ser una celda o un grupo de celdas) en la variable *Tabla*.

```
For i = 1 To Len(Tabla)
car = Mid(Tabla, i, 1)
   If car Like "[1234567890.]" Then
   celda = celda & car
   If i = Len(Tabla) Then
   .Text = celda
   celda = ""
   car = ""
   Exit Sub
   End If
   Else
   pal = Mid(Tabla, i, 2)
     If pal = vbCrLf Then
     .Text = celda
     celda = ""
     car = ""
        If i = Len(Tabla) - 1 Then
        Exit Sub
        Else
        .Row = .Row + 1
        .Col = Original
        i = i + 1
        GoTo 10
        End If
     End If
     If car = vbCr Then
     .Text = celda
     celda = ""
     car = ""
        If i = Len(Tabla) Then
        Exit Sub
        Else
        .Row = .Row + 1
        .Col = Original
        GoTo 10
        End If
     End If
   .Text = celda
   celda = ""
   .Col = .Col + 1
   car = ""
   End If
10: Next i

End With
```

- Aunque esta rutina luce algo complicada en realidad no lo es tanto, solo vacía el contenido de la variable *Tabla* en las celdas de la tabla *MSFlexGrid1*. Y lo hace de la siguiente forma:
1. Inspecciona uno a uno los caracteres almacenados en la variable *Tabla* (desde el primero *i = 1*, hasta el último *i = Len(Tabla)*). Si el carácter que se está inspeccionando es un *número* (1, 2, 3, 4, 5, 6, 7, 8, 9, 0) o el signo *punto* (.), se guarda en la variable *celda*.
2. Ciclo a ciclo los caracteres se concatenan (&) en la variable *celda* hasta que aparece alguno que no cumple con las condiciones anteriores (ser un número o un punto), luego entonces, tal carácter puede ser cualquiera de los dos siguientes: el que representa el avance (cambio) de celda, o el que representa el avance (cambio) de línea. En caso de que sea el primero (avance de celda), se asigna el contenido de la variable *celda* a la celda en uso, se limpia la variable y se avanza una columna. En caso de que sea el segundo (avance de línea), se asigna el contenido de la variable *celda* a la celda en uso, se limpia la variable, se avanza un renglón y se vuelve a la columna original.
3. Cuando se llega al último carácter de la variable *Tabla*, se asigna el contenido de la variable *celda* a la celda en uso y se finaliza el procedimiento.

```
End Sub
```

A excepción de los formularios en donde pegan la información que adquieren del *Portapapeles*, los procedimientos *PegarTabla2* y *PegarTabla3* son idénticos al anterior, por lo que no tiene caso que se expliquen.

El código completo de los procedimientos relacionados a la orden *Pegar* del menú *Editar Tabla* se encuentra en los **Anexos** de este documento.

5.4.3 Borrar

Cuando el usuario final haga clic en la orden *Borrar* del menú *Editar Tabla* se disparará el procedimiento *MnuBorrar_Click*. Este procedimiento borra el contenido de las celdas seleccionadas de la tabla *MSFlexGrid1* en uso (la tabla de *Form1*, *Form2* o *Form3*). A continuación se presenta *MnuBorrar_Click*:

```
Private Sub MnuBorrar_Click()

On Error GoTo ManipularError
```

- Esta sentencia intercepta cualquier tipo de error que se produzca en el procedimiento. Y la etiqueta *ManipularError* identifica la primera línea de la rutina que manipula el error.

```
If Module1.Bandera = 1 Then
Form1.MSFlexGrid1.FillStyle = flexFillRepeat
Form1.MSFlexGrid1.Text = ""
Form1.MSFlexGrid1.FillStyle = flexFillSingle
End If
```

- Si la variable *Bandera* almacena el número 1, es decir, el formulario *Form1* está cargado y activo, se borra el contenido de las celdas seleccionadas de la tabla *MSFlexGrid1* en uso (en este caso la tabla del formulario de análisis *Form1*). Este proceso se realiza a través de la propiedad .*Text*, estableciendo el contenido de la selección actual según el valor de la propiedad .*FillStyle*. Este valor puede ser *flexFillSingle*, lo que indica que el valor asignado a la propiedad .*Text* sólo afecta a la celda activa, o bien *flexFillRepeat*, indicando que afecta a todas las celdas seleccionadas.

```
If Module1.Bandera = 2 Then
Form2.MSFlexGrid1.FillStyle = flexFillRepeat
Form2.MSFlexGrid1.Text = ""
Form2.MSFlexGrid1.FillStyle = flexFillSingle
End If
```

- Si la variable *Bandera* almacena el número 2, es decir, el formulario *Form2* está cargado y activo, se borra el contenido de las celdas seleccionadas de la tabla *MSFlexGrid1* en uso (en este caso la tabla del formulario de análisis *Form2*).

```
If Module1.Bandera = 3 Then
Form3.MSFlexGrid1.FillStyle = flexFillRepeat
Form3.MSFlexGrid1.Text = ""
Form3.MSFlexGrid1.FillStyle = flexFillSingle
End If
```

- Si la variable *Bandera* almacena el número 3, es decir, el formulario *Form3* está cargado y activo, se borra el contenido de las celdas seleccionadas de la tabla *MSFlexGrid1* en uso (en este caso la tabla del formulario de análisis *Form3*).

```
Salir:
  Exit Sub
ManipularError:
  MsgBox Err.Description
  Resume Salir

End Sub
```

- Si en el procedimiento se origina algún error en tiempo de ejecución se manipula en esta rutina.

---

6 INTEGRACIÓN DE HERRAMIENTAS PARA EL ANÁLISIS DE INVENTARIOS

Este apartado tiene como meta satisfacer el objetivo secundario del presente documento, el cual es: *"Integrar software y herramientas para el análisis de modelos generales de inventarios al programa principal"*. Las órdenes de los menús *Mas Aplicaciones* y *Herramientas* precisamente satisfacen lo anterior.

Las órdenes del menú *Mas Aplicaciones* disparan programas especialmente diseñados para el análisis de inventarios (modelos generales), mientras que las órdenes del menú

Herramientas ejecutan software de análisis matemático. La estructura de ambos menús se muestra a continuación:

- Mas Aplicaciones
  - WinQSB (ITS)
  - The Management Scientist
  - Plantilla de Excel (IT)
  - Tora

- Herramientas
  - Calculadora
  - Derive

Visual Basic proporciona un medio para ejecutar programas desde una aplicación VB, la función *Shell*. Esta función es sencilla de utilizar, sólo hay que especificar el camino (ruta), nombre y tipo de ventana del programa que se desea ejecutar. Pero tiene una desventaja, sólo puede llamar aplicaciones de 16 bits (como *Tora* y *Derive*).

En caso de que se pretenda disparar aplicaciones de 32 bits (como *WinQSB(ITS)*, *The Management Scientist*, *Plantilla de Excel (IT)* y *Calculadora*) es necesario recurrir a la función de la API de Windows (función externa) *ShellExecute*, la cual permite ejecutar un fichero o bien la aplicación asociada con ese fichero.

Una función de la API de Windows no es otra cosa que una biblioteca dinámica (*Dynamic Link Library*, abreviadamente *DLL*), la cual permite que las aplicaciones Windows compartan código y recursos. Una *DLL* es en realidad un fichero ejecutable que contiene procedimientos que pueden ser utilizados por todas las aplicaciones. Las *DLL´s* son el corazón de las aplicaciones Windows. Hay miles de *DLL´s* que ejecutan una amplia variedad de tareas dispuestas para ser llamadas desde Visual Basic.

Ya que la *DLL ShellExecute* es una función externa a Visual Basic, es necesario proporcionar cierta información que permite localizarla y por lo tanto emplearla. Esta información se aporta incluyendo en la sección de declaraciones del formulario *MDIForm1* una declaración de *ShellExecute*, mediante la sentencia *Declare*. A continuación se presenta dicha declaración:

```
Public AIRR1 As String, AIRR2 As String, AIRR3 As String


Private Declare Function ShellExecute Lib "shell32.dll" Alias "ShellExecuteA" (ByVal
hwnd As Long, ByVal lpOperation As String, ByVal lpFile As String, ByVal lpParameters
As String, ByVal lpDirectory As String, ByVal nShowCmd As Long) As Long
```

- Con esta declaración ya se puede utilizar la función externa *ShellExecute* en Visual Basic. El argumento *hwnd* es el controlador de la ventana padre desde donde se lanza el proceso. El argumento *lpOperation* puede ser la cadena de caracteres “open”, “print” o “explore”. Si es “open”, la función abre el programa especificado por *lpFile*.

Los siguientes seis procedimientos permiten ejecutar los programas de los menús *Mas Aplicaciones* y *Herramientas*. Tales procedimientos se disparan cuando el usuario final hace clic en la orden correspondiente o en su botón asociado (en caso de que lo tenga).

```
Private Sub MnuWinQSB_Click()

On Error GoTo ManipularError

Dim X As Long, aplic As String, hWndAct As Long
aplic = "C:\ARCHIVOS DE PROGRAMA\AIRRAM\ITS.EXE"
X = ShellExecute(hwnd, "open", APLIC, vbNullString, vbNullString, SW_SHOW)

If X <= 32 Then
MsgBox "No se encontró la aplicación"
End If

Salir:
  Exit Sub
ManipularError:
  MsgBox Err.Description
  Resume Salir

End Sub
```

```
Private Sub MnuTheManagementScientist_Click()

On Error GoTo ManipularError

Dim X As Long, aplic As String, hWndAct As Long
aplic = "C:\ARCHIVOS DE PROGRAMA\AIRRAM\MS40.EXE"
X = ShellExecute(hwnd, "open", APLIC, vbNullString, vbNullString, SW_SHOW)
```

```
If X <= 32 Then
MsgBox "No se encontró la aplicación"
End If

Salir:
  Exit Sub
ManipularError:
  MsgBox Err.Description
  Resume Salir

End Sub
```

```
Private Sub MnuPlantilladeExcel_Click()

On Error GoTo ManipularError

Dim X As Long, aplic As String, hWndAct As Long
aplic = "C:\ARCHIVOS DE PROGRAMA\AIRRAM\INVENTORY THEORY.XLS"
X = ShellExecute(hwnd, "open", APLIC, vbNullString, vbNullString, SW_SHOW)

If X <= 32 Then
MsgBox "No se encontró la aplicación"
End If

Salir:
  Exit Sub
ManipularError:
  MsgBox Err.Description
  Resume Salir

End Sub
```

```
Private Sub MnuTora_Click()

On Error GoTo ManipularError

Shell "C:\ARCHIVOS DE PROGRAMA\AIRRAM\TORA.EXE", vbNormalFocus

Salir:
  Exit Sub
```

```
ManipularError:
  MsgBox Err.Description
  Resume Salir

End Sub
```

```
Private Sub MnuCalculadora_Click()

On Error GoTo ManipularError

Dim X As Long, aplic As String, hWndAct As Long
aplic = "C:\ARCHIVOS DE PROGRAMA\AIRRAM\CALC.EXE"
X = ShellExecute(hwnd, "open", APLIC, vbNullString, vbNullString, SW_SHOW)

If X <= 32 Then
MsgBox "No se encontró la aplicación"
End If

Salir:
  Exit Sub
ManipularError:
  MsgBox Err.Description
  Resume Salir

End Sub
```

```
Private Sub MnuDerive_Click()

On Error GoTo ManipularError

Shell "C:\ARCHIVOS DE PROGRAMA\AIRRAM\DERIVE.EXE", vbNormalFocus

Salir:
  Exit Sub
ManipularError:
  MsgBox Err.Description
  Resume Salir
```

End Sub

---

7 PROGRAMACIÓN DEL ENLACE A LA FACULTAD DE INGENIERÍA

En el apartado 1.8 del presente capítulo se diseñó gráficamente un enlace a la Facultad de Ingeniería, el cual es la animación (un escudo de la Facultad girando sobre si mismo) ubicada en el extremo superior derecho del formulario *MDIForm1*. El desarrollo del procedimiento que permite cargar la página electrónica de la Facultad se postergó a este apartado, porque en el 1.8 no se contaba con los conceptos y herramientas teóricas necesarias.

Cuando el usuario final haga doble clic sobre la animación (control *Picture1*) antes citada se disparará el procedimiento *Picture1_DblClick*. Este procedimiento carga el navegador *Microsoft Internet Explorer* después de fijar la página electrónica de la Facultad de Ingeniería como la página de inicio. A continuación se presenta *Picture1_DblClick*:

```
Private Sub Picture1_DblClick()

On Error GoTo ManipularError
```

- Esta sentencia intercepta cualquier tipo de error que se produzca en el procedimiento. Y la etiqueta *ManipularError* identifica la primera línea de la rutina que manipula el error.

```
Dim strString As String
 strString = "http://www.ingenieria.unam.mx/"
 SaveString HKEY_CURRENT_USER, "Software\Microsoft\Internet Explorer\Main",
"Start Page", strString
```

- Esta rutina dispara el procedimiento *SaveString*, el cual fija la página electrónica de la Facultad de Ingeniería como la página de inicio del navegador *Microsoft Internet Explorer*.

```
aplic = "C:\ARCHIVOS DE PROGRAMA\INTERNET EXPLORER\IEXPLORE.EXE"
X = ShellExecute(hwnd, "open", aplic, vbNullString, vbNullString, SW_SHOW)
```

- Esta rutina carga el navegador *Microsoft Internet Explorer*.

```
Salir:
  Exit Sub
ManipularError:
  MsgBox Err.Description
  Resume Salir

End Sub
```

- Si en el procedimiento se origina algún error en tiempo de ejecución se manipula en esta rutina.

El procedimiento *SaveString* utiliza tres nuevas *DLL´s* o funciones externas (*RegCloseKey*, *RegCreateKey* y *RegSetValueEx*), por lo que como en el caso de *ShellExecute* hay que declararlas en la sección de declaraciones del formulario *MDIForm1*. A continuación se presentan tales declaraciones:

```
Public AIRR1 As String, AIRR2 As String, AIRR3 As String

Private Declare Function ShellExecute Lib "shell32.dll" Alias "ShellExecuteA" (ByVal
hwnd As Long, ByVal lpOperation As String, ByVal lpFile As String, ByVal lpParameters
As String, ByVal lpDirectory As String, ByVal nShowCmd As Long) As Long

Private Declare Function RegCloseKey Lib "advapi32.dll" (ByVal hKey As Long) As
Long

Private Declare Function RegCreateKey Lib "advapi32.dll" Alias "RegCreateKeyA"
(ByVal hKey As Long, ByVal lpSubKey As String, phkResult As Long) As Long

Private Declare Function RegSetValueEx Lib "advapi32.dll" Alias "RegSetValueExA"
(ByVal hKey As Long, ByVal lpValueName As String, ByVal Reserved As Long, ByVal
dwType As Long, lpData As Any, ByVal cbData As Long) As Long
```

Antes de presentar el procedimiento *SaveString* es necesario introducir un concepto: *Registro de Windows*.

El *Registro de Windows* es la gran base de datos central del sistema operativo (Windows), guarda información del usuario, asociaciones de ficheros, configuración de las aplicaciones, información específica del ordenador, etc. La información en el *Registro de Windows* se ordena en *llaves* (rutas).

La llave que almacena la dirección (*URL*) de la página de inicio de *Internet Explorer* es: *"HKEY_CURRENT_USER\Software\Microsoft\Internet Explorer\Main\Start Page"*.

El procedimiento *SaveString* fija la página electrónica de la Facultad de Ingeniería como la página de inicio del navegador *Internet Explorer*, cambiando la dirección que

originalmente almacena la llave mencionada en el párrafo anterior, por la dirección de la página electrónica de la Facultad (http://www.ingenieria.unam.mx/). A continuación se presenta el procedimiento *SaveString*:

```
Private Sub SaveString(hKey As Long, strPath As String, strValue As String, strData As String)

    Dim Ret
    RegCreateKey hKey, strPath, Ret
    RegSetValueEx Ret, strValue, 0, REG_SZ, ByVal strData, Len(strData)
    RegCloseKey Ret

End Sub
```

8 SISTEMA DE AYUDA

Cuando una aplicación empieza a tener cierto grado de complejidad, el usuario espera tener asistencia a través de un fichero de ayuda (algo así como un pequeño manual). Mediante la orden *Temas de Ayuda* del menú *Ayuda* el usuario final cargará un pequeño manual sobre la aplicación (*AIRRAM*) que se ha venido desarrollando en este documento.

La orden *Acerca de AIRRAM* del menú *Ayuda* desplegará un formulario (una ventana) con información general de la aplicación.

8.1 Temas de Ayuda

Agregar un sistema de ayuda a una aplicación Visual Basic es bastante sencillo. Suponiendo que ya se tiene generado el fichero de ayuda, basta con que se asigne a la propiedad *HelpFile* del objeto *App* el nombre del fichero de ayuda (**.hlp*), para poder mostrar la ayuda cuando el usuario presione la tecla *F1* o solicite ayuda a través de la orden *Temas de Ayuda*.

Para asignar un fichero de ayuda a la propiedad *HelpFile*, se selecciona la orden *Propiedades del Proyecto* del menú *Proyecto* en la ventana de diseño de Visual Basic. La siguiente figura (Figura 6-26) muestra el diálogo *Propiedades del Proyecto*.

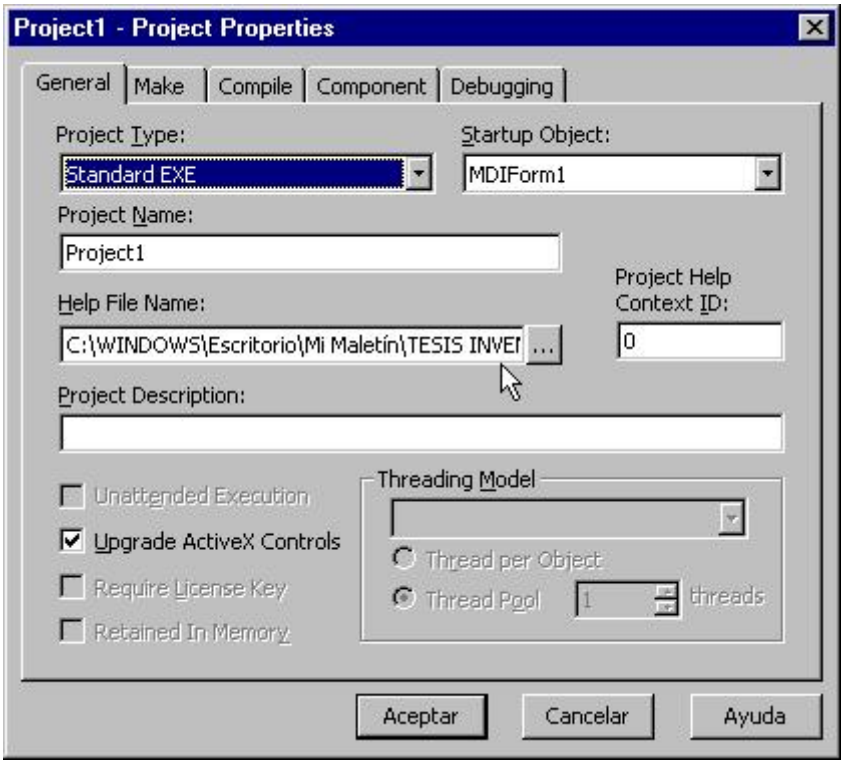


Figura 6-26
Diálogo *Propiedades del Proyecto*

Después, en la caja *Nombre del archivo de Ayuda* de la ficha *General* se escribe la ruta de acceso y el nombre del fichero de ayuda de la aplicación.

El fichero de ayuda de la aplicación (*AIRRAM*) no se va a desarrollar en este apartado, ya que contendrá casi la misma información que el capítulo que se dedicará a las conclusiones del presente documento.

Un fichero de ayuda puede ser escrito en cualquier procesador de textos (por ejemplo *Word*), a condición de que sea guardado con una extensión **.rtf* (*Rich Text Format*). Para que después, sea compilado con el programa *Help Workshop*, el cual convierte el fichero al formato **.hlp*. El motor de ayuda de Windows (*WinHelp*) sólo puede desplegar ficheros en formato **.hlp*, por lo que el fichero de ayuda de la aplicación debe tener ese formato.

Cuando el usuario final haga clic en la orden *Temas de Ayuda* (o en su botón asociado) del menú *Ayuda* se disparará el procedimiento *MnuTemasDeAyuda_Click*. Este procedimiento visualiza el motor de ayuda de Windows (ventana de ayuda) mostrando el fichero de ayuda especificado en la propiedad *HelpFile* del objeto *App*, con apoyo de la sentencia *SendKeys "{F1}"*. A continuación se presenta *MnuTemasDeAyuda_Click*:

```
Private Sub MnuTemasDeAyuda_Click()
SendKeys "{F1}"
```

End Sub

8.2 Acerca de AIRRAM

Para concluir con el desarrollo de *AIRRAM*, sólo hace falta incluir un formulario que despliegue información general de la aplicación. No es necesario programar dicho formulario, puesto que dentro de las herramientas de Visual Basic se incluye una con la que se puede agregar al proyecto un formulario prediseñado (plantilla), que cumple perfectamente con los requerimientos antes citados. El nombre de código del nuevo formulario será *frmAbout*.

Cuando se carga *frmAbout* despliega el título de la aplicación, la versión, una pequeña descripción e información de derechos de autor. En la siguiente figura (Figura 6-27) se presenta el formulario *frmAbout* en tiempo de ejecución.

Figura 6-27
Formulario *frmAbout* en tiempo de ejecución

Cuando el usuario final pulsa el botón *Aceptar* se descarga *frmAbout*. Y cuando pulsa el botón *Info. del sistema...* se ejecuta la aplicación *Información del sistema*, la cual muestra una visión general del hardware, de los componentes del sistema y del entorno de software del sistema que se está ejecutando. La siguiente figura (Figura 6-28) muestra una ventana de la aplicación *Información del sistema*.

