UNIVERSIDAD PANAMERICANA

ESCUELA DE INGENIERIA

INCORPORADA A LA U.N.A.M.

DISEÑO E IMPLEMENTACION DE UN SISTEMA INTEGRAL DE CONTROL PARA LA ADMINISTRACION DE UNA EMPRESA MANUFACTURERA

T E S I S P R O F E S I O N A L
QUE PARA OBTENER EL TITULO DE
INGENIERO MECANICO ELECTRICISTA
(AREA MECANICO)
P R E S E N T A:
JUAN PABLO RUIZ VELASCO MARQUEZ

MEXICO, D.F. *1988*

I N D I C E

CAPITULO I **INTRODUCCION**

1.1 Planteamiento del problema 3
1.2 El caso a solucionar 6
1.3 Descripción de los capítulos 7

CAPITULO II **ELECCION DE ALTERNATIVAS**

2.1 Posibilidades para el desarrollo del programa 9
2.2 Herramientas a utilizar 10
2.3 Solución al problema de la compatibilidad 12
2.4 Forma de hacer nuevas implementaciones 14

CAPITULO III **PLANTEAMIENTO DE NECESIDADES Y EL FLUJO DE INFORMACION** 15

CAPITULO IV **DESARROLLO DEL PROGRAMA**

4.1 Diagramas de flujo de programación 27
4.2 Estructura de las bases de datos 41
4.3 Programación 47
4.3.1 Presentación general del programa 47
4.3.2 Definiciones y declaraciones del S.I.A.C. 52
4.3.3 Procedimientos y funciones de uso generalizado 63
4.3.4 Subprogramas utilizados repetidamente ... 101
4.3.5 Subprograma de carga inicial 129
4.3.6 Subprograma de consultas 136
4.3.7 Subprograma de pedidos 157
4.3.8 Subprograma de entradas 177
4.3.9 Subprograma de salidas 204
4.3.10 Subprograma para el control del dinero .. 229
4.3.11 Subprograma de ajustes al inventario 233
4.3.12 Subprograma de resultados y utilerías ... 242

CAPITULO V **PUESTA EN MARCHA Y CONCLUSIONES**

5.1 Puesta en marcha 243
5.2 Conclusiones 254

BIBLIOGRAFIA 259

DISEÑO E IMPLEMENTACION DE UN SISTEMA INTEGRAL DE CONTROL PARA LA ADMINISTRACION DE UNA EMPRESA MANUFACTURERA

CAPITULO I INTRODUCCION

1.1 PLANTEAMIENTO DEL PROBLEMA

Hoy en dia una de las herramientas más importantes con que cuenta el hombre es la computadora. Esta herramienta puede ser utilizada en cualquier disciplina, aumentando la eficiencia y haciendo posibles muchas cosas que antes resultaban muy costosas y hasta imposibles de realizar. La computadora representa una herramienta de mucha utilidad para el ingeniero mecánico electricista ya que le proporciona una forma fácil y rápida de llegar a resultados productivos en sus multiples áreas de interés.

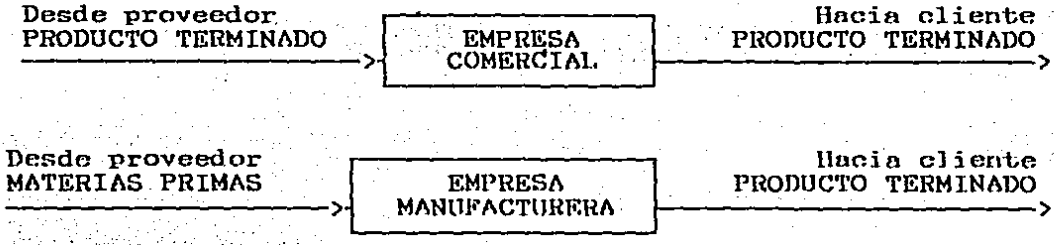
En la carrera de Ingeniería Mecánica Eléctrica se estudian áreas como lo son mecánica, eléctrica, electrónica, matemática, computacional, administrativa, etc., las cuales le sirven como herramientas prácticas para encontrar solución a las necesidades ilimitadas del hombre. De esta manera, el ingeniero cumple con el compromiso que significa el haber recibido una educación superior.

El estudiante de dicha carrera puede profundizar en el área administrativa y computacional para encontrar mejoras en el control administrativo de las empresas.

Esta tesis está dirigida a personas con conocimientos en programación de computadoras (en particular con conocimiento del lenguaje Pascal) y con experiencia en el uso de programas comerciales de aplicación para IBM/PC (principalmente dBase III Plus).

El desarrollo de esta tesis se centra en la solución de la automatización administrativa de las industrias manufactureras, las cuales presentan serios problemas en su control administrativo debido a la falta de programas comerciales (para computadoras personales) que consideren la etapa de producción.

El *software* existente se aplica para comercios, en donde los artículos que se adquieren (productos terminados) son los mismos que se venden (productos terminados), a diferencia de la industria manufacturera, en donde lo que se adquiere (materias primas) no es necesariamente lo que se vende (productos terminados).