Figura 6-28
Ventana de la aplicación *Información del sistema*

Cuando el usuario final haga clic en la orden *Acerca de AIRRAM* del menú *Ayuda* se disparará el procedimiento *MnuAcercaDeAIRRAM_Click*. Este procedimiento carga el formulario *frmAbout* de forma modal, es decir, el formulario tiene que ser cerrado (descargado) para continuar con la aplicación. A continuación se presenta *MnuAcercaDeAIRRAM_Click*:

```
Private Sub MnuAcercaDeAIRRAM_Click()
frmAbout.Show vbModal
End Sub
```

Aunque como ya se indicó el formulario *frmAbout* sólo es una plantilla que se agregó al proyecto, el código completo de los procedimientos relacionados a *frmAbout* se puede encontrar en los **Anexos** de este documento.

Ahora que ya se tiene el proyecto completo, se compila en Visual Basic y se guarda el archivo ejecutable resultante con el nombre de *AIRRAM.EXE*.

CAPÍTULO 7

Análisis de Caso

Ahora que la aplicación (*AIRRAM*) está completa hay que comprobar su eficacia, y la mejor manera de hacerlo es resolver un problema que se ajuste al modelo de inventarios de artículos múltiples con restricción de recursos (Presupuesto y Espacio). A lo largo de este capítulo se va a plantear y resolver un caso (problema) típico que se ajusta al modelo de inventarios antes citado. Tal caso se va a resolver de las siguientes dos formas:

- La primera utilizando el método de *Multiplicadores de Lagrange* apoyado en el programa de análisis matemático *Maple X*.
- Y la segunda simplemente con *AIRRAM*.

Obviamente ambos métodos tienen que arrojar los mismos resultados si *AIRRAM* realmente funciona.

1 PLANTEAMIENTO DEL PROBLEMA

La Bench Company es un pequeño fabricante de bancos de madera. Su línea incluye cuatro tipos de bancos de diferente tamaño, material, terminado y color. Los datos relevantes de producción son:

	Tipo de banco			
	1	2	3	4
Demanda anual (unidades)	1000	5000	10000	8000
Costo de preparación ($)	6	10	10	8
Costo unitario ($)	10	3	5	2
Espacio por unidad (ft^2)	5	1	1	1.5

Bench tiene un pequeño almacén para bancos terminados con un área de 1450 ft^2, también tiene un límite en el presupuesto de $ 3800 para inversión en inventario. Cada tipo de banco tiene un lugar fijo. Suponiendo que i = 20 % anual, calcule las cantidades óptimas que deben almacenarse.

Los datos relevantes de producción en términos de la nomenclatura que se ha venido utilizando en este documento son:

C = $ 3800
F = 1450 ft^2
i = 20 % anual

	Tipo de banco			
	1	2	3	4
D (unidades/año)	1000	5000	10000	8000
A ($)	6	10	10	8
c ($)	10	3	5	2
f (ft^2)	5	1	1	1.5

2 RESOLUCIÓN CON EL MÉTODO DE MULTIPLICADORES DE LAGRANGE

En este apartado se va a resolver el problema anterior utilizando el método de *Multiplicadores de Lagrange* auxiliándose del programa de análisis matemático *Maple X*. Por razones de rapidez, exactitud y confiabilidad todos los cálculos se efectuarán en *Maple X*, y estarán delimitados por un par de líneas como se hizo con el código del capítulo anterior.

Paso 1

Resolver el problema no restringido. Si ambas restricciones se satisfacen, esta solución es la óptima.

1. Se plantean las cantidades óptimas (Q^*_i, desde el primer hasta el cuarto artículo) de acuerdo al modelo fundamental en la teoría de análisis de inventarios (modelo EOQ):

```
>Q1:=sqrt((2*A1*D1)/(i*c1));Q2:=sqrt((2*A2*D2)/(i*c2));Q3:=sqrt((2*A3*D3)/(i*c3));
Q4:=sqrt((2*A4*D4)/(i*c4));
```

$$Q1 := \sqrt{2}\sqrt{\frac{A1\ D1}{i\ c1}}$$

$$Q2 := \sqrt{2}\sqrt{\frac{A2\ D2}{i\ c2}}$$

$$Q3 := \sqrt{2}\sqrt{\frac{A3\ D3}{i\ c3}}$$

$$Q4 := \sqrt{2}\sqrt{\frac{A4\ D4}{i\ c4}}$$

---

2. Se suministran los datos:

>C:=3800;F:=1450;i:=0.2;D1:=1000;A1:=6;c1:=10;f1:=5;D2:=5000;A2:=10;c2:=3;f2:=1;D3:=10000;A3:=10;c3:=5;f3:=1;D4:=8000;A4:=8;c4:=2;f4:=1.5;

$$C := 3800$$

$$F := 1450$$

$$i := .2$$

$$D1 := 1000$$

$$A1 := 6$$

$$c1 := 10$$

$$f1 := 5$$

$$D2 := 5000$$

$$A2 := 10$$

$$c2 := 3$$

$$f2 := 1$$

$$D3 := 10000$$

$$A3 := 10$$

$$c3 := 5$$

$$f3 := 1$$

$$D4 := 8000$$

$$A4 := 8$$

$$c4 := 2$$

$$f4 := 1.5$$

---

3. Se calculan las cantidades óptimas:

```
> Q1:=simplify(Q1);Q2:=simplify(Q2);Q3:=simplify(Q3);Q4:=simplify(Q4);
```

$$Q1 := 77.45966690$$

$$Q2 := 408.2482904$$

$$Q3 := 447.2135954$$

$$Q4 := 565.6854248$$

De acuerdo con lo anterior las cantidades óptimas son: Q^*_1 = 77.45966690, Q^*_2 = 408.2482904, Q^*_3 = 447.2135954 y Q^*_4 = 565.6854248. Pero, ¿ambas restricciones se satisfacen?.

4. Se comprueban las restricciones:

```
>Ks:=((c1*Q1)+(c2*Q2)+(c3*Q3)+(c4*Q4));Cs:=((f1*Q1)+(f2*Q2)+(f3*Q3)+(f4*Q4));
```

$$Ks := 5366.780367$$

$$Cs := 2091.288357$$

- *Ks* es la inversión total en el inventario, y debe ser menor o igual a la restricción de presupuesto (*C*).

- *Cs* es el espacio total en el inventario, y debe ser menor o igual a la restricción de espacio (*F*).

$$Ks = \$\ 5366.780367 > C = \$\ 3800$$
$$Cs = 2091.288357\ ft^2 > F = 1450\ ft^2$$

Se violan ambas restricciones, por lo tanto hay que continuar con el segundo paso del método de *Multiplicadores de Lagrange*.

Paso 2

Se incluye una de las restricciones (la de presupuesto), y se resuelve el problema de una restricción para encontrar las cantidades óptimas. Si la restricción de espacio se satisface, esta solución es la óptima.

1. Se establece el costo total anual promedio del inventario (*K(Q)*):

```
>K1:=(c1*D1)+((A1*D1)/Q1)+((i*c1*Q1)/2);K2:=(c2*D2)+((A2*D2)/Q2)+((i*c2*Q2)/2
);K3:=(c3*D3)+((A3*D3)/Q3)+((i*c3*Q3)/2);K4:=(c4*D4)+((A4*D4)/Q4)+((i*c4*Q4)/2
);
```

$$K1 := c1\ D1 + \frac{A1\ D1}{Q1} + \frac{1}{2} i\ c1\ Q1$$

$$K2 := c2\ D2 + \frac{A2\ D2}{Q2} + \frac{1}{2} i\ c2\ Q2$$

$$K3 := c3\ D3 + \frac{A3\ D3}{Q3} + \frac{1}{2} i\ c3\ Q3$$

$$K4 := c4\ D4 + \frac{A4\ D4}{Q4} + \frac{1}{2} i\ c4\ Q4$$

```
> K(Q):=K1+K2+K3+K4;
```

$$K(Q) := c1\ D1 + \frac{A1\ D1}{Q1} + \frac{1}{2} i\, c1\ Q1 + c2\ D2 + \frac{A2\ D2}{Q2} + \frac{1}{2} i\, c2\ Q2 + c3\ D3 + \frac{A3\ D3}{Q3} + \frac{1}{2} i\, c3\ Q3 + c4\ D4 + \frac{A4\ D4}{Q4} + \frac{1}{2} i\, c4\ Q4$$

---

2. Se establece la ecuación de *Lagrange* ($K(Q, \lambda)$):

```
> Ks:=((c1*Q1)+(c2*Q2)+(c3*Q3)+(c4*Q4));
```

$$Ks := c1\ Q1 + c2\ Q2 + c3\ Q3 + c4\ Q4$$

```
> K(Q,lambda):=K(Q)+(lambda*(Ks-C));
```

$$K(Q, \lambda) := c1\ D1 + \frac{A1\ D1}{Q1} + \frac{1}{2} i\, c1\ Q1 + c2\ D2 + \frac{A2\ D2}{Q2} + \frac{1}{2} i\, c2\ Q2 + c3\ D3 + \frac{A3\ D3}{Q3} + \frac{1}{2} i\, c3\ Q3 + c4\ D4 + \frac{A4\ D4}{Q4} + \frac{1}{2} i\, c4\ Q4 + \lambda\,(c1\ Q1 + c2\ Q2 + c3\ Q3 + c4\ Q4 - C)$$

---

3. Se forma un sistema de ecuaciones no lineales con cinco incógnitas (las cuales son las cantidades óptimas y λ) derivando parcialmente la ecuación de *Lagrange* con respecto a dichas incógnitas:

```
> Ec1:=diff(K(Q,lambda), Q1);Ec2:=diff(K(Q,lambda), Q2);Ec3:=diff(K(Q,lambda),
Q3);Ec4:=diff(K(Q,lambda), Q4);Ec5:=diff(K(Q,lambda), lambda);
```

$$Ec1 := -\frac{A1\ D1}{Q1^2} + \frac{1}{2} i\, c1 + \lambda\, c1$$

$$Ec2 := -\frac{A2\ D2}{Q2^2} + \frac{1}{2} i\, c2 + \lambda\, c2$$

$$Ec3 := -\frac{A3\ D3}{Q3^2} + \frac{1}{2} i\ c3 + \lambda\ c3$$

$$Ec4 := -\frac{A4\ D4}{Q4^2} + \frac{1}{2} i\ c4 + \lambda\ c4$$

$$Ec5 := c1\ Q1 + c2\ Q2 + c3\ Q3 + c4\ Q4 - C$$

A propósito, λ es el *Multiplicador de Lagrange*.

4. Se resuelve el sistema de ecuaciones:

```
> Sols:=solve({Ec1, Ec2, Ec3, Ec4, Ec5}, {Q1,Q2,Q3,Q4,lambda});
```

$$Sols := \{Q1 = -215.1504408, \lambda = -.08703815661, Q4 = 1571.236662, Q3 = 1242.171649, Q2 = -1133.942388\},$$
$$\{\lambda = -.08325800554, Q2 = 997.7477979, Q4 = -1382.519903, Q3 = 1092.977951, Q1 = -189.3093343\},$$
$$\{Q4 = 400.5389577, Q1 = 54.84605557, Q2 = 289.0640938, Q3 = 316.6538495, \lambda = .09946212963\},$$
$$\{\lambda = -.09703300349, Q3 = 2596.309606, Q4 = -3284.100747, Q2 = -2370.095563, Q1 = 449.6940150\},$$
$$\{Q2 = 406.3675541, Q3 = 445.1533520, Q1 = -77.10282229, \lambda = .0009277751259, Q4 = 563.0794002\},$$
$$\{\lambda = -.03327523508, Q4 = -692.5186328, Q1 = 94.82701917, Q2 = 499.7822739, Q3 = 547.4840505\},$$
$$\{Q2 = -531.7754393, \lambda = -.04106243357, Q1 = 100.8972955, Q4 = 736.8496632, Q3 = 582.5308072\},$$
$$\{Q4 = 2402.747487, Q1 = 329.0097496, \lambda = -.09445714248, Q2 = 1734.033635, Q3 = -1899.538675\}$$

---

5. Sólo en el tercer juego de respuestas todas las cantidades óptimas son positivas, por lo tanto, ese es el juego de respuestas correctas:

```
> Sols[3];
```

$$\{Q4 = 400.5389577, Q1 = 54.84605557, Q2 = 289.0640938, Q3 = 316.6538495, \lambda = .09946212963\}$$

De acuerdo con lo anterior las cantidades óptimas son: $Q^*_1 = 54.84605557$, $Q^*_2 = 289.0640938$, $Q^*_3 = 316.6538495$ y $Q^*_4 = 400.5389577$. Pero, ¿se satisface la restricción de espacio?.

6. Se comprueba la restricción de espacio:

```
> Cs:=((f1*(Sols[3,2]))+(f2*(Sols[3,3]))+(f3*(Sols[3,4]))+(f4*(Sols[3,1])));
```

$$Cs := 5\ Q1 + Q2 + Q3 + 1.5\ Q4 = 1480.756658$$

$$Cs = 1480.756658\ \text{ft}^2 > F = 1450\ \text{ft}^2$$

Se viola la restricción de espacio, por lo tanto hay que continuar con el tercer paso del método de *Multiplicadores de Lagrange*.

Paso 3

Se repite el proceso sólo con la restricción de espacio. Si la restricción de presupuesto se satisface, esta solución es la óptima.

1. Se establece una nueva ecuación de *Lagrange* ($K(Q, \lambda)$):

```
> Cs:=((f1*Q1)+(f2*Q2)+(f3*Q3)+(f4*Q4));
```

$$Cs := f1\ Q1 + f2\ Q2 + f3\ Q3 + f4\ Q4$$

```
> K(Q,lambda):=K(Q)+(lambda*(Cs-F));
```

$$K(Q, \lambda) := c1\ D1 + \frac{A1\ D1}{Q1} + \frac{1}{2} i\ c1\ Q1 + c2\ D2 + \frac{A2\ D2}{Q2} + \frac{1}{2} i\ c2\ Q2 + c3\ D3 + \frac{A3\ D3}{Q3} + \frac{1}{2} i\ c3\ Q3 + c4\ D4 + \frac{A4\ D4}{Q4}$$
$$+ \frac{1}{2} i\ c4\ Q4 + \lambda\ (f1\ Q1 + f2\ Q2 + f3\ Q3 + f4\ Q4 - F)$$

---

2. Se forma un sistema de ecuaciones no lineales con cinco incógnitas (las cuales son las cantidades óptimas y λ) derivando parcialmente la nueva ecuación de *Lagrange* con respecto a dichas incógnitas:

```
> Ec1:=diff(K(Q,lambda), Q1);Ec2:=diff(K(Q,lambda), Q2);Ec3:=diff(K(Q,lambda),
Q3);Ec4:=diff(K(Q,lambda), Q4);Ec5:=diff(K(Q,lambda), lambda);
```

$$Ec1 := -\frac{A1\ D1}{Q1^2} + \frac{1}{2} i\ c1 + \lambda\ f1$$

$$Ec2 := -\frac{A2\ D2}{Q2^2} + \frac{1}{2} i\ c2 + \lambda\ f2$$

$$Ec3 := -\frac{A3\ D3}{Q3^2} + \frac{1}{2} i\ c3 + \lambda\ f3$$

$$Ec4 := -\frac{A4\ D4}{Q4^2} + \frac{1}{2} i\ c4 + \lambda\ f4$$

$$Ec5 := f1\ Q1 + f2\ Q2 + f3\ Q3 + f4\ Q4 - F$$

---

3. Se resuelve el sistema de ecuaciones:

```
> Sols:=solve({Ec1, Ec2, Ec3, Ec4, Ec5}, {Q1,Q2,Q3,Q4,lambda});
```

$$Sols := \{Q3 = -515.3649527, Q4 = 2082.540009, Q1 = -125.2411275, \lambda = -.1234954423, Q2 = -532.2394235\},$$
$$\{\lambda = -.1101408683, Q3 = 506.4611589, Q2 = -513.1794500, Q4 = 1356.347284, Q1 = -115.5605271\},$$
$$\{\lambda = -.1098123087, Q3 = -506.2478798, Q1 = -115.3498378, Q4 = 1346.840722, Q2 = 512.7359857\},$$
$$\{Q2 = 432.7730617, Q1 = -84.77771563, Q4 = 652.2346071, \lambda = -.03303792797, Q3 = 462.7636057\}, \{$$
$$Q2 = 826.9487399 + 1214.112152\ I, Q1 = -10.20117409 - 103.6589261\ I, Q4 = -30.09392111 - 490.7828242\ I,$$
$$\lambda = -.3084850147 - .02156104073\ I, Q3 = 719.1980123 + 40.35671521\ I\}, \{Q2 = 826.9487399 - 1214.112152\ I,$$
$$Q1 = -10.20117409 + 103.6589261\ I, Q4 = -30.09392111 + 490.7828242\ I, \lambda = -.3084850147 + .02156104073\ I,$$

$Q3 = 719.1980123 - 40.35671521\,I\}, \{Q3 = -700.7380384 - 11.66910443\,I, Q2 = 2186.170573 + 1334.664918\,I,$
$Q1 = -3.903697255 - 111.2978114\,I, Q4 = -10.60936547 - 511.0045047\,I, \lambda = -.2965173760 - .006778905955\,I\}$
$, \{Q2 = 2186.170573 - 1334.664918\,I, Q1 = -3.903697255 + 111.2978114\,I, Q4 = -10.60936547 + 511.0045047\,I,$
$\lambda = -.2965173760 + .006778905955\,I, Q3 = -700.7380384 + 11.66910443\,I\}, \{$
$Q2 = -31.00260669 - 501.0531801\,I, Q1 = -2.607743569 - 63.41523835\,I, Q4 = -11.49113870 - 341.9988756\,I,$
$\lambda = -.4968871744 - .02445838054\,I, Q3 = 1511.278033 + 1331.127685\,I\}, \{Q4 = -11.49113870 + 341.9988756\,I,$
$\lambda = -.4968871744 + .02445838054\,I, Q3 = 1511.278033 - 1331.127685\,I, Q2 = -31.00260669 + 501.0531801\,I,$
$Q1 = -2.607743569 + 63.41523835\,I\}, \{Q4 = -10.81314404 - 361.6030272\,I, \lambda = -.4587649329 - .01948041351\,I,$
$Q3 = 1444.882468 + 324.1234658\,I, Q2 = 34.10858813 + 558.0531952\,I, Q1 = -2.554267993 - 67.95442405\,I\}, \{$
$Q2 = 34.10858813 - 558.0531952\,I, Q1 = -2.554267993 + 67.95442405\,I, Q4 = -10.81314404 + 361.6030272\,I,$
$\lambda = -.4587649329 + .01948041351\,I, Q3 = 1444.882468 - 324.1234658\,I\}, \{Q4 = 10.66058853 + 360.1645760\,I,$
$\lambda = -.4613873929 - .01943731423\,I, Q2 = -33.21822610 - 553.6128999\,I, Q3 = 1479.780210 + 351.4467703\,I,$
$Q1 = -2.510573272 - 67.61614689\,I\}, \{Q3 = 1479.780210 - 351.4467703\,I, Q2 = -33.21822610 + 553.6128999\,I,$
$Q1 = -2.510573272 + 67.61614689\,I, Q4 = 10.66058853 - 360.1645760\,I, \lambda = -.4613873929 + .01943731423\,I\}, \{$
$Q2 = 845.5509147 + 2.219113979\,I, Q1 = -1.219374403 - 199.7655242\,I, Q4 = 1.260078503 + 664.1297732\,I,$
$\lambda = -.2300671056 - .0003670746012\,I, Q3 = 608.6558396 + .4138472950\,I\}, \{Q4 = 1.260078503 - 664.1297732\,I,$
$Q3 = 608.6558396 - .4138472950\,I, \lambda = -.2300671056 + .0003670746012\,I, Q2 = 845.5509147 - 2.219113979\,I,$
$Q1 = -1.219374403 + 199.7655242\,I\}, \{Q4 = -3.509838396 + 522.8902500\,I,$
$\lambda = -.2893636266 + .002094763824\,I, Q3 = -688.9968071 + 3.425927533\,I, Q2 = 2137.473040 - 208.4783160\,I,$
$Q1 = 1.357704969 - 115.8565973\,I\}, \{Q4 = -3.509838396 - 522.8902500\,I, \lambda = -.2893636266 - .002094763824\,I,$
$Q3 = -688.9968071 - 3.425927533\,I, Q2 = 2137.473040 + 208.4783160\,I, Q1 = 1.357704969 + 115.8565973\,I\}, \{$
$\lambda = -.4775823510 + .02837053825\,I, Q2 = -41.72320217 + 525.6356944\,I, Q3 = 1496.700330 - 724.9056410\,I,$
$Q4 = -14.44558879 + 351.1607155\,I, Q1 = 3.338251041 - 65.49422532\,I\}, \{Q2 = -41.72320217 - 525.6356944\,I,$
$Q1 = 3.338251041 + 65.49422532\,I, Q4 = -14.44558879 - 351.1607155\,I, \lambda = -.4775823510 - .02837053825\,I,$
$Q3 = 1496.700330 + 724.9056410\,I\},$
$\{Q1 = 52.61323830, Q4 = 341.0426808, \lambda = .2335020743, Q3 = 369.2319950, Q2 = 306.1377923\}, \{$
$Q2 = 721.7858692 - 235.7935751\,I, Q4 = -150.5835484 + 606.1038381\,I, \lambda = -.2300053381 + .05119541417\,I,$
$Q1 = 70.69987826 - 123.3848280\,I, Q3 = 600.5900621 - 56.43804197\,I\}, \{Q3 = 600.5900621 + 56.43804197\,I,$
$Q2 = 721.7858692 + 235.7935751\,I, Q1 = 70.69987826 + 123.3848280\,I, Q4 = -150.5835484 - 606.1038381\,I,$
$\lambda = -.2300053381 - .05119541417\,I\},$

$$\{\lambda = -.03602125190, Q3 = 464.2489772, Q2 = -435.2116574, Q1 = 85.54543515, Q4 = 662.1570030\},$$
$$\{Q3 = -468.3730095, Q2 = 442.0764209, Q1 = 87.74967700, Q4 = 691.6988024, \lambda = -.04415596181\},$$
$$\{Q2 = -508.4920956, Q1 = 113.3636983, \lambda = -.1066244368, Q3 = -504.1924187, Q4 = 1263.910682\}, \{$$
$$Q3 = -575.7182953 + 8.348347542\, I, \lambda = -.1984870371 + .008746167761\, I, Q1 = 281.2824548 - 236.8021204\, I,$$
$$Q4 = -53.71228333 + 803.8376318\, I, Q2 = 699.8744462 - 30.09419346\, I\}, \{Q3 = -575.7182953 - 8.348347542\, I,$$
$$Q1 = 281.2824548 + 236.8021204\, I, Q4 = -53.71228333 - 803.8376318\, I, \lambda = -.1984870371 - .008746167761\, I,$$
$$Q2 = 699.8744462 + 30.09419346\, I\}, \{Q2 = -698.9803813 + 23.50222048\, I, Q1 = 327.5746152 - 246.0394269\, I,$$
$$\lambda = -.1980077106 + .006866453972\, I, Q3 = 575.3311662 - 6.539859894\, I, Q4 = -42.81590744 + 808.8231827\, I\},$$
$$\{Q4 = -42.81590744 - 808.8231827\, I, Q2 = -698.9803813 - 23.50222048\, I, \lambda = -.1980077106 - .006866453972\, I,$$
$$Q1 = 327.5746152 + 246.0394269\, I, Q3 = 575.3311662 + 6.539859894\, I\}, \{\lambda = -.1977960323 + .002468879137\, I,$$
$$Q4 = -15.56516005 + 813.1139545\, I, Q1 = 549.5725059 - 246.0930958\, I, Q3 = -575.2267179 + 2.349640804\, I,$$
$$Q2 = -699.2880715 + 8.444906702\, I\}, \{Q1 = 549.5725059 + 246.0930958\, I, Q2 = -699.2880715 - 8.444906702\, I,$$
$$Q3 = -575.2267179 - 2.349640804\, I, \lambda = -.1977960323 - .002468879137\, I, Q4 = -15.56516005 - 813.1139545\, I\}$$

---

4. Sólo en el vigésimo primer juego de respuestas todas las cantidades óptimas son reales y positivas, por lo tanto, ese es el juego de respuestas correctas:

```
> Sols[21];
```

$$\{Q1 = 52.61323830, Q4 = 341.0426808, \lambda = .2335020743, Q3 = 369.2319950, Q2 = 306.1377923\}$$

De acuerdo con lo anterior las cantidades óptimas son: $Q^*_1 = 52.61323830$, $Q^*_2 = 306.1377923$, $Q^*_3 = 369.2319950$ y $Q^*_4 = 341.0426808$. Pero, ¿se satisface la restricción de presupuesto?.

5. Se comprueba la restricción de presupuesto:

```
> Ks:=((c1*(Sols[21,1]))+(c2*(Sols[21,5]))+(c3*(Sols[21,4]))+(c4*(Sols[21,2])));
```

$$Ks := 10\,Q1 + 3\,Q2 + 5\,Q3 + 2\,Q4 = 3972.791097$$

$$Ks = \$\ 3972.791097 > C = \$\ 3800$$

Se viola la restricción de presupuesto, por lo tanto hay que continuar con el cuarto paso del método de *Multiplicadores de Lagrange*.

Paso 4

Ambas restricciones son activas, por lo tanto debe resolverse el problema involucrando las dos restricciones.

1. Se establece una nueva ecuación de *Lagrange* ($K(Q, \lambda_1, \lambda_2)$):

```
>Ks:=((c1*Q1)+(c2*Q2)+(c3*Q3)+(c4*Q4));Cs:=((f1*Q1)+(f2*Q2)+(f3*Q3)+(f4*Q4));
```

$$Ks := c1\,Q1 + c2\,Q2 + c3\,Q3 + c4\,Q4$$

$$Cs := f1\,Q1 + f2\,Q2 + f3\,Q3 + f4\,Q4$$

```
> K(Q,lambda1,lambda2):=K(Q)+(lambda1*(Ks-C))+(lambda2*(Cs-F));
```

$$K(Q, \lambda1, \lambda2) := c1\,D1 + \frac{A1\,D1}{Q1} + \frac{1}{2} i\,c1\,Q1 + c2\,D2 + \frac{A2\,D2}{Q2} + \frac{1}{2} i\,c2\,Q2 + c3\,D3 + \frac{A3\,D3}{Q3} + \frac{1}{2} i\,c3\,Q3 + c4\,D4$$
$$+ \frac{A4\,D4}{Q4} + \frac{1}{2} i\,c4\,Q4 + \lambda1\,(c1\,Q1 + c2\,Q2 + c3\,Q3 + c4\,Q4 - C) + \lambda2\,(f1\,Q1 + f2\,Q2 + f3\,Q3 + f4\,Q4 - F)$$

A propósito, en este paso se manejan dos *Multiplicadores de Lagrange* (λ_1 y λ_2).

2. Se forma un sistema de ecuaciones no lineales con seis incógnitas (las cuales son las cantidades óptimas, λ_1 y λ_2) derivando parcialmente la nueva ecuación de *Lagrange* con respecto a dichas incógnitas:

> Ec1:=diff(K(Q,lambda1,lambda2), Q1);Ec2:=diff(K(Q,lambda1,lambda2), Q2);Ec3:=diff(K(Q,lambda1,lambda2), Q3);Ec4:=diff(K(Q,lambda1,lambda2), Q4);Ec5:=diff(K(Q,lambda1,lambda2),lambda1);Ec6:=diff(K(Q,lambda1,lambda2), lambda2);

$$Ec1 := -\frac{A1\ \mathrm{D1}}{Q1^2} + \frac{1}{2} i\, c1 + \lambda 1\ c1 + \lambda 2\ f1$$

$$Ec2 := -\frac{A2\ \mathrm{D2}}{Q2^2} + \frac{1}{2} i\, c2 + \lambda 1\ c2 + \lambda 2\ f2$$

$$Ec3 := -\frac{A3\ \mathrm{D3}}{Q3^2} + \frac{1}{2} i\, c3 + \lambda 1\ c3 + \lambda 2\ f3$$

$$Ec4 := -\frac{A4\ \mathrm{D4}}{Q4^2} + \frac{1}{2} i\, c4 + \lambda 1\ c4 + \lambda 2\ f4$$

$$Ec5 := c1\ Q1 + c2\ Q2 + c3\ Q3 + c4\ Q4 - C$$

$$Ec6 := f1\ Q1 + f2\ Q2 + f3\ Q3 + f4\ Q4 - F$$

3. Se resuelve el sistema de ecuaciones:

> Sols:=solve({Ec1, Ec2, Ec3, Ec4, Ec5,Ec6}, {Q1,Q2,Q3,Q4,lambda1,lambda2});

$$Sols := \{\lambda 2 = -.002572879807,\ Q4 = 1630.638850,\ \lambda 1 = -.08603566400,\ Q1 = -217.5464652,\ Q3 = 1219.432400,\ Q2 = -1127.658349\},\ \{Q3 = 404.6238069,\ Q2 = 392.6836135,\ Q1 = -81.42799440,\ \lambda 2 = -.1055626798,\ \lambda 1 = .04327200375,\ Q4 = 706.5550344\},\ \{Q3 = -217.7725324 + 110.8317246\, I,\ Q2 = 1634.637279 - 175.9100771\, I,\ \lambda 2 = -1.433359849 - 2.021186477\, I,\ Q1 = -17.76885364 - 33.95985854\, I,\ \lambda 1 = .3838114626 + .6750407358\, I,\ Q4 = 81.31968135 + 156.5850968\, I\},\ \{Q1 = -17.76885364 + 33.95985854\, I,$$

$$Q3 = -217.7725324 - 110.8317246\,I,\ Q2 = 1634.637279 + 175.9100771\,I,\ \lambda 2 = -1.433359849 + 2.021186477\,I,$$
$$\lambda 1 = .3838114626 - .6750407358\,I,\ Q4 = 81.31968135 - 156.5850968\,I\},\ \{Q4 = 95.33822626 - 73.82139919\,I,$$
$$Q3 = -104.0655692 - 132.1041730\,I,\ Q2 = 1307.534934 + 322.4911197\,I,\ Q1 = 20.70465921 - 15.93096959\,I,$$
$$\lambda 2 = 1.302891659 + 5.124398778\,I,\ \lambda 1 = -.5261615599 - 1.712406033\,I\},\ \{Q2 = 1307.534934 - 322.4911197\,I,$$
$$Q1 = 20.70465921 + 15.93096959\,I,\ \lambda 2 = 1.302891659 - 5.124398778\,I,\ \lambda 1 = -.5261615599 + 1.712406033\,I,$$
$$Q4 = 95.33822626 + 73.82139919\,I,\ Q3 = -104.0655692 + 132.1041730\,I\},\ \{Q4 = 375.0308919,$$
$$Q3 = 327.8399726,\ \lambda 1 = .07101540252,\ Q2 = 291.5109741,\ Q1 = 53.62054309,\ \lambda 2 = .07533688844\},\ \{$$
$$Q2 = -541.4515626,\ Q1 = 90.87886004,\ \lambda 2 = .09479059313,\ \lambda 1 = -.07474698772,\ Q4 = 576.3127681,$$
$$Q3 = 672.5881103\},\ \{Q3 = 191.0617370 + 62.72575910\,I,\ Q2 = 234.4152448 + 75.04644146\,I,$$
$$Q1 = 232.6244669 - 106.5215557\,I,\ \lambda 2 = -1.307813520 + 1.000267169\,I,\ \lambda 1 = .5598949836 - .4931943899\,I,$$
$$Q4 = -92.39954432 + 263.2237187\,I\},\ \{Q3 = 191.0617370 - 62.72575910\,I,\ Q2 = 234.4152448 - 75.04644146\,I,$$
$$Q1 = 232.6244669 + 106.5215557\,I,\ \lambda 2 = -1.307813520 - 1.000267169\,I,\ \lambda 1 = .5598949836 + .4931943899\,I,$$
$$Q4 = -92.39954432 - 263.2237187\,I\},\ \{Q4 = -339.9812836 + 427.2882409\,I,$$
$$Q2 = -393.2566991 - 289.4495778\,I,\ Q1 = 407.0940305 - 118.0790780\,I,\ \lambda 2 = -.1077343959 + .4111734579\,I,$$
$$\lambda 1 = -.04331161734 - .2037998036\,I,\ Q3 = 317.7584718 + 238.9126062\,I\},\ \{$$
$$Q4 = -339.9812836 - 427.2882409\,I,\ Q3 = 317.7584718 - 238.9126062\,I,\ Q2 = -393.2566991 + 289.4495778\,I,$$
$$Q1 = 407.0940305 + 118.0790780\,I,\ \lambda 2 = -.1077343959 - .4111734579\,I,\ \lambda 1 = -.04331161734 + .2037998036\,I\}$$

---

4. Sólo en el séptimo juego de respuestas todas las cantidades óptimas son reales y positivas, por lo tanto, ese es el juego de respuestas correctas:

```
> Sols[7];
```

$$\{Q4 = 375.0308919,\ Q3 = 327.8399726,\ \lambda 1 = .07101540252,\ Q2 = 291.5109741,\ Q1 = 53.62054309,$$
$$\lambda 2 = .07533688844\}$$

De acuerdo con lo anterior las cantidades óptimas son: $Q^*_1 = 53.62054309$, $Q^*_2 = 291.5109741$, $Q^*_3 = 327.8399726$ y $Q^*_4 = 375.0308919$. Mientras que los *Multiplicadores de Lagrange* valen: $\lambda_1 = 0.07101540252$ y $\lambda_2 = 0.07533688844$.

5. Se comprueban ambas restricciones:

```
>Ks:=((c1*(Sols[7,5]))+(c2*(Sols[7,4]))+(c3*(Sols[7,2]))+(c4*(Sols[7,1])));Cs:=((f1*(
Sols[7,5]))+(f2*(Sols[7,4]))+(f3*(Sols[7,2]))+(f4*(Sols[7,1])));
```

$$Ks := 10\ Q1 + 3\ Q2 + 5\ Q3 + 2\ Q4 = 3800.000000$$

$$Cs := 5\ Q1 + Q2 + Q3 + 1.5\ Q4 = 1450.000000$$

$$Ks = C = \$\ 3800$$
$$Cs = F = 1450\ ft^2$$

Ambas restricciones se satisfacen, por lo tanto se ha llegado a la solución óptima.

6. También se obtiene el costo total anual promedio del inventario ($K(Q)$) relacionado a la solución óptima:

```
> K(Q):=K(Q);
```

$$K(Q) := 92139.09701$$

La solución es:

λ_1 = 0.07101540252
λ_2 = 0.07533688844
Q^*_1 = 53.62054309
Q^*_2 = 291.5109741
Q^*_3 = 327.8399726
Q^*_4 = 375.0308919
$K(Q)$ = 92139.09701

3 RESOLUCIÓN CON AIRRAM

En este apartado se va a resolver el mismo caso que se ha venido estudiando a lo largo de este capítulo, pero de una forma diferente. En esta ocasión se va a resolver con ayuda *AIRRAM* a través del siguiente procedimiento:

1. Se carga la aplicación (*AIRRAM*) y se llama a la interfaz en donde se puede dar solución al caso en estudio. Tal interfaz es la que lleva el nombre de *Análisis con Restricción de Presupuesto y Espacio* y se dispara a través de la orden *Presupuesto y Espacio* (o su botón asociado) del menú *Análisis con Restricción de...*
2. Se suministran los datos como se muestra en la siguiente figura (Figura 7-1):

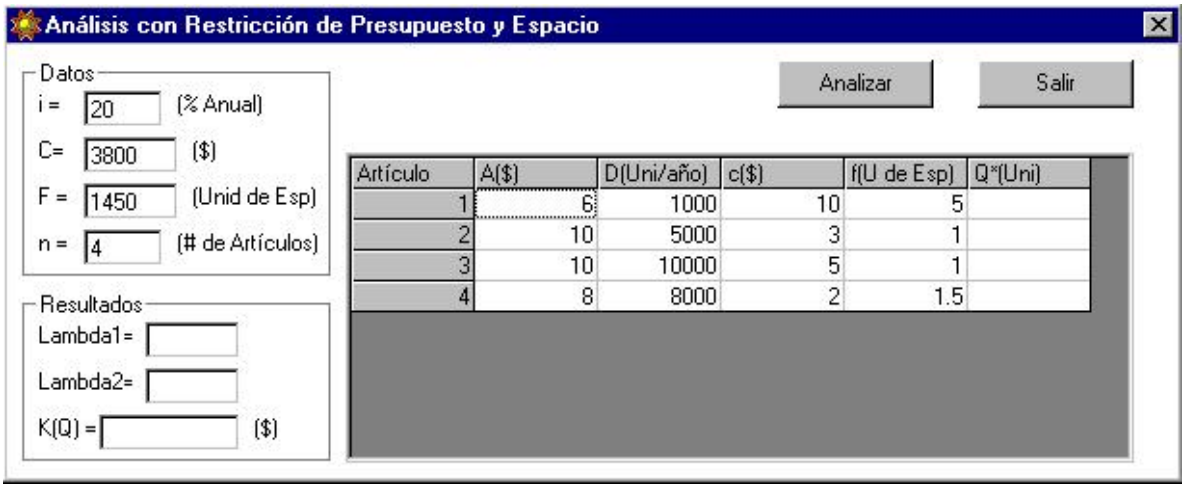


Figura 7-1
Interfaz de *Análisis con Restricción de Presupuesto y Espacio* (datos del caso)

3. Se dispara el análisis pulsando el botón etiquetado con la leyenda *Analizar*.
4. La aplicación interrumpe el análisis para informar mediante una caja de diálogo que ambas restricciones son activas. El análisis se reanuda una vez que se cierra la caja de diálogo.
5. Cuando el proceso termina se despliegan los resultados como se muestra en la siguiente figura (Figura 7-2):

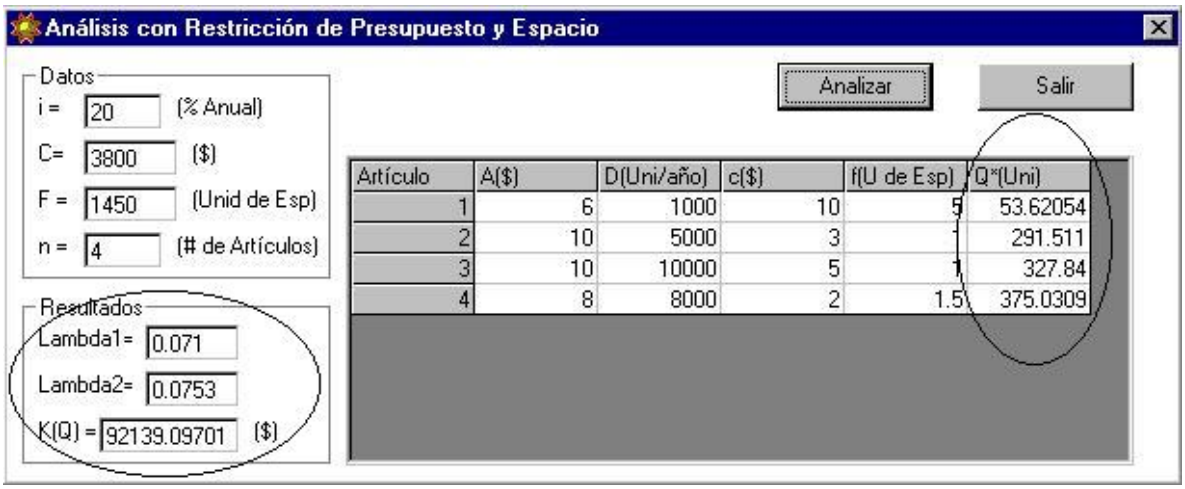


Figura 7-2
Interfaz de *Análisis con Restricción de Presupuesto y Espacio* (resultados)

De acuerdo con lo anterior la solución que se obtiene utilizando *AIRRAM* es:

$\lambda_1 = 0.071$
$\lambda_2 = 0.0753$
$Q^*_1 = 53.62054$
$Q^*_2 = 291.511$
$Q^*_3 = 327.84$
$Q^*_4 = 375.0309$
$K(Q) = 92139.09701$

Comparando esta solución con la que se alcanzó en el apartado anterior:

$\lambda_1 = 0.07101540252$
$\lambda_2 = 0.07533688844$
$Q^*_1 = 53.62054309$
$Q^*_2 = 291.5109741$
$Q^*_3 = 327.8399726$
$Q^*_4 = 375.0308919$
$K(Q) = 92139.09701$

Se puede decir que ambas soluciones son casi idénticas, la única diferencia radica en que las cifras que se obtienen mediante *AIRRAM* están redondeadas a efectos de que puedan desplegarse correctamente en las celdas correspondientes.

4 REDONDEO DE LAS CANTIDADES ÓPTIMAS

Las cantidades óptimas (Q^*_i) representan bancos de algún tipo que se tienen que almacenar, y el lector estará de acuerdo que no se pueden almacenar cantidades fraccionarias de bancos; por lo que es necesario encontrar las cifras "enteras" a las que hay que redondear las cantidades óptimas.

Cada Q^*_i puede redondearse hacia arriba o hacia abajo, es decir, al entero próximo anterior o posterior. Lo que provoca que existan dos posibles respuestas para cada Q^*_i, y como se cuenta con cuatro Q^*_i hay $2 \times 2 \times 2 \times 2 = 16$ posibles juegos de respuestas.

Esos juegos de respuestas son:

$$Q^*_1, Q^*_2, Q^*_3, Q^*_4$$

(53, 291, 327, 375)
(53, 291, 327, 376)
(53, 291, 328, 375)
(53, 291, 328, 376)
(53, 292, 327, 375)
(53, 292, 327, 376)
(53, 292, 328, 375)
(53, 292, 328, 376)
(54, 291, 327, 375)
(54, 291, 327, 376)
(54, 291, 328, 375)
(54, 291, 328, 376)
(54, 292, 327, 375)
(54, 292, 327, 376)
(54, 292, 328, 375)
(54, 292, 328, 376)

Para encontrar el juego de respuestas correctas es necesario evaluar con cual de ellas se alcanza el costo total anual promedio del inventario (*K(Q)*) "mínimo", mientras se satisfacen ambas restricciones ($Ks \leq C$ y $Cs \leq F$). Tal evaluación se puede realizar rápidamente en *Maple X*, puesto que como el lector probablemente recuerde en el apartado 2 del presente capítulo se definió *Ks*, *Cs* y *K(Q)* para el caso en estudio.

En la siguiente corrida de *Maple* simplemente se evalúa el *Ks*, *Cs* y *K(Q)* que corresponde a cada uno de los 16 juegos de respuestas anteriores.

>eval([Ks,Cs,K(Q)],[Q1=53,Q2=291,Q3=327,Q4=375]);eval([Ks,Cs,K(Q)],[Q1=53,Q2=291,Q3=327,Q4=376]);eval([Ks,Cs,K(Q)],[Q1=53,Q2=291,Q3=328,Q4=375]);eval([Ks,Cs,K(Q)],[Q1=53,Q2=291,Q3=328,Q4=376]);eval([Ks,Cs,K(Q)],[Q1=53,Q2=292,Q3=327,Q4=375]);eval([Ks,Cs,K(Q)],[Q1=53,Q2=292,Q3=327,Q4=376]);eval([Ks,Cs,K(Q)],[Q1=53,Q2=292,Q3=328,Q4=375]);eval([Ks,Cs,K(Q)],[Q1=53,Q2=292,Q3=328,Q4=376]);eval([Ks,Cs,K(Q)],[Q1=54,Q2=291,Q3=327,Q4=375]);eval([Ks,Cs,K(Q)],[Q1=54,Q2=291,Q3=327,Q4=376]);eval([Ks,Cs,K(Q)],[Q1=54,Q2=291,Q3=328,Q4=375]);eval([Ks,Cs,K(Q)],[Q1=54,Q2=291,Q3=328,Q4=376]);eval([Ks,Cs,K(Q)],[Q1=54,Q2=292,Q3=327,Q4=375]);eval([Ks,Cs,K(Q)],[Q1=54,Q2=292,Q3=327,Q4=376]);eval([Ks,Cs,K(Q)],[Q1=54,Q2=292,Q3=328,Q4=375]);eval([Ks,Cs,K(Q)],[Q1=54,Q2=292,Q3=328,Q4=376]);

[3788, 1445.5, 92140.30593]

[3790, 1447.0, 92140.05203]

[3793, 1446.5, 92139.87358]

[3795, 1448.0, 92139.61968]

[3791, 1446.5, 92140.01750]

[3793, 1448.0, 92139.76360]

[3796, 1447.5, 92139.58515]

[3798, 1449.0, 92139.33125]

[3798, 1450.5, 92139.20949]

[3800, 1452.0, 92138.95559]

[3803, 1451.5, 92138.77714]

[3805, 1453.0, 92138.52324]

[3801, 1451.5, 92138.92106]

[3803, 1453.0, 92138.66716]

[3806, 1452.5, 92138.48871]

[3808, 1454.0, 92138.23481]

En la octava evaluación se alcanza el *K(Q)* mínimo mientras se satisfacen ambas restricciones. De acuerdo con tal evaluación:

- Ks = \$ 3798 < C = \$ 3800
- Cs = 1449 ft^2 < F = 1450 ft^2
- $K(Q)$ = \$ 92139.33125

El juego de respuestas (Q^*_1, Q^*_2, Q^*_3, Q^*_4) que corresponde a la octava evaluación es (53, 292, 328, 376), por lo que las cantidades óptimas para el caso en estudio aproximadas a números enteros son:

- Q^*_1 = 53 bancos del tipo 1
- Q^*_2 = 292 bancos del tipo 2
- Q^*_3 = 328 bancos del tipo 3
- Q^*_4 = 376 bancos del tipo 4

Finalmente se llega a la conclusión de que la Bench Company debe almacenar las siguientes cantidades óptimas de bancos si desea reducir sus costos de inventario al mínimo.

53 bancos del tipo 1
292 bancos del tipo 2
328 bancos del tipo 3
376 bancos del tipo 4

Adicionalmente, el costo total anual promedio del inventario relacionado a las cantidades óptimas anteriores es de:

\$ 92,139.33

Conclusiones

La importancia de las conclusiones en una tesis de licenciatura es evidente, puesto que en ellas se indican hallazgos, aportaciones a la disciplina en estudio y tal vez recomendaciones que puedan resultar útiles a la problemática planteada. En consecuencia, las conclusiones de este documento estarán divididas en cuatro apartados principales en los que se proporcionará la siguiente información:

1. Alcance del software.
2. Un panorama general de lo que el usuario final puede hacer con el programa (*AIRRAM*) y la manera de hacerlo, algo así como un pequeño manual o síntesis.
3. Las principales especificaciones del programa.
4. Y el procedimiento de instalación de *AIRRAM*.

1 ALCANCE DEL SOFTWARE

Si bien un software de las características que se desarrolló puede ser útil en el ejercicio profesional de un Ingeniero Industrial, el objetivo fundamental que se persiguió al desarrollarlo no fue ese, sino el de proporcionar una herramienta acorde a las necesidades de la cátedra en la carrera de Ing. Industrial.

Teniendo en cuenta lo anterior, la solución del problema en estudio se proporciona mediante un software de las siguientes prestaciones generales:

- Cuenta con tres ventanas (interfaces) para el análisis de problemas que se ajustan al modelo de inventarios de artículos múltiples con restricción de recursos (Presupuesto y Espacio). En la primer ventana se analizan problemas que se ajustan al modelo de inventarios en estudio bajo la restricción de Presupuesto, en la segunda bajo la restricción de Espacio y en la tercera bajo ambas restricciones.
- También cuenta con opciones que permiten manipular información de una sesión de trabajo a otra, es decir, opciones que permiten guardar, abrir e imprimir información generada en la aplicación.
- En caso de que se manipule una gran cantidad de información se incorporan opciones que permiten editarla fácilmente, es decir, opciones que permiten borrar, exportar e importar dicha información.
- Se integra software y herramientas para el análisis de modelos generales de inventarios al programa principal.
- Existe la posibilidad de que la aplicación sea distribuida por Internet. Teniendo en cuenta lo anterior, el programa se podría utilizar como un medio para que sus usuarios conozcan la Facultad de Ingeniería por medio del enlace a la página electrónica de la escuela con el que cuenta el programa.
- Por último, se incorpora un sistema de ayuda que brinda soporte al usuario final de la aplicación.

2 SÍNTESIS DEL PROGRAMA

2.1 ¿Qué es *AIRRAM*?

Es una aplicación que analiza el problema del modelo de inventarios de artículos múltiples con restricción de recursos (Presupuesto y Espacio), auxiliándose de tres diferentes ventanas de análisis. También cuenta con software y herramientas para el análisis de modelos general de inventarios.

En la siguiente figura (Figura 8-1) se muestra la ventana principal de *AIRRAM*.

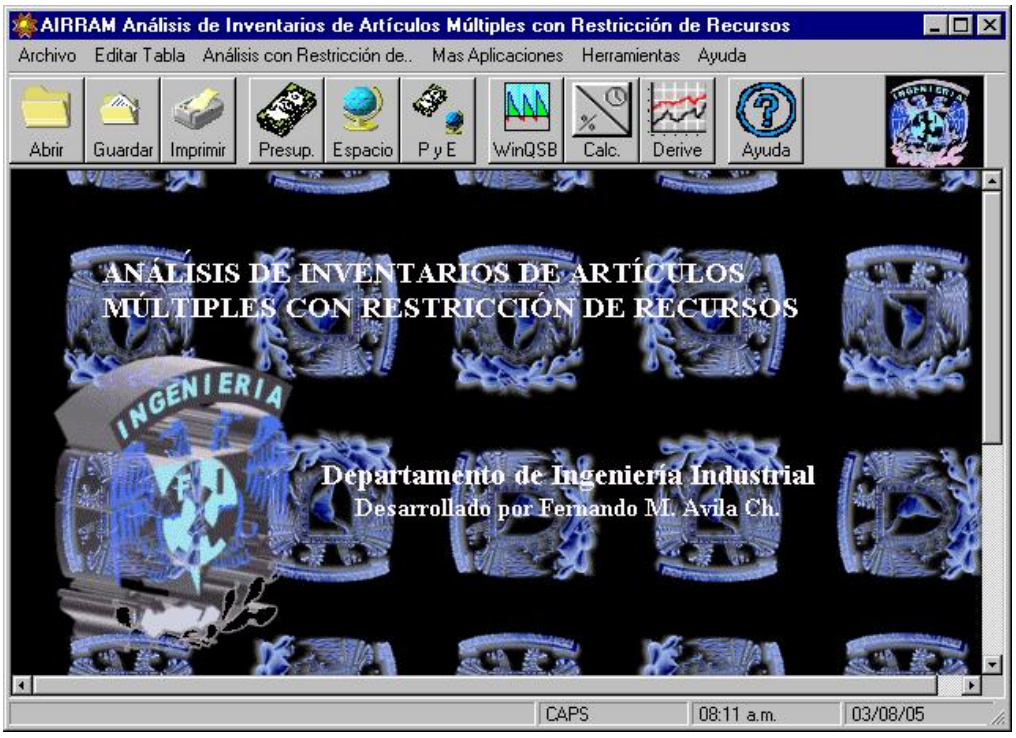


Figura 8-1
Ventana principal de *AIRRAM*

2.2 Variables de las ventanas de análisis

En la siguiente figura (Figura 8-2) se señalan las diferentes variables que se manejan en las ventanas de análisis.

Figura 8-2
Variables de la ventana de *Análisis con Restricción de Presupuesto y Espacio*

Se decidió utilizar como ejemplo la ventana de *Análisis con Restricción de Presupuesto y Espacio* porque es la que cuenta con la totalidad de las variables del modelo en estudio.

En la siguiente lista se define cada una de las variables señaladas en la figura anterior.

Datos:

1. i (i). Es el costo total anual de mantener el inventario, debe ser suministrado en porcentaje anual, es decir, tiene que ser un número real positivo entre 0 y 100.
2. C (C). Es el límite de la inversión total en inventario (restricción de presupuesto), debe ser suministrada en unidades monetarias ($), es decir, tiene que ser un número real positivo.
3. F (F). Es el límite del espacio total en inventario (restricción de espacio), debe ser suministrada en unidades de espacio (ft^2, m^2, etc.), es decir, tiene que ser un número real positivo.
4. n (n). Es el número de clases de artículos en el sistema, tiene que ser un número real, positivo y entero entre 1 y 10000. La tabla de cada ventana de análisis tiene tantos renglones como clases de artículos se reportan en el sistema. A medida que n crece, también crece el tamaño de su respectiva tabla. Cuando en una PC estándar se intentan generar tablas como las de *AIRRAM* de más de 10000 renglones, el sistema operativo se vuelve lento e inestable, es por ello que se decidió definir el límite superior de n en 10000.
5. A (A_i). Es el costo de ordenar para cada clase de artículo en el sistema, debe ser suministrado en unidades monetarias ($), es decir, tiene que ser un número real positivo.
6. D (D_i). Es la demanda para cada clase de artículo en el sistema, debe ser suministrada en unidades por año, es decir, tiene que ser un número real, positivo y entero.

7. c (c_i). Es el costo unitario para cada clase de artículo en el sistema, debe ser suministrado en unidades monetarias ($), es decir, tiene que ser un número real positivo.
8. f (f_i). Es el espacio unitario para cada clase de artículo en el sistema, debe ser suministrado en unidades de espacio (ft^2, m^2, etc.), es decir, tiene que ser un número real positivo.

Resultados:

9. Q* (Q^*_i). Es el tamaño de lote óptimo (cantidad económica a ordenar) para cada clase de artículo en el sistema, es desplegado en unidades, es decir, como un número real positivo.
10. Lambda1 (λ_1). Es el primer multiplicador de Lagrange, es adimensional, es desplegado como un número real positivo.
11. Lambda2 (λ_2). Es el segundo multiplicador de Lagrange, también es adimensional, es desplegado como un número real positivo.
12. K(Q) ($K(Q)$). Es el costo total anual promedio del inventario, es desplegado en unidades monetarias ($), es decir, como un número real positivo.

Las variables C, A_i, y c_i deben ser suministradas con el mismo tipo de unidades, así como F y f_i.

2.3 Ventanas de análisis

La aplicación cuenta con las siguientes ventanas de análisis:

- En la primer ventana es posible resolver problemas que se ajusten al modelo de inventarios en estudio bajo la restricción de presupuesto. En la siguiente figura (Figura 8-3) se muestra la ventana de *Análisis con la Restricción de Presupuesto*.

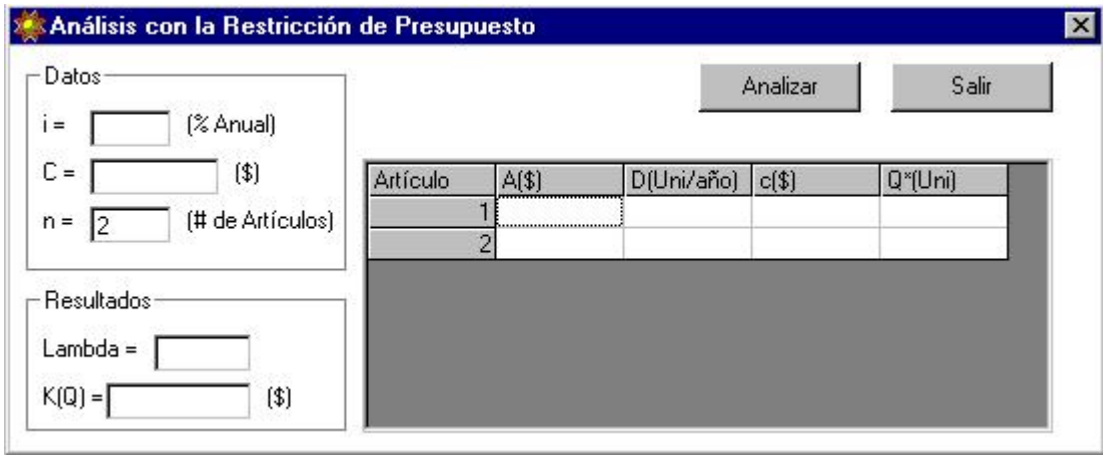


Figura 8-3
Ventana de *Análisis con la Restricción de Presupuesto*

- En la segunda es posible resolver problemas que se ajusten al modelo en estudio bajo la restricción de espacio. En la siguiente figura (Figura 8-4) se muestra la ventana de *Análisis con la Restricción de Espacio*.

Figura 8-4
Ventana de *Análisis con la Restricción de Espacio*

- Y en la tercera es posible resolver problemas que se ajusten al modelo en estudio bajo ambas restricciones (presupuesto y espacio). En la siguiente figura (Figura 8-5) se muestra la ventana de *Análisis con Restricción de Presupuesto y Espacio*.

Figura 8-5
Ventana de *Análisis con Restricción de Presupuesto y Espacio*

El procedimiento general de resolución para los problemas que se ajustan al modelo de inventarios en estudio a través de las ventanas de análisis de *AIRRAM* es:

1. Cargar la ventana adecuada haciendo clic en alguna de las órdenes (o en sus botones asociados) del menú *Análisis con Restricción de.*.
2. Vaciar los datos del problema en la ventana.

3. Una vez que se han vaciado todos los datos pulsar el botón etiquetado con la leyenda *Analizar*. Si el análisis toma más de cinco segundos la aplicación dispara una caja de diálogo que permite continuar o abortar el análisis.
4. Se despliegan los resultados.
5. Si se desea cerrar la ventana pulsar el botón etiquetado con la leyenda *Salir*.

Cuando se ingrese información a la tabla de cada ventana de análisis, el usuario se percatará que sólo se pueden ingresar números, es decir, caracteres de 0 a 9 y el símbolo "*punto*". La aplicación precisamente fue diseñada de esa forma con el propósito de reducir los errores en tiempo de ejecución. Si se presiona la tecla *delete* (suprimir) se borrará el contenido completo de la celda en uso de la tabla, y si se presiona la tecla *backspace* (borrar o retroceder un espacio) un carácter a la vez.

2.4 Menú Archivo

Las órdenes del menú *Archivo* permiten que el usuario final de la aplicación almacene, recupere e imprima información recolectada y generada en *AIRRAM*.

Por otro lado, en la aplicación se pueden generar tres tipos de ficheros de datos, cada uno con su propia extensión.

- Si el fichero tiene una extensión **.AIRR1*, es un fichero que fue creado a partir de información perteneciente a una ventana de *Análisis con la Restricción de Presupuesto*, y por lo tanto, en el momento en que se abra, su información será depositada en una ventana del mismo tipo, en otras palabras, un fichero con extensión **.AIRR1* corresponde a una ventana de *Análisis con la Restricción de Presupuesto*.
- Un fichero con extensión **.AIRR2* corresponde a una ventana de *Análisis con la Restricción de Espacio*.
- Un fichero con extensión **.AIRR3* corresponde a una ventana de *Análisis con Restricción de Presupuesto y Espacio*.

2.4.1 Orden Abrir...

A través de la orden *Abrir...* del menú *Archivo* es posible abrir ficheros de datos generados en *AIRRAM* (**.AIRR1*, **.AIRR2* o **.AIRR3*), y depositar la información que contienen en su respectiva ventana de análisis.

El procedimiento para abrir ficheros de datos generados en *AIRRAM* es:

1. Hacer clic en la orden *Abrir...* (o en su botón asociado) del menú *Archivo*.
2. Una vez que se visualiza la caja de diálogo *Abrir*, seleccionar el tipo de fichero que se desea abrir (**.AIRR1*, **.AIRR2* o **.AIRR3*).
3. Proporcionar la ubicación y el nombre del fichero directamente, o navegando a través del árbol de carpetas de la PC.
4. Pulsar el botón etiquetado con la leyenda *Abrir*.

5. Dependiendo de que tipo de fichero se abrió, su información se deposita en alguna de las ventanas de análisis.

2.4.2 Orden Guardar como...

A través de la orden *Guardar como...* del menú *Archivo* es posible guardar la información que contienen las ventanas de análisis de *AIRRAM* en su respectivo fichero de datos (**.AIRR1*, **.AIRR2* o **.AIRR3*).

El procedimiento para guardar la información de las ventanas de análisis en su respectivo fichero de datos es:

1. Antes que nada tener cargada y activa alguna ventana. Hasta que se cumple la condición anterior, la orden *Guardar como...* y su botón asociado se encuentran deshabilitados.
2. Hacer clic en la orden *Guardar como...* (o en su botón asociado) del menú *Archivo*.
3. Una vez que se visualiza la caja de diálogo *Guardar como*, definir el nombre y la ubicación del fichero. No es necesario especificar el tipo de fichero, pues la aplicación automáticamente asigna el tipo de fichero correspondiente a la ventana de análisis activa al momento de hacer clic en la orden *Guardar como...* (o en su botón asociado) del menú *Archivo*.
4. Pulsar el botón etiquetado con la leyenda *Guardar*.

2.4.3 Orden Imprimir

A través de la orden *Imprimir* del menú *Archivo* es posible imprimir las ventanas de análisis de *AIRRAM*. Con esta orden se imprime la ventana activa "entera", es decir, la ventana, sus controles, mapas de bits, información, etc.

El procedimiento para imprimir las ventanas de análisis es:

1. Tener cargada y activa alguna ventana. Hasta que se cumple la condición anterior, la orden *Imprimir* y su botón asociado se encuentran deshabilitados.
2. Hacer clic en la orden *Imprimir* (o en su botón asociado) del menú *Archivo*.
3. Una vez que se visualiza la caja de diálogo *Imprimir*, fijar las propiedades de impresión (número de copias, tamaño del papel, calidad de la impresión, etc.).
4. Pulsar el botón etiquetado con la leyenda *Aceptar*.

2.4.4 Orden Salir

Al hacer clic en la orden *Salir* del menú *Archivo* se cierra la aplicación (*AIRRAM*).

2.5 Menú Editar Tabla

Las órdenes del menú *Editar Tabla* permiten que el usuario final de la aplicación copie, borre y pegue información en la tabla de cada ventana de análisis.

Para que las órdenes del menú *Editar Tabla* estén habilitadas, tiene que estar seleccionada parte o toda la tabla de alguna ventana de análisis.

2.5.1 Orden Copiar

A través de la orden *Copiar* del menú *Editar Tabla* es posible copiar parte o toda la tabla de alguna ventana de análisis al *Portapapeles* de Windows.

El procedimiento para copiar parte o toda la tabla de la ventana de análisis activa al *Portapapeles* es:

1. Seleccionar el rango de celdas que se desea copiar.
2. Hacer clic en la orden *Copiar* del menú *Editar Tabla*.

Como el *Portapapeles* de Windows ya tiene almacenado el rango de celdas seleccionadas, es posible exportarlas o pegarlas en alguna otra aplicación (*Excel*, por ejemplo), o bien, en la tabla de alguna otra ventana de análisis.

2.5.2 Orden Pegar

A través de la orden *Pegar* del menú *Editar Tabla* es posible pegar el contenido del *Portapapeles* (que debe ser una celda o un grupo de celdas) en la tabla de alguna ventana de análisis.

El procedimiento para pegar el contenido del *Portapapeles* en la tabla de la ventana de análisis activa es:

1. Seleccionar un punto inicial de inserción en la tabla, es decir, seleccionar la celda en donde se comenzarán a pegar las celdas del *Portapapeles*. A partir del punto inicial de inserción debe haber espacio suficiente en la tabla para pegar el contenido del *Portapapeles*. Por ejemplo, si se intenta pegar una tabla de 4×4 (4 renglones y 4 columnas) que viene de *Excel* en la tabla de alguna ventana de análisis de *AIRRAM* en donde a partir de su punto inicial de inserción sólo queda espacio para almacenar una tabla de 3×4 se generará un error de ejecución. Dicho error no interrumpe la aplicación, pero en lo posible debe evitarse.
2. Hacer clic en la orden *Pegar* del menú *Editar Tabla*.

2.5.3 Orden Borrar

A través de la orden *Borrar* del menú *Editar Tabla* es posible borrar parte o toda la tabla de alguna ventana de análisis.

El procedimiento para borrar parte o toda la tabla de la ventana de análisis activa es:

1. Seleccionar el rango de celdas que se desea borrar.
2. Hacer clic en la orden *Borrar* del menú *Editar Tabla*.

2.6 Menú Mas Aplicaciones

Las órdenes del menú *Mas Aplicaciones* permiten que el usuario final de *AIRRAM* cargue programas especialmente diseñados para el análisis de inventarios (modelos generales).

Al hacer clic en la orden *WinQSB (ITS)* (o en su botón asociado) del menú *Mas Aplicaciones* se ejecutará el programa *WinQSB* en su módulo *ITS* (Inventory Theory and System).

Al hacer clic en la orden *The Management Scientist* del menú *Mas Aplicaciones* se ejecutará el programa del mismo nombre.

Al hacer clic en la orden *Plantilla de Excel (IT)* del menú *Mas Aplicaciones* se ejecutará un archivo de *Excel* (*Inventory Theory.xls*) que contiene plantillas para el análisis de modelos generales de inventarios. Para que las plantillas funcionen correctamente hay que habilitar las *Macros* del archivo al iniciar su carga.

Al hacer clic en la orden *Tora* del menú *Mas Aplicaciones* se ejecutará el programa del mismo nombre.[24]

2.7 Menú Herramientas

Las órdenes del menú *Herramientas* permiten que el usuario final de *AIRRAM* cargue programas de análisis matemático.

Al hacer clic en la orden *Calculadora* (o en su botón asociado) del menú *Herramientas* se cargará la calculadora que proporciona el sistema operativo Windows en todas sus versiones. En la siguiente figura (Figura 8-6) se muestra la calculadora de Windows.

[24] Para mayor referencia con respecto a los programas que se llaman desde el menú *Mas Aplicaciones* dirigirse a el capítulo 5 apartado 1 de este documento.

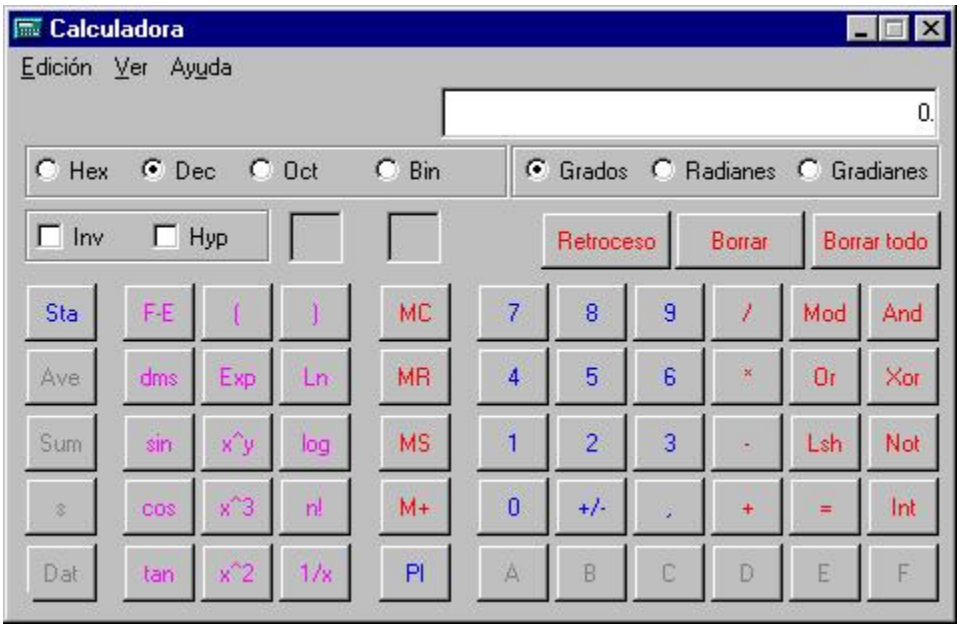


Figura 8-6
Calculadora de Windows

Al hacer clic en la orden *Derive* (o en su botón asociado) del menú *Herramientas* se ejecutará el programa del mismo nombre.[25]

2.8 Menú Ayuda

Las órdenes del menú *Ayuda* permiten que el usuario final cargue un pequeño manual en línea sobre *AIRRAM* y una ventana con información general de la aplicación.

2.8.1 Orden Temas de Ayuda

A través de la orden *Temas de Ayuda* del menú *Ayuda* es posible cargar el fichero de ayuda de la aplicación. Tal fichero se visualiza en el motor de ayuda de Windows (*WinHelp*).

El procedimiento para acceder a la ayuda en línea de *AIRRAM* es:

1. Hacer clic en la orden *Temas de Ayuda* (o en su botón asociado) del menú *Ayuda*.
2. Una vez que se visualiza el fichero de ayuda de *AIRRAM* en *WinHelp* es posible navegar por los diferentes temas de ayuda a través de la ventana del índice, o a través de los *links* (ligas) que se encuentran en la ventana de contenido.

En la siguiente figura (Figura 8-7) se muestra la ventana de contenido del Sistema de Ayuda de *AIRRAM*.

[25] Para mayor referencia con respecto al programa *Derive* dirigirse a el capítulo 5 apartado 2.1 de este documento.

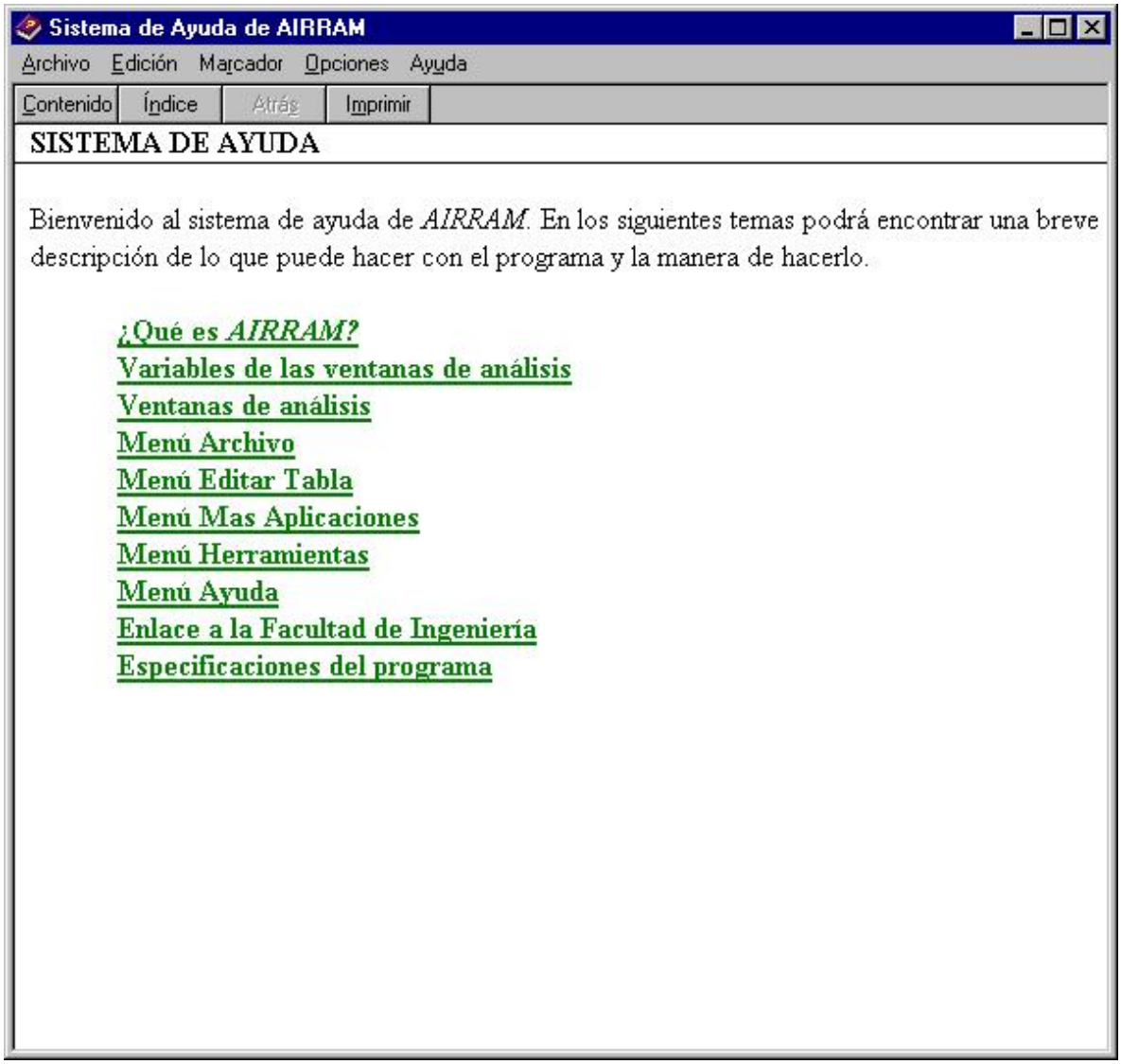


Figura 8-7
Ventana de contenido del Sistema de Ayuda de *AIRRAM*

No se considera necesario profundizar en el manejo de *WinHelp*, ya que cualquier usuario que acostumbre utilizar Windows estará familiarizado con él (casi todas las aplicaciones que corren en Windows lo emplean).

2.8.2 Orden Acerca de AIRRAM

A través de la orden *Acerca de AIRRAM* del menú *Ayuda* es posible cargar una ventana con información general de la aplicación. En la siguiente figura (Figura 8-8) se muestra dicha ventana.

Figura 8-8
Ventana con información general de la aplicación (*AIRRAM*)

El procedimiento para cargar y manipular la ventana con información general de la aplicación es:

1. Hacer clic en la orden *Acerca de AIRRAM* del menú *Ayuda*.
2. Una vez que se visualiza la ventana se puede observar en ella el título de la aplicación, la versión, una pequeña descripción e información de derechos de autor.
3. Si se pulsa el botón etiquetado con la leyenda *Aceptar* se cierra la ventana. Pero, si se pulsa el botón etiquetado con la leyenda *Info. del sistema...* se ejecuta la aplicación *Información del sistema*, la cual muestra una visión general del hardware, de los componentes del sistema y del entorno de software del sistema que se está ejecutando. La siguiente figura (Figura 8-9) muestra una ventana de la aplicación *Información del sistema*.

Figura 8-9
Ventana de la aplicación *Información del sistema*

2.9 Enlace a la Facultad de Ingeniería

La aplicación cuenta con un enlace a la Facultad de Ingeniería, el cual es la animación (un escudo de la Facultad girando sobre si mismo) ubicada en el extremo superior derecho de la ventana principal de *AIRRAM*. En la siguiente figura (Figura 8-10) se muestra dicho enlace.

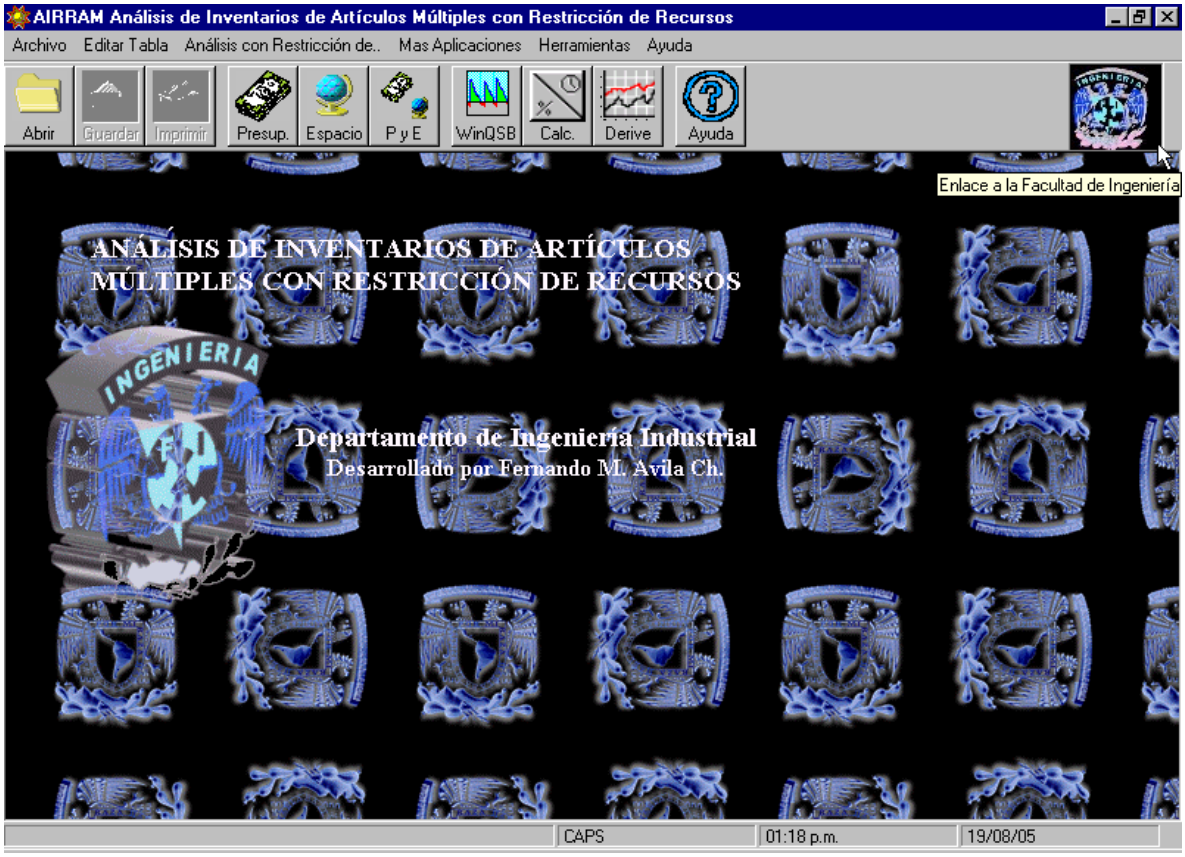


Figura 8-10
Enlace a la Facultad de Ingeniería

Al hacer doble clic sobre la animación antes citada se cargará el navegador *Microsoft Internet Explorer* después de fijar la página electrónica de la Facultad de Ingeniería como la página de inicio. En la siguiente figura (Figura 8-11) se muestra la página de bienvenida de la Facultad.

Figura 8-11
Página de bienvenida de la Facultad de Ingeniería

3 ESPECIFICACIONES DEL PROGRAMA

Las principales especificaciones del programa son:

- La aplicación analiza problemas que se ajustan al modelo de inventarios de artículos múltiples bajo la restricción de presupuesto, espacio o ambas.
- También integra software y herramientas para el análisis de modelos generales de inventarios como el módulo *ITS* (Inventory Theory and System) de *WinQSB*, *The Management Scientist*, plantillas *IT* (Inventory Theory) de *Excel*, *Tora*, la calculadora estándar de Windows y *Derive*.
- Soporta hasta 10,000 clases de artículos.
- La PC en donde corra *AIRRAM* debe tener cargados los programas *Internet Explorer* y *Excel* en cualquiera de sus versiones.
- La configuración regional y de idioma de Windows debe estar especificada en Español -México- (lo que usualmente sucede). En el Panel de Control se encuentra la opción de configuración regional y de idioma del sistema operativo. En otros países se les da un formato diferente a los números, lo que puede provocar que la

aplicación arroje resultados erróneos si Windows no está configurado correctamente.

La PC donde se vaya a correr *AIRRAM* debe cumplir los siguientes requisitos mínimos del sistema:

- Procesador 486DX a 66 MHz o superior.
- 24 MB de memoria RAM o mayor.
- 15 MB de espacio libre en el disco duro.
- Unidad de CD-ROM o DVD-ROM.
- Sistema operativo Windows 98 SE o superior.

4 PROCEDIMIENTO DE INSTALACIÓN

En caso de que se esté utilizando un sistema operativo multiusuario (Windows XP por ejemplo) es preferible instalar la aplicación con privilegios de administrador, es decir, con una cuenta de administrador.

Antes de *AIRRAM* se deben instalar los programas *WinQSB* y *The Management Scientist*.

A continuación se presenta el procedimiento de instalación de *WinQSB*:

1. Localizar el archivo *winqsb.zip* en el CD de instalación.
2. Descomprimir el archivo y depositar su contenido en una nueva carpeta.
3. Abrir la carpeta y ejecutar el instalador (*Setup.exe*).
4. Seguir las instrucciones de instalación. Cuando aparezca la ventana en donde se pide el nombre de usuario (*user name*) y la compañía u organización (*company or organization*) introducir cualquier nombre de usuario y compañía.

Procedimiento de instalación de *The Management Scientist*:

1. Abrir la carpeta *Ms* del CD de instalación.
2. Abrir la carpeta *Instalar* y ejecutar el instalador (*setup.exe*).
3. Seguir las instrucciones de instalación.

Procedimiento de instalación de *AIRRAM*:

1. Abrir la carpeta *Airram* del CD de instalación.
2. Ejecutar el instalador (*setup.exe*).
3. Seguir las instrucciones de instalación.

Puede ser que durante la instalación de *AIRRAM* se informe que el sistema cuenta con una copia más reciente de algún archivo que se intenta instalar (lo que es común con archivos de sistema como los **.sys*, **.ocx*, **.dll*, etc.). En este caso hay que seleccionar la opción que permite guardar el archivo más reciente y continuar con la instalación.

Anexos

1 CÓDIGO COMPLETO ASOCIADO A LA INTERFAZ PRINCIPAL

```
Public AIRR1 As String, AIRR2 As String, AIRR3 As String

Private Declare Function ShellExecute Lib "shell32.dll" Alias "ShellExecuteA" (ByVal
hwnd As Long, ByVal lpOperation As String, ByVal lpFile As String, ByVal lpParameters
As String, ByVal lpDirectory As String, ByVal nShowCmd As Long) As Long

Private Declare Function RegCloseKey Lib "advapi32.dll" (ByVal hKey As Long) As
Long
Private Declare Function RegCreateKey Lib "advapi32.dll" Alias "RegCreateKeyA"
(ByVal hKey As Long, ByVal lpSubKey As String, phkResult As Long) As Long
Private Declare Function RegSetValueEx Lib "advapi32.dll" Alias "RegSetValueExA"
(ByVal hKey As Long, ByVal lpValueName As String, ByVal Reserved As Long, ByVal
dwType As Long, lpData As Any, ByVal cbData As Long) As Long
```

```
Private Sub MDIForm_Load()

MDIForm1.MnuGuardarComo.Enabled = False
Toolbar1.Buttons(2).Enabled = False
MDIForm1.MnuImprimir.Enabled = False
Toolbar1.Buttons(3).Enabled = False

MDIForm1.MnuCopiar.Enabled = False
MDIForm1.MnuBorrar.Enabled = False
MDIForm1.MnuPegar.Enabled = False

Picture1.Left = MDIForm1.Width - 1275



End Sub
```

```
Private Sub MDIForm_Resize()
Picture1.Left = MDIForm1.Width - 1275
End Sub
```

```
Private Sub MnuAbrir_Click()

On Error GoTo ManipularErrorAbrir
  CommonDialog1.CancelError = True
  CommonDialog1.Filter = "Restricción de Presupuesto (*.AIRR1)|*.AIRR1|" & _
                  "Restricción de Espacio (*.AIRR2)|*.AIRR2|" & _
                  "Restricción de Presupuesto y Espacio (*.AIRR3)|*.AIRR3|" & _
                  "Todos (*.*)|*.*"
  CommonDialog1.FilterIndex = 1
  CommonDialog1.ShowOpen

  If CommonDialog1.FileTitle Like "*.AIRR1" Then
  AIRR1 = CommonDialog1.FileName
  AbrirFichero1
  End If

  If CommonDialog1.FileTitle Like "*.AIRR2" Then
  AIRR2 = CommonDialog1.FileName
  AbrirFichero2
  End If

  If CommonDialog1.FileTitle Like "*.AIRR3" Then
  AIRR3 = CommonDialog1.FileName
  AbrirFichero3
  End If

SalirAbrir:
  Exit Sub

ManipularErrorAbrir:
  If Err.Number = cdlCancel Then Exit Sub
  MsgBox Err.Description
  Resume SalirAbrir


End Sub
```

```
Private Sub MnuEspacio_Click()
Load Form2
End Sub
```

```
Private Sub MnuPresupuesto_Click()
Load Form1
End Sub
```

```
Private Sub MnuPyE_Click()
Load Form3
End Sub
```

```
Private Sub Timer1_Timer()

Select Case Picture1.Picture
  Case Empty
  Picture1.Picture = Picture2.Picture
  Case Picture2.Picture
  Picture1.Picture = Picture3.Picture
  Case Picture3.Picture
  Picture1.Picture = Picture4.Picture
  Case Picture4.Picture
  Picture1.Picture = Picture5.Picture
  Case Picture5.Picture
  Picture1.Picture = Picture6.Picture
  Case Picture6.Picture
  Picture1.Picture = Picture7.Picture
  Case Picture7.Picture
  Picture1.Picture = Picture8.Picture
  Case Picture8.Picture
  Picture1.Picture = Picture9.Picture
  Case Picture9.Picture
  Picture1.Picture = Picture10.Picture
  Case Picture10.Picture
  Picture1.Picture = Picture11.Picture
  Case Picture11.Picture
  Picture1.Picture = Picture12.Picture
  Case Picture12.Picture
  Picture1.Picture = Picture2.Picture
End Select

End Sub
```

```
Private Sub Toolbar1_ButtonClick(ByVal Button As MSComctlLib.Button)
```

```
Select Case Button.Key

  Case "BtAbrir"
  MnuAbrir_Click
  Case "BtGuardar"
  MnuGuardarComo_Click
  Case "BtImprimir"
  MnuImprimir_Click
  Case "BtPresup."
  MnuPresupuesto_Click
  Case "BtEspacio"
  MnuEspacio_Click
  Case "BtPyE"
  MnuPyE_Click
  Case "BtWinQSB"
  MnuWinQSB_Click
  Case "BtCalc."
  MnuCalculadora_Click
  Case "BtDerive"
  MnuDerive_Click
  Case "BtAyuda"
  MnuTemasDeAyuda_Click

End Select


End Sub
```

```
Private Sub AbrirFichero1()

Dim Cadena As String
Dim NumFichero As Integer


  NumFichero = FreeFile
  On Error GoTo RutinaErrorAbrir


  Open AIRR1 For Input As #NumFichero

  Load Form1
  Form1.SetFocus

    Input #NumFichero, a
    Form1.Text1.Text = a
```

```
    Input #NumFichero, a
    Form1.Text2.Text = a
    Input #NumFichero, a
    Form1.Text3.Text = a
    Input #NumFichero, a
    Form1.Text4.Text = a
    Input #NumFichero, a
    Form1.Text5.Text = a

a = Form1.Text3.Text
If IsNumeric(a) And (a >= 1) And (a <> "") Then
  If (a / CInt(a)) = 1 Then
    With Form1.MSFlexGrid1
      For j = 1 To Form1.Text3.Text
        .Row = j
        For K = 1 To 4
          .Col = K
          Input #NumFichero, a
          .Text = a
        Next K
      Next j
    End With
  End If
End If

  Close #NumFichero

SalirAbrir:
  Exit Sub
RutinaErrorAbrir:
  MsgBox "Error al abrir el fichero", 48
  Resume SalirAbrir

End Sub
```

```
Private Sub AbrirFichero2()

Dim Cadena As String
Dim NumFichero As Integer


  NumFichero = FreeFile
  On Error GoTo RutinaErrorAbrir
```

```
  Open AIRR2 For Input As #NumFichero

  Load Form2
  Form2.SetFocus

    Input #NumFichero, a
    Form2.Text1.Text = a
    Input #NumFichero, a
    Form2.Text2.Text = a
    Input #NumFichero, a
    Form2.Text3.Text = a
    Input #NumFichero, a
    Form2.Text4.Text = a
    Input #NumFichero, a
    Form2.Text5.Text = a

a = Form2.Text3.Text
If IsNumeric(a) And (a >= 1) And (a <> "") Then
  If (a / CInt(a)) = 1 Then
    With Form2.MSFlexGrid1
      For j = 1 To Form2.Text3.Text
        .Row = j
        For K = 1 To 5
          .Col = K
          Input #NumFichero, a
          .Text = a
        Next K
      Next j
    End With
  End If
End If

  Close #NumFichero

SalirAbrir:
  Exit Sub
RutinaErrorAbrir:
  MsgBox "Error al abrir el fichero", 48
  Resume SalirAbrir

End Sub
```

```
Private Sub AbrirFichero3()
```

```
Dim Cadena As String
Dim NumFichero As Integer


  NumFichero = FreeFile
  On Error GoTo RutinaErrorAbrir


  Open AIRR3 For Input As #NumFichero

  Load Form3
  Form3.SetFocus

    Input #NumFichero, a
    Form3.Text1.Text = a
    Input #NumFichero, a
    Form3.Text6.Text = a
    Input #NumFichero, a
    Form3.Text2.Text = a
    Input #NumFichero, a
    Form3.Text3.Text = a
    Input #NumFichero, a
    Form3.Text4.Text = a
    Input #NumFichero, a
    Form3.Text7.Text = a
    Input #NumFichero, a
    Form3.Text5.Text = a

a = Form3.Text3.Text
If IsNumeric(a) And (a >= 1) And (a <> "") Then
  If (a / CInt(a)) = 1 Then
    With Form3.MSFlexGrid1
      For j = 1 To Form3.Text3.Text
        .Row = j
        For K = 1 To 5
          .Col = K
          Input #NumFichero, a
          .Text = a
        Next K
      Next j
    End With
  End If
End If

  Close #NumFichero

SalirAbrir:
```

```
  Exit Sub
RutinaErrorAbrir:
  MsgBox "Error al abrir el fichero", 48
  Resume SalirAbrir

End Sub
```

```
Private Sub MnuGuardarComo_Click()

If Module1.Bandera = 1 Then

On Error GoTo ManipularErrorGuardar
  CommonDialog1.CancelError = True
  CommonDialog1.Filter = "Restricción de Presupuesto (*.AIRR1)|*.AIRR1|" & _
                  "Todos (*.*)|*.*"
  CommonDialog1.FilterIndex = 1
  CommonDialog1.ShowSave
  AIRR1 = CommonDialog1.FileName
  GuardarFichero1

End If

If Module1.Bandera = 2 Then

On Error GoTo ManipularErrorGuardar
  CommonDialog1.CancelError = True
  CommonDialog1.Filter = "Restricción de Espacio (*.AIRR2)|*.AIRR2|" & _
                  "Todos (*.*)|*.*"
  CommonDialog1.FilterIndex = 1
  CommonDialog1.ShowSave
  AIRR2 = CommonDialog1.FileName
  GuardarFichero2

End If

If Module1.Bandera = 3 Then

On Error GoTo ManipularErrorGuardar
  CommonDialog1.CancelError = True
  CommonDialog1.Filter = "Restricción de Presupuesto y Espacio (*.AIRR3)|*.AIRR3|" &
_"Todos (*.*)|*.*"
  CommonDialog1.FilterIndex = 1
  CommonDialog1.ShowSave
  AIRR3 = CommonDialog1.FileName
  GuardarFichero3
```

```
End If

SalirGuardar:
  Exit Sub
ManipularErrorGuardar:
  If Err.Number = cdlCancel Then Exit Sub
  MsgBox Err.Description
  Resume SalirGuardar

End Sub
```

```
Private Sub GuardarFichero1()

Dim Cadena As String
Dim NumFichero As Integer


  NumFichero = FreeFile
  On Error GoTo RutinaErrorGuardar


  Open AIRR1 For Append As #NumFichero
  If (LOF(NumFichero) <> 0) Then
    Cadena = "El fichero ya existe" & vbCrLf
    Cadena = Cadena & "¿desea sobreescribirlo?"
    If MsgBox(Cadena, 36) = vbYes Then
      Close #NumFichero
      Kill AIRR1
      Open AIRR1 For Output As #NumFichero
    Else
      Close #NumFichero
      Exit Sub
    End If
  End If



    Print #NumFichero, Form1.Text1.Text
    Print #NumFichero, Form1.Text2.Text
    Print #NumFichero, Form1.Text3.Text
    Print #NumFichero, Form1.Text4.Text
    Print #NumFichero, Form1.Text5.Text

 a = Form1.Text3.Text
```

```
 If IsNumeric(a) And (a >= 1) And (a <> "") Then
  If (a / CInt(a)) = 1 Then
    With Form1.MSFlexGrid1
      For j = 1 To Form1.n
        .Row = j
        For K = 1 To 4
          .Col = K
          Print #NumFichero, .Text
        Next K
      Next j
    End With
  End If
 End If

  Close #NumFichero

SalirGuardar:
  Exit Sub
RutinaErrorGuardar:
  MsgBox "Error al abrir el fichero", 48
  Resume SalirGuardar

End Sub
```

```
Private Sub GuardarFichero2()

Dim Cadena As String
Dim NumFichero As Integer


  NumFichero = FreeFile
  On Error GoTo RutinaErrorGuardar


  Open AIRR2 For Append As #NumFichero
  If (LOF(NumFichero) <> 0) Then
    Cadena = "El fichero ya existe" & vbCrLf
    Cadena = Cadena & "¿desea sobreescribirlo?"
    If MsgBox(Cadena, 36) = vbYes Then
      Close #NumFichero
      Kill AIRR2
      Open AIRR2 For Output As #NumFichero
    Else
      Close #NumFichero
      Exit Sub
```

```
   End If
  End If



   Print #NumFichero, Form2.Text1.Text
   Print #NumFichero, Form2.Text2.Text
   Print #NumFichero, Form2.Text3.Text
   Print #NumFichero, Form2.Text4.Text
   Print #NumFichero, Form2.Text5.Text

 a = Form2.Text3.Text
 If IsNumeric(a) And (a >= 1) And (a <> "") Then
  If (a / CInt(a)) = 1 Then
   With Form2.MSFlexGrid1
     For j = 1 To Form2.n
       .Row = j
       For K = 1 To 5
         .Col = K
         Print #NumFichero, .Text
       Next K
     Next j
   End With
  End If
 End If

  Close #NumFichero

SalirGuardar:
  Exit Sub
RutinaErrorGuardar:
  MsgBox "Error al abrir el fichero", 48
  Resume SalirGuardar

End Sub
```

```
Private Sub GuardarFichero3()

Dim Cadena As String
Dim NumFichero As Integer


  NumFichero = FreeFile
  On Error GoTo RutinaErrorGuardar
```

```
 Open AIRR3 For Append As #NumFichero
 If (LOF(NumFichero) <> 0) Then
  Cadena = "El fichero ya existe" & vbCrLf
  Cadena = Cadena & "¿desea sobreescribirlo?"
  If MsgBox(Cadena, 36) = vbYes Then
   Close #NumFichero
   Kill AIRR3
   Open AIRR3 For Output As #NumFichero
  Else
   Close #NumFichero
   Exit Sub
  End If
 End If



  Print #NumFichero, Form3.Text1.Text
  Print #NumFichero, Form3.Text6.Text
  Print #NumFichero, Form3.Text2.Text
  Print #NumFichero, Form3.Text3.Text
  Print #NumFichero, Form3.Text4.Text
  Print #NumFichero, Form3.Text7.Text
  Print #NumFichero, Form3.Text5.Text

a = Form3.Text3.Text
If IsNumeric(a) And (a >= 1) And (a <> "") Then
 If (a / CInt(a)) = 1 Then
  With Form3.MSFlexGrid1
   For j = 1 To Form3.n
    .Row = j
    For K = 1 To 5
     .Col = K
     Print #NumFichero, .Text
    Next K
   Next j
  End With
 End If
End If

 Close #NumFichero

SalirGuardar:
 Exit Sub
RutinaErrorGuardar:
 MsgBox "Error al abrir el fichero", 48
```

```
  Resume SalirGuardar

End Sub
```

```
Private Sub MnuImprimir_Click()


  On Error GoTo ManipularErrorImprimir
  CommonDialog2.CancelError = True
  CommonDialog2.ShowPrinter



     If Module1.Bandera = 1 Then
     Form1.PrintForm
     End If
     If Module1.Bandera = 2 Then
     Form2.PrintForm
     End If
     If Module1.Bandera = 3 Then
     Form3.PrintForm
     End If



SalirImprimir:
  Exit Sub
ManipularErrorImprimir:
  If Err.Number = cdlCancel Then Exit Sub
  MsgBox Err.Description
  Resume SalirImprimir

End Sub
```

```
Private Sub MnuSalir_Click()
End
End Sub
```

```
Private Sub MnuCopiar_Click()
```

```
On Error GoTo ManipularError

If Module1.Bandera = 1 Then
CopiarTabla1
End If

If Module1.Bandera = 2 Then
CopiarTabla2
End If


If Module1.Bandera = 3 Then
CopiarTabla3
End If


Salir:
  Exit Sub
ManipularError:
  MsgBox Err.Description
  Resume Salir

End Sub
```

```
Private Sub CopiarTabla1()

With Form1.MSFlexGrid1
  Dim TextoSelec As String
  TextoSelec = .Clip
  Clipboard.Clear
  Clipboard.SetText TextoSelec
.Row = 1
.Col = 1
End With

End Sub
```

```
Private Sub CopiarTabla2()

With Form2.MSFlexGrid1
  Dim TextoSelec As String
  TextoSelec = .Clip
  Clipboard.Clear
```

```
  Clipboard.SetText TextoSelec
.Row = 1
.Col = 1
End With

End Sub
```

```
Private Sub CopiarTabla3()

With Form3.MSFlexGrid1
  Dim TextoSelec As String
  TextoSelec = .Clip
  Clipboard.Clear
  Clipboard.SetText TextoSelec
.Row = 1
.Col = 1
End With

End Sub
```

```
Private Sub MnuPegar_Click()

On Error GoTo ManipularError

If Module1.Bandera = 1 Then
PegarTabla1
End If

If Module1.Bandera = 2 Then
PegarTabla2
End If


If Module1.Bandera = 3 Then
PegarTabla3
End If


Salir:
  Exit Sub
ManipularError:
  MsgBox Err.Description
```

```
  Resume Salir

End Sub
```

```
Private Sub PegarTabla1()

With Form1.MSFlexGrid1

Original = .Col
Tabla = Clipboard.GetText

For i = 1 To Len(Tabla)
car = Mid(Tabla, i, 1)
    If car Like "[1234567890.]" Then
    celda = celda & car
    If i = Len(Tabla) Then
    .Text = celda
    celda = ""
    car = ""
    Exit Sub
    End If
    Else
    pal = Mid(Tabla, i, 2)
      If pal = vbCrLf Then
      .Text = celda
      celda = ""
      car = ""
         If i = Len(Tabla) - 1 Then
         Exit Sub
         Else
         .Row = .Row + 1
         .Col = Original
         i = i + 1
         GoTo 10
         End If
      End If
      If car = vbCr Then
      .Text = celda
      celda = ""
      car = ""
         If i = Len(Tabla) Then
         Exit Sub
         Else
         .Row = .Row + 1
```

```
          .Col = Original
          GoTo 10
          End If
       End If
    .Text = celda
    celda = ""
    .Col = .Col + 1
    car = ""
    End If
10: Next i

End With

End Sub
```

```
Private Sub PegarTabla2()

With Form2.MSFlexGrid1

Original = .Col
Tabla = Clipboard.GetText

For i = 1 To Len(Tabla)
car = Mid(Tabla, i, 1)
    If car Like "[1234567890.]" Then
    celda = celda & car
    If i = Len(Tabla) Then
    .Text = celda
    celda = ""
    car = ""
    Exit Sub
    End If
    Else
    pal = Mid(Tabla, i, 2)
       If pal = vbCrLf Then
       .Text = celda
       celda = ""
       car = ""
          If i = Len(Tabla) - 1 Then
          Exit Sub
          Else
          .Row = .Row + 1
          .Col = Original
          i = i + 1
          GoTo 10
```

```
        End If
      End If
      If car = vbCr Then
      .Text = celda
      celda = ""
      car = ""
         If i = Len(Tabla) Then
         Exit Sub
         Else
         .Row = .Row + 1
         .Col = Original
         GoTo 10
         End If
      End If
   .Text = celda
   celda = ""
   .Col = .Col + 1
   car = ""
   End If
10: Next i

End With

End Sub
```

```
Private Sub PegarTabla3()

With Form3.MSFlexGrid1

Original = .Col
Tabla = Clipboard.GetText

For i = 1 To Len(Tabla)
car = Mid(Tabla, i, 1)
   If car Like "[1234567890.]" Then
   celda = celda & car
   If i = Len(Tabla) Then
   .Text = celda
   celda = ""
   car = ""
   Exit Sub
   End If
   Else
   pal = Mid(Tabla, i, 2)
      If pal = vbCrLf Then
```

```
      .Text = celda
      celda = ""
      car = ""
         If i = Len(Tabla) - 1 Then
         Exit Sub
         Else
         .Row = .Row + 1
         .Col = Original
         i = i + 1
         GoTo 10
         End If
      End If
      If car = vbCr Then
      .Text = celda
      celda = ""
      car = ""
         If i = Len(Tabla) Then
         Exit Sub
         Else
         .Row = .Row + 1
         .Col = Original
         GoTo 10
         End If
      End If
   .Text = celda
   celda = ""
   .Col = .Col + 1
   car = ""
   End If
10: Next i

End With

End Sub
```

```
Private Sub MnuBorrar_Click()

On Error GoTo ManipularError

If Module1.Bandera = 1 Then
Form1.MSFlexGrid1.FillStyle = flexFillRepeat
Form1.MSFlexGrid1.Text = ""
Form1.MSFlexGrid1.FillStyle = flexFillSingle
End If
```

```
If Module1.Bandera = 2 Then
Form2.MSFlexGrid1.FillStyle = flexFillRepeat
Form2.MSFlexGrid1.Text = ""
Form2.MSFlexGrid1.FillStyle = flexFillSingle
End If


If Module1.Bandera = 3 Then
Form3.MSFlexGrid1.FillStyle = flexFillRepeat
Form3.MSFlexGrid1.Text = ""
Form3.MSFlexGrid1.FillStyle = flexFillSingle
End If


Salir:
  Exit Sub
ManipularError:
  MsgBox Err.Description
  Resume Salir

End Sub
```

```
Private Sub MnuWinQSB_Click()

On Error GoTo ManipularError

Dim X As Long, APLIC As String, hWndAct As Long
APLIC = "C:\ARCHIVOS DE PROGRAMA\AIRRAM\ITS.EXE"
X = ShellExecute(hwnd, "open", APLIC, vbNullString, vbNullString, SW_SHOW)

If X <= 32 Then
MsgBox "No se encontró la aplicación"
End If

Salir:
  Exit Sub
ManipularError:
  MsgBox Err.Description
  Resume Salir

End Sub
```

```
Private Sub MnuTheManagementScientist_Click()
```

```
On Error GoTo ManipularError

Dim X As Long, APLIC As String, hWndAct As Long
APLIC = "C:\ARCHIVOS DE PROGRAMA\AIRRAM\MS40.EXE"
X = ShellExecute(hwnd, "open", APLIC, vbNullString, vbNullString, SW_SHOW)

If X <= 32 Then
MsgBox "No se encontró la aplicación"
End If

Salir:
  Exit Sub
ManipularError:
  MsgBox Err.Description
  Resume Salir

End Sub
```

```
Private Sub MnuPlantilladeExcel_Click()

On Error GoTo ManipularError

Dim X As Long, APLIC As String, hWndAct As Long
APLIC = "C:\ARCHIVOS DE PROGRAMA\AIRRAM\INVENTORY THEORY.XLS"
X = ShellExecute(hwnd, "open", APLIC, vbNullString, vbNullString, SW_SHOW)

If X <= 32 Then
MsgBox "No se encontró la aplicación"
End If

Salir:
  Exit Sub
ManipularError:
  MsgBox Err.Description
  Resume Salir

End Sub
```

```
Private Sub MnuTora_Click()

On Error GoTo ManipularError
```

```
Shell "C:\ARCHIVOS DE PROGRAMA\AIRRAM\TORA.EXE", vbNormalFocus

Salir:
  Exit Sub
ManipularError:
  MsgBox Err.Description
  Resume Salir

End Sub
```

```
Private Sub MnuCalculadora_Click()

On Error GoTo ManipularError

Dim X As Long, APLIC As String, hWndAct As Long
APLIC = "C:\ARCHIVOS DE PROGRAMA\AIRRAM\CALC.EXE"
X = ShellExecute(hwnd, "open", APLIC, vbNullString, vbNullString, SW_SHOW)

If X <= 32 Then
MsgBox "No se encontró la aplicación"
End If

Salir:
  Exit Sub
ManipularError:
  MsgBox Err.Description
  Resume Salir

End Sub
```

```
Private Sub MnuDerive_Click()

On Error GoTo ManipularError

Shell "C:\ARCHIVOS DE PROGRAMA\AIRRAM\DERIVE.EXE", vbNormalFocus

Salir:
  Exit Sub
ManipularError:
  MsgBox Err.Description
  Resume Salir
```

```
End Sub
```

```
Private Sub Picture1_DblClick()

On Error GoTo ManipularError

Dim strString As String
 strString = "http://www.ingenieria.unam.mx/"
 SaveString HKEY_CURRENT_USER, "Software\Microsoft\Internet Explorer\Main",
"Start Page", strString

aplic = "C:\ARCHIVOS DE PROGRAMA\INTERNET EXPLORER\IEXPLORE.EXE"
X = ShellExecute(hwnd, "open", aplic, vbNullString, vbNullString, SW_SHOW)

Salir:
  Exit Sub
ManipularError:
  MsgBox Err.Description
  Resume Salir

End Sub
```

```
Private Sub SaveString(hKey As Long, strPath As String, strValue As String, strData As
String)

    Dim Ret
    RegCreateKey hKey, strPath, Ret
    RegSetValueEx Ret, strValue, 0, REG_SZ, ByVal strData, Len(strData)
    RegCloseKey Ret

End Sub
```

```
Private Sub MnuTemasDeAyuda_Click()
SendKeys "{F1}"
End Sub
```

```
Private Sub MnuAcercaDeAIRRAM_Click()
frmAbout.Show vbModal
```

```
End Sub
```

2 CÓDIGO COMPLETO ASOCIADO AL FORMULARIO FORM1

```
Public n, i, C As Double, Cs As Double
```

```
Private Sub Paso1Calcular()
Cs = 0
With MSFlexGrid1
For j = 1 To n
   .Row = j
   For K = 1 To 4
      .Col = K
        Select Case K
          Case 1: A = .Text
          Case 2: D = .Text
          Case 3: Cm = .Text
          Case 4:
          Qop = Sqr((2 * A * D) / (i * Cm))
          cu = Cm * Qop
        End Select
   Next K
   Cs = Cs + cu

Next j
End With

End Sub
```

```
Private Sub Paso1Escribir()
KT = 0
With MSFlexGrid1
For j = 1 To n
   .Row = j
   For K = 1 To 4
      .Col = K
        Select Case K
          Case 1: A = .Text
          Case 2: D = .Text
          Case 3: Cm = .Text
          Case 4:
          Qop = Sqr((2 * A * D) / (i * Cm))
```

```
            .Text = CSng(Qop)
            Kp = (Cm * D) + ((A * D) / Qop) + ((Cm * i) * (Qop / 2))
          End Select
    Next K
    KT = KT + Kp
Next j
Text5.Text = KT
End With

End Sub
```

```
Private Sub Paso2Calcular()
S = 0
KT = 0

With MSFlexGrid1
For j = 1 To n
    .Row = j
    For K = 1 To 3
        .Col = K
          Select Case K
            Case 1: A = .Text
            Case 2: D = .Text
            Case 3: Cm = .Text
            End Select
    Next K
    T = Sqr((2 * A * D * Cm))
    S = S + T
Next j

Lambda = ((1 / (2 * C * C)) * (S * S)) - (i / 2)
Text4.Text = Lambda

For j = 1 To n
    .Row = j
    For K = 1 To 4
        .Col = K
          Select Case K
            Case 1: A = .Text
            Case 2: D = .Text
            Case 3: Cm = .Text
            Case 4:
            Qop = Sqr((2 * A * D) / ((i * Cm) + (2 * Lambda * Cm)))
            .Text = CSng(Qop)
            Kp = (Cm * D) + ((A * D) / Qop) + ((Cm * i) * (Qop / 2))
```

```
        End Select
    Next K
    KT = KT + Kp
Next j

Text5.Text = KT

End With

End Sub
```

```
Private Sub Command1_Click()

On Error GoTo ManipularError

i = Text1.Text
CP = Text2.Text
n = Text3.Text

If IsNumeric(i) And IsNumeric(CP) And IsNumeric(n) And (i > 0) And (i <= 100) And (CP
>= 0) And (n >= 1) And (i <> "") And (CP <> "") And (n <> "") Then
  If (n / CInt(n)) = 1 Then

    With MSFlexGrid1
     For j = 1 To n
       .Row = j
        For K = 1 To 3
          .Col = K
           If .Text = "" Then
           MsgBox "Alguna o algunas de las celdas de la tabla estan vacías", 16
           Exit Sub
           Else

           End If
        Next K
     Next j
    End With

  C = CP

  i = i / 100

  Call Paso1Calcular

    If (Cs <= C) Then
```

```
    Call Paso1Escribir
    Text4.Text = ""
    Else
    Call Paso2Calcular
    End If


  Else
  MsgBox "Dato inválido ¡Revisar el frame de Datos!", 16
  End If


Else
MsgBox "Dato inválido ¡Revisar el frame de Datos!", 16
End If

ManipularError:
   If Err.Number = 0 Then
   Exit Sub
   Else
   MsgBox "Dato inválido:" & Err.Description, 16
   Err.Clear
   Exit Sub
   End If

End Sub
```

```
Private Sub Command2_Click()
Unload Form1
End Sub
```

```
Private Sub Form_Activate()
MDIForm1.MnuGuardarComo.Enabled = True
MDIForm1.Toolbar1.Buttons(2).Enabled = True


MDIForm1.MnuImprimir.Enabled = True
MDIForm1.Toolbar1.Buttons(3).Enabled = True

Module1.Bandera = 1

End Sub
```

```
Private Sub Form_Deactivate()
MDIForm1.MnuGuardarComo.Enabled = False
MDIForm1.Toolbar1.Buttons(2).Enabled = False

MDIForm1.MnuImprimir.Enabled = False
MDIForm1.Toolbar1.Buttons(3).Enabled = False

Module1.Bandera = 0

End Sub
```

```
Private Sub Form_Load()
n = Text3.Text

MSFlexGrid1.Rows = n + 1

With MSFlexGrid1

  .Row = 0
  For i = 0 To 4
    .Col = i
    Select Case i
      Case 0: .Text = "Artículo"
      Case 1: .Text = "A($)"
      Case 2: .Text = "D(Uni/año)"
      Case 3: .Text = "c($)"
      Case 4: .Text = "Q*(Uni)"
    End Select
  Next i

  Number = 0
  .Col = 0
  For i = 1 To n
    .Row = i
    Number = Number + 1
      .Text = Number
  Next i

.Row = 1
.Col = 1

End With
```

```
End Sub
```

```
Private Sub Form_Unload(Cancel As Integer)

MDIForm1.MnuGuardarComo.Enabled = False
MDIForm1.Toolbar1.Buttons(2).Enabled = False


MDIForm1.MnuImprimir.Enabled = False
MDIForm1.Toolbar1.Buttons(3).Enabled = False


Module1.Bandera = 0

MDIForm1.MnuCopiar.Enabled = False
MDIForm1.MnuBorrar.Enabled = False
MDIForm1.MnuPegar.Enabled = False

End Sub
```

```
Private Sub MSFlexGrid1_GotFocus()
MDIForm1.MnuCopiar.Enabled = True
MDIForm1.MnuBorrar.Enabled = True
MDIForm1.MnuPegar.Enabled = True
End Sub
```

```
Private Sub MSFlexGrid1_KeyPress(KeyAscii As Integer)

If KeyAscii >= 46 And KeyAscii <= 57 Then
MSFlexGrid1.Text = MSFlexGrid1.Text & Chr(KeyAscii)
End If

End Sub
```

```
Private Sub MSFlexGrid1_KeyUp(KeyCode As Integer, Shift As Integer)

Select Case KeyCode
  Case vbKeyDelete
```

```
    MSFlexGrid1.Text = ""
  Case vbKeyBack
    If Len(MSFlexGrid1.Text) > 0 Then
       MSFlexGrid1.Text = Left(MSFlexGrid1.Text, Len(MSFlexGrid1.Text) - 1)
    End If
End Select

End Sub
```

```
Private Sub MSFlexGrid1_LostFocus()
MDIForm1.MnuCopiar.Enabled = False
MDIForm1.MnuBorrar.Enabled = False
MDIForm1.MnuPegar.Enabled = False
End Sub
```

```
Private Sub Text3_Change()

On Error GoTo ManipularError

n = Text3.Text

If n <> "" And n > 0 And IsNumeric(n) And (n <= 10000) Then
    If (n / CInt(n)) = 1 Then
    MSFlexGrid1.Rows = n + 1
    Else
    MSFlexGrid1.Rows = 1
    End If
Else
MSFlexGrid1.Rows = 1
End If

With MSFlexGrid1

   .Row = 0
   For i = 0 To 4
     .Col = i
     Select Case i
       Case 0: .Text = "Artículo"
       Case 1: .Text = "A($)"
       Case 2: .Text = "D(Uni/año)"
       Case 3: .Text = "c($)"
       Case 4: .Text = "Q*(Uni)"
     End Select
```

```
  Next i


If n <> "" And n > 0 And IsNumeric(n) And (n <= 10000) Then
  If (n / CInt(n)) = 1 Then
  Number = 0
  .Col = 0
  For i = 1 To n
    .Row = i
    Number = Number + 1
      .Text = Number
  Next i
  Else

  End If
Else

End If

End With

ManipularError:
   If Err.Number = 0 Then
   Exit Sub
   Else
   MsgBox "Dato inválido:" & Err.Description, 16
   Err.Clear
   Exit Sub
   End If

End Sub
```

3 CÓDIGO COMPLETO ASOCIADO AL FORMULARIO FORM2

```
Public n, i, F As Double, Cs As Double
```

```
Public Function FNF(Lambda As Variant)
S = 0
With MSFlexGrid1
For j = 1 To n
   .Row = j
   For K = 1 To 4
      .Col = K
```

```
        Select Case K
          Case 1: A = .Text
          Case 2: D = .Text
          Case 3: Cm = .Text
          Case 4: Fm = .Text
          End Select
   Next K
   T = (Sqr((2 * A * D) / ((i * Cm) + (2 * Lambda * Fm)))) * (Fm)
   S = S + T
Next j

FNF = S - F

End With

End Function
```

```
Private Sub Paso2Calcular()

KT = 0
Contador = 0
Sum = 0

LI = 0
LS = 10

While (FNF(LI) * FNF(LS)) >= 0
  LS = LS * 10
  Contador = Contador + 1
  If Contador >= 10 Then
  MsgBox "No existe Lambda", 16
  Exit Sub
  End If
Wend




30:
varAntes = Timer
X = (LI + LS) / 2
40: If FNF(X) = 0 Then GoTo 100
50: If (FNF(LI) * FNF(X)) <= 0 Then GoTo 70
60: LI = X
```

```
65: GoTo 80
70: LS = X
80: If (Abs(LS - LI) / (Abs(LI) + Abs(LS))) < 0.0001 Then GoTo 100
90:
varDespues = Timer
Dif = varDespues - varAntes
Sum = Sum + Dif
If Sum >= 5 Then
Sum = 0
Resp = MsgBox("El análisis tomará algunos segundos más, ¿Desea continuar?", vbYesNo)
  If Resp = 7 Then Exit Sub
End If
GoTo 30
100: Lambda = X



With MSFlexGrid1

Text4.Text = Lambda

For j = 1 To n
   .Row = j
   For K = 1 To 5
      .Col = K
        Select Case K
          Case 1: A = .Text
          Case 2: D = .Text
          Case 3: Cm = .Text
          Case 4: Fm = .Text
          Case 5:
          Qop = Sqr((2 * A * D) / ((i * Cm) + (2 * Lambda * Fm)))
          .Text = CSng(Qop)
          Kp = (Cm * D) + ((A * D) / Qop) + ((Cm * i) * (Qop / 2))
        End Select
   Next K
   KT = KT + Kp
Next j

Text5.Text = KT

End With
End Sub
```

```
Private Sub Paso1Escribir()
KT = 0
With MSFlexGrid1
For j = 1 To n
   .Row = j
   For K = 1 To 5
      .Col = K
        Select Case K
          Case 1: A = .Text
          Case 2: D = .Text
          Case 3: Cm = .Text
          Case 4:
          Case 5:
          Qop = Sqr((2 * A * D) / (i * Cm))
          .Text = CSng(Qop)
          Kp = (Cm * D) + ((A * D) / Qop) + ((Cm * i) * (Qop / 2))
        End Select
   Next K
   KT = KT + Kp
Next j
Text5.Text = KT
End With

End Sub
```

```
Private Sub Paso1Calcular()
Cs = 0
With MSFlexGrid1
For j = 1 To n
   .Row = j
   For K = 1 To 4
      .Col = K
        Select Case K
          Case 1: A = .Text
          Case 2: D = .Text
          Case 3: Cm = .Text
          Case 4:
          Fm = .Text
          Qop = Sqr((2 * A * D) / (i * Cm))
          cu = Fm * Qop
        End Select
   Next K
```

```
   Cs = Cs + cu

Next j
End With

End Sub
```

```
Private Sub Command1_Click()

On Error GoTo ManipularError

i = Text1.Text
CP = Text2.Text
n = Text3.Text

If IsNumeric(i) And IsNumeric(CP) And IsNumeric(n) And (i > 0) And (i <= 100) And (CP
>= 0) And (n >= 1) And (i <> "") And (CP <> "") And (n <> "") Then
  If (n / CInt(n)) = 1 Then

    With MSFlexGrid1
     For j = 1 To n
       .Row = j
         For K = 1 To 4
           .Col = K
            If .Text = "" Then
            MsgBox "Alguna o algunas de las celdas de la tabla estan vacías", 16
            Exit Sub
            Else

            End If
         Next K
     Next j
    End With

  F = CP

  i = i / 100

  Call Paso1Calcular

    If (Cs <= F) Then
    Call Paso1Escribir
    Text4.Text = ""
    Else
    Call Paso2Calcular
```

```
    End If


  Else
  MsgBox "Dato inválido ¡Revisar el frame de Datos!", 16
  End If


Else
MsgBox "Dato inválido ¡Revisar el frame de Datos!", 16
End If

ManipularError:
   If Err.Number = 0 Then
   Exit Sub
   Else
   MsgBox "Dato inválido:" & Err.Description, 16
   Err.Clear
   Exit Sub
   End If

End Sub
```

```
Private Sub Command2_Click()
Unload Form2
End Sub
```

```
Private Sub Form_Activate()
MDIForm1.MnuGuardarComo.Enabled = True
MDIForm1.Toolbar1.Buttons(2).Enabled = True


MDIForm1.MnuImprimir.Enabled = True
MDIForm1.Toolbar1.Buttons(3).Enabled = True

Module1.Bandera = 2

End Sub
```

```
Private Sub Form_Deactivate()
```

```
MDIForm1.MnuGuardarComo.Enabled = False
MDIForm1.Toolbar1.Buttons(2).Enabled = False

MDIForm1.MnuImprimir.Enabled = False
MDIForm1.Toolbar1.Buttons(3).Enabled = False

Module1.Bandera = 0

End Sub
```

```
Private Sub Form_Load()
n = Text3.Text

MSFlexGrid1.Rows = n + 1

With MSFlexGrid1

  .Row = 0
  For i = 0 To 5
    .Col = i
    Select Case i
      Case 0: .Text = "Artículo"
      Case 1: .Text = "A($)"
      Case 2: .Text = "D(Uni/año)"
      Case 3: .Text = "c($)"
      Case 4: .Text = "f(U de Esp)"
      Case 5: .Text = "Q*(Uni)"
    End Select
  Next i

  Number = 0
  .Col = 0
  For i = 1 To n
    .Row = i
    Number = Number + 1
       .Text = Number
  Next i

.Row = 1
.Col = 1

End With
```

```
End Sub
```

```
Private Sub Form_Unload(Cancel As Integer)
MDIForm1.MnuGuardarComo.Enabled = False
MDIForm1.Toolbar1.Buttons(2).Enabled = False

MDIForm1.MnuImprimir.Enabled = False
MDIForm1.Toolbar1.Buttons(3).Enabled = False

Module1.Bandera = 0

MDIForm1.MnuCopiar.Enabled = False
MDIForm1.MnuBorrar.Enabled = False
MDIForm1.MnuPegar.Enabled = False

End Sub
```

```
Private Sub MSFlexGrid1_GotFocus()
MDIForm1.MnuCopiar.Enabled = True
MDIForm1.MnuBorrar.Enabled = True
MDIForm1.MnuPegar.Enabled = True
End Sub
```

```
Private Sub MSFlexGrid1_KeyPress(KeyAscii As Integer)

If KeyAscii >= 46 And KeyAscii <= 57 Then
MSFlexGrid1.Text = MSFlexGrid1.Text & Chr(KeyAscii)
End If

End Sub
```

```
Private Sub MSFlexGrid1_KeyUp(KeyCode As Integer, Shift As Integer)

Select Case KeyCode
  Case vbKeyDelete
    MSFlexGrid1.Text = ""
  Case vbKeyBack
    If Len(MSFlexGrid1.Text) > 0 Then
      MSFlexGrid1.Text = Left(MSFlexGrid1.Text, Len(MSFlexGrid1.Text) - 1)
```

```
   End If
End Select

End Sub
```

```
Private Sub MSFlexGrid1_LostFocus()
MDIForm1.MnuCopiar.Enabled = False
MDIForm1.MnuBorrar.Enabled = False
MDIForm1.MnuPegar.Enabled = False
End Sub
```

```
Private Sub Text3_Change()

On Error GoTo ManipularError

n = Text3.Text

If n <> "" And n > 0 And IsNumeric(n) And (n <= 10000) Then
   If (n / CInt(n)) = 1 Then
   MSFlexGrid1.Rows = n + 1
   Else
   MSFlexGrid1.Rows = 1
   End If
Else
MSFlexGrid1.Rows = 1
End If

With MSFlexGrid1

  .Row = 0
  For i = 0 To 5
    .Col = i
    Select Case i
      Case 0: .Text = "Artículo"
      Case 1: .Text = "A($)"
      Case 2: .Text = "D(Uni/año)"
      Case 3: .Text = "c($)"
      Case 4: .Text = "f(U de Esp)"
      Case 5: .Text = "Q*(Uni)"
    End Select
  Next i
```

```
If n <> "" And n > 0 And IsNumeric(n) And (n <= 10000) Then
  If (n / CInt(n)) = 1 Then
  Number = 0
  .Col = 0
  For i = 1 To n
    .Row = i
    Number = Number + 1
      .Text = Number
  Next i
  Else

  End If
Else

End If

End With

ManipularError:
   If Err.Number = 0 Then
   Exit Sub
   Else
   MsgBox "Dato inválido:" & Err.Description, 16
   Err.Clear
   Exit Sub
   End If

End Sub
```

4 CÓDIGO COMPLETO ASOCIADO AL FORMULARIO FORM3

```
Public n, i, C As Double, F As Double, Ks As Double, Cs As Double
```

```
Public Function FNF(Lambda As Variant)
S = 0
With MSFlexGrid1
For j = 1 To n
   .Row = j
   For K = 1 To 4
      .Col = K
        Select Case K
          Case 1: A = .Text
          Case 2: D = .Text
```

```
            Case 3: Cm = .Text
            Case 4: Fm = .Text
            End Select
    Next K
    T = (Sqr((2 * A * D) / ((i * Cm) + (2 * Lambda * Fm)))) * (Fm)
    S = S + T
Next j

FNF = S - F

End With

End Function
```

```
Private Sub Paso2Escribir()
S = 0
KT = 0

With MSFlexGrid1
For j = 1 To n
    .Row = j
    For K = 1 To 3
        .Col = K
          Select Case K
            Case 1: A = .Text
            Case 2: D = .Text
            Case 3: Cm = .Text
            End Select
    Next K
    T = Sqr((2 * A * D * Cm))
    S = S + T
Next j

Lambda = ((1 / (2 * C * C)) * (S * S)) - (i / 2)
Text4.Text = Lambda

For j = 1 To n
    .Row = j
    For K = 1 To 5
        .Col = K
          Select Case K
            Case 1: A = .Text
            Case 2: D = .Text
            Case 3: Cm = .Text
            Case 4:
```

```
            Case 5:
            Qop = Sqr((2 * A * D) / ((i * Cm) + (2 * Lambda * Cm)))
            .Text = CSng(Qop)
            Kp = (Cm * D) + ((A * D) / Qop) + ((Cm * i) * (Qop / 2))
          End Select
    Next K
    KT = KT + Kp
Next j

Text5.Text = KT

End With
End Sub
```

```
Private Sub Paso2Calcular()
S = 0
Cs = 0

With MSFlexGrid1
For j = 1 To n
    .Row = j
    For K = 1 To 3
        .Col = K
          Select Case K
            Case 1: A = .Text
            Case 2: D = .Text
            Case 3: Cm = .Text
            End Select
    Next K
    T = Sqr((2 * A * D * Cm))
    S = S + T
Next j

Lambda = ((1 / (2 * C * C)) * (S * S)) - (i / 2)


For j = 1 To n
    .Row = j
    For K = 1 To 4
        .Col = K
          Select Case K
            Case 1: A = .Text
            Case 2: D = .Text
            Case 3: Cm = .Text
            Case 4:
```

```
          Fm = .Text
          Qop = Sqr((2 * A * D) / ((i * Cm) + (2 * Lambda * Cm)))
          cu = Fm * Qop
        End Select
    Next K
    Cs = Cs + cu
Next j
End With

End Sub
```

```
Private Sub Paso1Escribir()
KT = 0
With MSFlexGrid1
For j = 1 To n
    .Row = j
    For K = 1 To 5
       .Col = K
         Select Case K
           Case 1: A = .Text
           Case 2: D = .Text
           Case 3: Cm = .Text
           Case 4:
           Case 5:
           Qop = Sqr((2 * A * D) / (i * Cm))
           .Text = CSng(Qop)
           Kp = (Cm * D) + ((A * D) / Qop) + ((Cm * i) * (Qop / 2))
         End Select
    Next K
    KT = KT + Kp
Next j
Text5.Text = KT
End With

End Sub
```

```
Private Sub Paso1Calcular()
Ks = 0
Cs = 0
With MSFlexGrid1
For j = 1 To n
    .Row = j
    For K = 1 To 4
```

```
        .Col = K
          Select Case K
            Case 1: A = .Text
            Case 2: D = .Text
            Case 3: Cm = .Text
            Case 4:
            Fm = .Text
            Qop = Sqr((2 * A * D) / (i * Cm))
            ku = Cm * Qop
            cu = Fm * Qop
          End Select
    Next K
    Ks = Ks + ku
    Cs = Cs + cu

Next j
End With

End Sub
```

```
Private Sub Paso3Calcular()
Ks = 0
Contador = 0
Sum = 0

LI = 0
LS = 10

While (FNF(LI) * FNF(LS)) >= 0
  LS = LS * 10
  Contador = Contador + 1
  If Contador >= 10 Then
  MsgBox "No existe Lambda", 16
  Exit Sub
  End If
Wend



30:
varAntes = Timer
X = (LI + LS) / 2
40: If FNF(X) = 0 Then GoTo 100
50: If (FNF(LI) * FNF(X)) <= 0 Then GoTo 70
```

```
60: LI = X
65: GoTo 80
70: LS = X
80: If (Abs(LS - LI) / (Abs(LI) + Abs(LS))) < 0.0001 Then GoTo 100
90:
varDespues = Timer
Dif = varDespues - varAntes
Sum = Sum + Dif
If Sum >= 5 Then
Sum = 0
Resp = MsgBox("El análisis tomará algunos segundos más, ¿Desea continuar?", vbYesNo)
  If Resp = 7 Then Exit Sub
End If
GoTo 30
100: Lambda = X
```

```
With MSFlexGrid1
For j = 1 To n
   .Row = j
   For K = 1 To 5
      .Col = K
        Select Case K
          Case 1: A = .Text
          Case 2: D = .Text
          Case 3: Cm = .Text
          Case 4: Fm = .Text
          Case 5:
          Qop = Sqr((2 * A * D) / ((i * Cm) + (2 * Lambda * Fm)))
          ku = Cm * Qop
        End Select
   Next K
   Ks = Ks + ku
Next j
End With

End Sub
```

```
Private Sub Paso3Escribir()
KT = 0
Contador = 0
Sum = 0

LI = 0
LS = 10

While (FNF(LI) * FNF(LS)) >= 0
  LS = LS * 10
  Contador = Contador + 1
  If Contador >= 10 Then
  MsgBox "No existe Lambda", 16
  Exit Sub
  End If
Wend



30:
varAntes = Timer
X = (LI + LS) / 2
40: If FNF(X) = 0 Then GoTo 100
50: If (FNF(LI) * FNF(X)) <= 0 Then GoTo 70
60: LI = X
65: GoTo 80
70: LS = X
80: If (Abs(LS - LI) / (Abs(LI) + Abs(LS))) < 0.0001 Then GoTo 100
90:
varDespues = Timer
Dif = varDespues - varAntes
Sum = Sum + Dif
If Sum >= 5 Then
Sum = 0
Resp = MsgBox("El análisis tomará algunos segundos más, ¿Desea continuar?", vbYesNo)
  If Resp = 7 Then Exit Sub
End If
GoTo 30
100: Lambda = X
```

```
With MSFlexGrid1

Text4.Text = Lambda

For j = 1 To n
   .Row = j
   For K = 1 To 5
      .Col = K
        Select Case K
          Case 1: A = .Text
          Case 2: D = .Text
          Case 3: Cm = .Text
          Case 4: Fm = .Text
          Case 5:
          Qop = Sqr((2 * A * D) / ((i * Cm) + (2 * Lambda * Fm)))
          .Text = CSng(Qop)
          Kp = (Cm * D) + ((A * D) / Qop) + ((Cm * i) * (Qop / 2))
        End Select
   Next K
   KT = KT + Kp
Next j

Text5.Text = KT

End With

End Sub
```

```
Private Sub Command1_Click()

On Error GoTo ManipularError

i = Text1.Text
CP = Text2.Text
n = Text3.Text
CP´ = Text6.Text

If IsNumeric(i) And IsNumeric(CP) And IsNumeric(n) And IsNumeric(CP´) And (i > 0)
And (i <= 100) And (CP >= 0) And (n >= 1) And (CP´ >= 0) And (i <> "") And (CP <> "")
And (n <> "") And (CP´ <> "") Then
  If (n / CInt(n)) = 1 Then
```

```
  With MSFlexGrid1
   For j = 1 To n
      .Row = j
       For K = 1 To 4
         .Col = K
          If .Text = "" Then
          MsgBox "Alguna o algunas de las celdas de la tabla estan vacías", 16
          Exit Sub
          Else

          End If
       Next K
   Next j
  End With

F = CP
C = CP´

i = i / 100

Call Paso1Calcular

   If (Ks <= C) And (Cs <= F) Then
   MsgBox "Ninguna Restricción es Activa"
   Call Paso1Escribir
   Text4.Text = ""
   Text7.Text = ""
   Else
   Call Paso2Calcular
    If (Cs <= F) Then
    MsgBox "Solo una Restricción es Activa"
    Paso2Escribir
    Text7.Text = ""
    Else
    Call Paso3Calcular
     If (Ks <= C) Then
     MsgBox "Solo una Restricción es Activa"
     Paso3Escribir
     Text7.Text = ""
     Else
     MsgBox "Ambas Restricciones son Activas"
     Paso4Calcular
     End If
    End If
   End If
```

```
   Else
   MsgBox "Dato inválido ¡Revisar el frame de Datos!", 16
   End If


Else
MsgBox "Dato inválido ¡Revisar el frame de Datos!", 16
End If

ManipularError:
   If Err.Number = 0 Then
   Exit Sub
   Else
   MsgBox "Dato inválido:" & Err.Description, 16
   Err.Clear
   Exit Sub
   End If

End Sub
```

```
Private Sub Command2_Click()
Unload Form3
End Sub
```

```
Private Sub Form_Activate()
MDIForm1.MnuGuardarComo.Enabled = True
MDIForm1.Toolbar1.Buttons(2).Enabled = True

MDIForm1.MnuImprimir.Enabled = True
MDIForm1.Toolbar1.Buttons(3).Enabled = True

Module1.Bandera = 3

End Sub
```

```
Private Sub Form_Deactivate()
MDIForm1.MnuGuardarComo.Enabled = False
MDIForm1.Toolbar1.Buttons(2).Enabled = False

MDIForm1.MnuImprimir.Enabled = False
MDIForm1.Toolbar1.Buttons(3).Enabled = False
```

```
Module1.Bandera = 0

End Sub
```

```
Private Sub Form_Load()
n = Text3.Text

MSFlexGrid1.Rows = n + 1

With MSFlexGrid1

  .Row = 0
  For i = 0 To 5
    .Col = i
    Select Case i
      Case 0: .Text = "Artículo"
      Case 1: .Text = "A($)"
      Case 2: .Text = "D(Uni/año)"
      Case 3: .Text = "c($)"
      Case 4: .Text = "f(U de Esp)"
      Case 5: .Text = "Q*(Uni)"
    End Select
  Next i

  Number = 0
  .Col = 0
  For i = 1 To n
    .Row = i
    Number = Number + 1
      .Text = Number
  Next i

.Row = 1
.Col = 1

End With


End Sub
```

```
Private Sub Form_Unload(Cancel As Integer)
```

```
MDIForm1.MnuGuardarComo.Enabled = False
MDIForm1.Toolbar1.Buttons(2).Enabled = False


MDIForm1.MnuImprimir.Enabled = False
MDIForm1.Toolbar1.Buttons(3).Enabled = False


Module1.Bandera = 0

MDIForm1.MnuCopiar.Enabled = False
MDIForm1.MnuBorrar.Enabled = False
MDIForm1.MnuPegar.Enabled = False

End Sub
```

```
Private Sub MSFlexGrid1_GotFocus()
MDIForm1.MnuCopiar.Enabled = True
MDIForm1.MnuBorrar.Enabled = True
MDIForm1.MnuPegar.Enabled = True
End Sub
```

```
Private Sub MSFlexGrid1_KeyPress(KeyAscii As Integer)

If KeyAscii >= 46 And KeyAscii <= 57 Then
MSFlexGrid1.Text = MSFlexGrid1.Text & Chr(KeyAscii)
End If

End Sub
```

```
Private Sub MSFlexGrid1_KeyUp(KeyCode As Integer, Shift As Integer)

Select Case KeyCode
  Case vbKeyDelete
    MSFlexGrid1.Text = ""
  Case vbKeyBack
    If Len(MSFlexGrid1.Text) > 0 Then
       MSFlexGrid1.Text = Left(MSFlexGrid1.Text, Len(MSFlexGrid1.Text) - 1)
    End If
End Select
```

```
End Sub
```

```
Private Sub MSFlexGrid1_LostFocus()
MDIForm1.MnuCopiar.Enabled = False
MDIForm1.MnuBorrar.Enabled = False
MDIForm1.MnuPegar.Enabled = False
End Sub
```

```
Private Sub Text3_Change()

On Error GoTo ManipularError

n = Text3.Text

If n <> "" And n > 0 And IsNumeric(n) And (n <= 10000) Then
   If (n / CInt(n)) = 1 Then
   MSFlexGrid1.Rows = n + 1
   Else
   MSFlexGrid1.Rows = 1
   End If
Else
MSFlexGrid1.Rows = 1
End If

With MSFlexGrid1

  .Row = 0
  For i = 0 To 5
    .Col = i
    Select Case i
      Case 0: .Text = "Artículo"
      Case 1: .Text = "A($)"
      Case 2: .Text = "D(Uni/año)"
      Case 3: .Text = "c($)"
      Case 4: .Text = "f(U de Esp)"
      Case 5: .Text = "Q*(Uni)"
    End Select
  Next i



If n <> "" And n > 0 And IsNumeric(n) And (n <= 10000) Then
  If (n / CInt(n)) = 1 Then
```

```
  Number = 0
  .Col = 0
  For i = 1 To n
   .Row = i
   Number = Number + 1
     .Text = Number
  Next i
  Else

  End If
Else

End If

End With

ManipularError:
   If Err.Number = 0 Then
   Exit Sub
   Else
   MsgBox "Dato inválido:" & Err.Description, 16
   Err.Clear
   Exit Sub
   End If

End Sub
```

```
Public Function FNF1(Lambda1 As Variant, Lambda2 As Variant)
S = 0
With MSFlexGrid1
For j = 1 To n
   .Row = j
   For K = 1 To 4
      .Col = K
        Select Case K
          Case 1: A = .Text
          Case 2: D = .Text
          Case 3: Cm = .Text
          Case 4: Fm = .Text
          End Select
   Next K
   T = (Sqr((2 * A * D) / ((i * Cm) + ((2) * ((Lambda1 * Cm) + (Lambda2 * Fm)))))) *
(Cm)
   S = S + T
Next j
```

```
FNF1 = S - C

End With

End Function
```

```
Public Function FNF2(Lambda1 As Variant, Lambda2 As Variant)
S = 0
With MSFlexGrid1
For j = 1 To n
   .Row = j
   For K = 1 To 4
      .Col = K
        Select Case K
          Case 1: A = .Text
          Case 2: D = .Text
          Case 3: Cm = .Text
          Case 4: Fm = .Text
          End Select
   Next K
   T = (Sqr((2 * A * D) / ((i * Cm) + ((2) * ((Lambda1 * Cm) + (Lambda2 * Fm)))))) *
(Fm)
   S = S + T
Next j

FNF2 = S - F

End With

End Function
```

```
Private Sub Paso4Calcular()
KT = 0
Sum = 0

L1 = 0
L2 = 0


10:
varAntes = Timer
```

```
L1N = L1 + (((FNF2(L1, L2) * FNF1L2(L1, L2)) - (FNF1(L1, L2) * FNF2L2(L1, L2))) /
((FNF1L1(L1, L2) * FNF2L2(L1, L2)) - (FNF2L1(L1, L2) * FNF1L2(L1, L2))))
L2N = L2 + (((FNF2L1(L1, L2) * FNF1(L1, L2)) - (FNF1L1(L1, L2) * FNF2(L1, L2))) /
((FNF1L1(L1, L2) * FNF2L2(L1, L2)) - (FNF2L1(L1, L2) * FNF1L2(L1, L2))))
varDespues = Timer
Dif = varDespues - varAntes
Sum = Sum + Dif
If Sum >= 5 Then
Sum = 0
Resp = MsgBox("El análisis tomará algunos segundos más, ¿Desea continuar?", vbYesNo)
  If Resp = 7 Then Exit Sub
End If
If (L1N = L1) And (L2N = L2) Then GoTo 20
L1 = L1N
L2 = L2N
GoTo 10
20:
Lambda1 = L1N
Lambda2 = L2N
```

```
With MSFlexGrid1

Text4.Text = CCur(Lambda1)
Text7.Text = CCur(Lambda2)

For j = 1 To n
   .Row = j
   For K = 1 To 5
      .Col = K
        Select Case K
          Case 1: A = .Text
          Case 2: D = .Text
          Case 3: Cm = .Text
          Case 4: Fm = .Text
          Case 5:
          Qop = Sqr((2 * A * D) / ((i * Cm) + (2 * Lambda1 * Cm) + (2 * Lambda2 * Fm)))
          .Text = CSng(Qop)
          Kp = (Cm * D) + ((A * D) / Qop) + ((Cm * i) * (Qop / 2))
        End Select
   Next K
   KT = KT + Kp
```

```
Next j

Text5.Text = KT

End With

End Sub
```

```
Public Function FNF1L1(Lambda1 As Variant, Lambda2 As Variant)
S = 0
With MSFlexGrid1
For j = 1 To n
   .Row = j
   For K = 1 To 4
      .Col = K
        Select Case K
          Case 1: A = .Text
          Case 2: D = .Text
          Case 3: Cm = .Text
          Case 4: Fm = .Text
          End Select
   Next K
   T = (Cm ^ 2) * ((2 * A * D) ^ (1 / 2)) * (((i * Cm) + ((2) * ((Lambda1 * Cm) +
(Lambda2 * Fm)))) ^ (-3 / 2))
   S = S + T
Next j

FNF1L1 = -S

End With

End Function
```

```
Public Function FNF1L2(Lambda1 As Variant, Lambda2 As Variant)
S = 0
With MSFlexGrid1
For j = 1 To n
   .Row = j
   For K = 1 To 4
      .Col = K
        Select Case K
          Case 1: A = .Text
          Case 2: D = .Text
```

```
            Case 3: Cm = .Text
            Case 4: Fm = .Text
            End Select
    Next K
    T = (Cm * Fm) * ((2 * A * D) ^ (1 / 2)) * (((i * Cm) + ((2) * ((Lambda1 * Cm) +
(Lambda2 * Fm)))) ^ (-3 / 2))
    S = S + T
Next j

FNF1L2 = -S

End With

End Function
```

```
Public Function FNF2L1(Lambda1 As Variant, Lambda2 As Variant)
S = 0
With MSFlexGrid1
For j = 1 To n
    .Row = j
    For K = 1 To 4
        .Col = K
          Select Case K
            Case 1: A = .Text
            Case 2: D = .Text
            Case 3: Cm = .Text
            Case 4: Fm = .Text
            End Select
    Next K
    T = (Cm * Fm) * ((2 * A * D) ^ (1 / 2)) * (((i * Cm) + ((2) * ((Lambda1 * Cm) +
(Lambda2 * Fm)))) ^ (-3 / 2))
    S = S + T
Next j

FNF2L1 = -S

End With

End Function
```

```
Public Function FNF2L2(Lambda1 As Variant, Lambda2 As Variant)
S = 0
With MSFlexGrid1
```

```
For j = 1 To n
   .Row = j
   For K = 1 To 4
      .Col = K
        Select Case K
          Case 1: A = .Text
          Case 2: D = .Text
          Case 3: Cm = .Text
          Case 4: Fm = .Text
          End Select
   Next K
   T = (Fm ^ 2) * ((2 * A * D) ^ (1 / 2)) * (((i * Cm) + ((2) * ((Lambda1 * Cm) +
(Lambda2 * Fm)))) ^ (-3 / 2))
   S = S + T
Next j

FNF2L2 = -S

End With

End Function
```

5 CÓDIGO COMPLETO ASOCIADO AL FORMULARIO FRMABOUT

```
Option Explicit

Const READ_CONTROL = &H20000
Const KEY_QUERY_VALUE = &H1
Const KEY_SET_VALUE = &H2
Const KEY_CREATE_SUB_KEY = &H4
Const KEY_ENUMERATE_SUB_KEYS = &H8
Const KEY_NOTIFY = &H10
Const KEY_CREATE_LINK = &H20
Const KEY_ALL_ACCESS = KEY_QUERY_VALUE + KEY_SET_VALUE + _
                KEY_CREATE_SUB_KEY + KEY_ENUMERATE_SUB_KEYS + _
                KEY_NOTIFY + KEY_CREATE_LINK + READ_CONTROL

Const HKEY_LOCAL_MACHINE = &H80000002
Const ERROR_SUCCESS = 0
Const REG_SZ = 1
Const REG_DWORD = 4

Const gREGKEYSYSINFOLOC = "SOFTWARE\Microsoft\Shared Tools Location"
Const gREGVALSYSINFOLOC = "MSINFO"
Const gREGKEYSYSINFO = "SOFTWARE\Microsoft\Shared Tools\MSINFO"
Const gREGVALSYSINFO = "PATH"
```

```
Private Declare Function RegOpenKeyEx Lib "advapi32" Alias "RegOpenKeyExA" (ByVal hKey As Long, ByVal lpSubKey As String, ByVal ulOptions As Long, ByVal samDesired As Long, ByRef phkResult As Long) As Long
Private Declare Function RegQueryValueEx Lib "advapi32" Alias "RegQueryValueExA" (ByVal hKey As Long, ByVal lpValueName As String, ByVal lpReserved As Long, ByRef lpType As Long, ByVal lpData As String, ByRef lpcbData As Long) As Long
Private Declare Function RegCloseKey Lib "advapi32" (ByVal hKey As Long) As Long
```

```
Private Sub cmdSysInfo_Click()
  Call StartSysInfo
End Sub
```

```
Private Sub cmdOK_Click()
  Unload Me
End Sub
```

```
Private Sub Form_Load()
   Me.Caption = "Acerca de " & App.Title
   lblVersion.Caption = "Version " & App.Major & "." & App.Minor & "." & App.Revision
   lblTitle.Caption = App.Title
   lblDescription.Caption = App.FileDescription
   lblDisclaimer.Caption = App.LegalCopyright
End Sub
```

```
Public Sub StartSysInfo()
   On Error GoTo SysInfoErr

   Dim rc As Long
   Dim SysInfoPath As String

   If GetKeyValue(HKEY_LOCAL_MACHINE, gREGKEYSYSINFO, gREGVALSYSINFO, SysInfoPath) Then
   ElseIf GetKeyValue(HKEY_LOCAL_MACHINE, gREGKEYSYSINFOLOC, gREGVALSYSINFOLOC, SysInfoPath) Then
      If (Dir(SysInfoPath & "\MSINFO32.EXE") <> "") Then
         SysInfoPath = SysInfoPath & "\MSINFO32.EXE"
```

```
      Else
         GoTo SysInfoErr
      End If
   Else
      GoTo SysInfoErr
   End If

   Call Shell(SysInfoPath, vbNormalFocus)

   Exit Sub
SysInfoErr:
   MsgBox "System Information Is Unavailable At This Time", vbOKOnly
End Sub
```

```
Public Function GetKeyValue(KeyRoot As Long, KeyName As String, SubKeyRef As
String, ByRef KeyVal As String) As Boolean
   Dim i As Long
   Dim rc As Long
   Dim hKey As Long
   Dim hDepth As Long
   Dim KeyValType As Long
   Dim tmpVal As String
   Dim KeyValSize As Long
   rc = RegOpenKeyEx(KeyRoot, KeyName, 0, KEY_ALL_ACCESS, hKey)

   If (rc <> ERROR_SUCCESS) Then GoTo GetKeyError

   tmpVal = String$(1024, 0)
   KeyValSize = 1024

   rc = RegQueryValueEx(hKey, SubKeyRef, 0, _
                  KeyValType, tmpVal, KeyValSize)

   If (rc <> ERROR_SUCCESS) Then GoTo GetKeyError

   If (Asc(Mid(tmpVal, KeyValSize, 1)) = 0) Then
      tmpVal = Left(tmpVal, KeyValSize - 1)
   Else
      tmpVal = Left(tmpVal, KeyValSize)
   End If
   Select Case KeyValType
   Case REG_SZ
      KeyVal = tmpVal
   Case REG_DWORD
      For i = Len(tmpVal) To 1 Step -1
```

```
            KeyVal = KeyVal + Hex(Asc(Mid(tmpVal, i, 1)))
        Next
        KeyVal = Format$("&h" + KeyVal)
    End Select

    GetKeyValue = True
    rc = RegCloseKey(hKey)
    Exit Function

GetKeyError:
    KeyVal = ""
    GetKeyValue = False
    rc = RegCloseKey(hKey)
End Function
```

6 CÓDIGO COMPLETO ASOCIADO AL MÓDULO ESTÁNDAR MODULE1

```
Public Bandera
```

Bibliografía

- PLANEACIÓN Y CONTROL DE LA PRODUCCIÓN, Daniel Sipper y Robert L. Bulfin Jr., McGraw-Hill, México, 2005.
- PLANEACIÓN DE LA PRODUCCIÓN Y CONTROL DE INVENTARIOS, Narasimhan, Seetharama/ otros, Segunda edición, Prentice-Hall, México, 1997.
- Apuntes de la clase de Planeación y Control de la Producción.
- JUSTO A TIEMPO Y CALIDAD TOTAL, Gustavo Gutiérrez Garza, Ediciones Castillo, México, 2000.
- SCIENTIFIC INVENTORY CONTROL, C. D. Lewis, London Butterworths, Hungría.
- ANALYSIS OF INVENTORY SYSTEMS, Hadley and Whitin, Prentice-Hall, United States of America, 1963.
- MODELOS MATEMÁTICOS DE INVENTARIOS, Gabriel Poveda Ramos, Publicaciones Técnicas, Colombia, 1969.
- EL PROBLEMA DE INVENTARIO: CON MULTIPRODUCTO (TESIS DE MAESTRÍA), Isabel Patricia Aguilar Juárez, Facultad de Ingeniería UNAM, México, 1996.
- EL CÁLCULO CON GEOMETRÍA ANALÍTICA, Louis Leithold, Sexta edición, HARLA, Colombia, 1996.
- CURSO DE PROGRAMACIÓN, Castro, J. Cucker, F. Messeguer, X. Rubio, A. Solano, L. Valles, McGraw-Hill, España, 1993.
- PROGRAMACIÓN METÓDICA, Balcázar J.L., McGraw-Hill, España, 1993.
- LabVIEW USER MANUAL, National Instruments, United States of America, 2001.
- ENCICLOPEDIA DE MICROSOFT VISUAL BASIC 6.0, Francisco Javier Ceballos, Alfaomega, México, 2001.
- APRENDIENDO VISUAL BASIC 6.0, Greg Perry, Prentice-Hall, México, 1999.
- VISUAL BASIC 5.0, Marco A. Tiznado, McGraw-Hill, Colombia, 1998.

- COMO PROGRAMAR EN C/C++, H. M. Deitel y P. J. Deitel, Prentice-Hall, México, 1995.

- INTRODUCCIÓN A LOS SISTEMAS OPERATIVOS, H. M. Deitel, Addison-Wesley Iberoamericana, México, 1993.

- DOS PARA MORTALES, Steven Woas, Addison-Wesley Iberoamericana, Estados Unidos de América , 1993.

- LINUX C/CD, Raúl Montero Rivero, Ediciones Anaya Multimedia, México, 1998.

- DERIVE: APLICACIONES MATEMÁTICAS PARA PC, Agustín Carrillo de Albornoz Torres, Addison-Wesley Iberoamericana, Estados Unidos de América, 1996.

- ANÁLISIS NUMÉRICO, Curtis F. Gerald, Representaciones y Servicios de Ingeniería, México, 1987.

- NUMERICAL RECIPES IN C (THE ART OF SCIENTIFIC COMPUTING), William H. Press, Second Edition, Cambridge University Press, United States of America, 1992.

- ANÁLISIS NUMÉRICO Y VISUALIZACIÓN GRÁFICA CON MATLAB, Shoichiro Nakamura, Prentice-Hall, México, 1999.

- CÓMO ELABORAR Y ASESORAR UNA INVESTIGACIÓN DE TESIS, Carlos Muñoz Razo, Prentice-Hall, México, 1998.

Otras fuentes:

- Biblioteca Virtual de Ingeniería Industrial de la USM.

- Manual en línea de WinQSB.

- Manual en línea de Tora.