Es principalmente esta diferencia la que hace que se presenten dificultades para generar *software* comercial de aplicación general para la empresa manufacturera. Es decir, resulta muy complejo hacer un programa fácil de usar que resuelva en forma satisfactoria las necesidades de todo tipo de empresa manufacturera, ya que existen muchas variantes entre cada una de ellas. Si existiera, por ejemplo, un programa aplicable a:

+ Una fábrica que maneja muchos productos terminados distintos que no cambian normalmente (productos de línea), pocas materias primas y clientes constantes.

+ Una fábrica que hace productos terminados con especificaciones especiales en cada venta, muchas materias primas y clientes variantes.

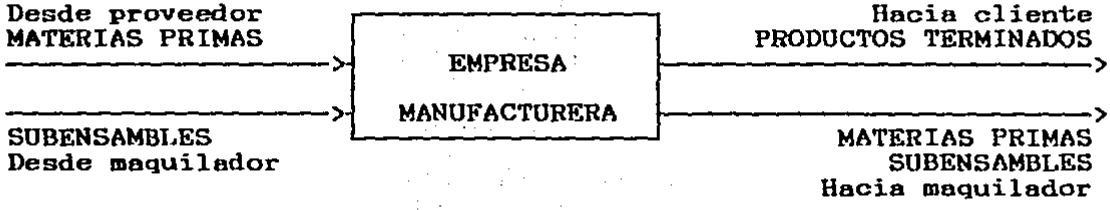
La utilización de un mismo programa en ambas fábricas sería complicada y ocasionaría resultados ineficientes, debido a las grandes diferencias existentes entre ambas. Lo óptimo sería un programa especial para cada una de ellas, que cumpla con los requisitos necesarios en cada caso.

En su movimiento interno las empresas manufactureras producen, a partir de las materias primas que adquieren, una serie de partes que denominaremos subensambles, los cuales no son el producto terminado sino partes que integran a otras partes. Se puede decir que los subensambles estarán formados por materias primas y hasta otros subensambles, y el producto terminado se formará por materias primas y subensambles. Podría parecer difícil el diferenciar un subensamble de un producto terminado, ya que ambos están formados del mismo tipo de partes. Sin embargo, será fácil distinguir los productos terminados de los subensambles porque los primeros son los que se venden al

cliente, mientras que los subensambles no tienen como finalidad su venta.

La definición de los subensambles es de vital importancia debido a que se hace más clara la identificación de las partes integrantes de un producto terminado.

Es necesario tener un control de las entradas y salidas de estas empresas. Aun cuando generalmente las materias primas representan a las entradas, y los productos terminados a las salidas, también existe la posibilidad de salida de materias primas o subensambles hacia otras empresas llamadas maquiladoras y, consiguientemente, la entrada de nuevos subensambles. Por lo tanto, el esquema anterior se ve de la siguiente manera:



Este esquema es aplicable a toda clase de empresa, por ejemplo:

+ A un comercio, en que cada materia prima forma un producto terminado y no existen subensambles ni maquilas. El problema es que la aplicación del programa exigiría la existencia de producciones ficticias que convirtieran a cada materia prima en un producto terminado para su venta posterior.

+ A una maquiladora, empresa que trabaja en forma idéntica al esquema mostrado. Las materias primas son los artículos que le envía su cliente y aquéllas que compra para su propio funcionamiento. Los productos terminados son los artículos que esta empresa regresa ya procesados. Por último, los subensambles representan procesos intermedios de la maquila que se hace. Además, queda la posibilidad de que el maquilador envíe, a su vez, materiales a otro maquilador.

Tomando en cuenta estos aspectos y algunos otros menos relevantes, se puede tener un control muy eficiente de los inventarios. A partir de éste, es muy sencillo ampliar el programa para controlar otros aspectos como son los pedidos que hacen los clientes, y en base a éstos generar las órdenes de compra, maquila y producción. Al tener registrados los pedidos y la descripción de los clientes con sus características

necesarias, se puede llevar un control de cuentas por cobrar y un estado de cuenta de los mismos. A partir de las órdenes de compra y órdenes de maquila se puede llevar un estado de cuenta de adeudos a proveedores y maquilas.

Como se puede ver, el sistema crece y se hace más sofisticado, dando un control cada vez más completo de la empresa.

Hay muchas cosas que se pueden implementar a un sistema de este tipo, como lo son: estadísticas de compras y de ventas, proyecciones a futuro, envío de cartas a proveedores y distribuidores, gráficas de ventas por producto, etc.

También se podrían contestar preguntas tales como: ¿Quién es el distribuidor que más compró en determinado período? ¿Cuál es el artículo de menor venta? ¿Cuál es el mínimo y el máximo en inventarios más recomendable para cada artículo?....

Las respuestas a estas preguntas son de mucha utilidad para una administración eficiente de la empresa.

La existencia de las computadoras hace posible el control administrativo de dichas empresas en una forma muy eficiente y relativamente barata. En esta tesis, se analiza la aplicación de la computadora personal al caso concreto de una empresa manufacturera.

La solución que en esta tesis se encuentra es tan sólo una de las múltiples formas posibles de resolver el problema. Se busca principalmente que el programa a proponer sea versátil, que verifique en lo posible los datos introducidos por el usuario, que no haya que introducir dos veces la misma información y que presente una forma sencilla y potente de obtener reportes diversos.

1.2 EL CASO A SOLUCIONAR

La fábrica en la que se aplicará el programa para resolver las principales necesidades de control administrativo (lo cual no significa que sea la única aplicación para este programa) se dedica a hacer luminarios de varios tipos y estilos; tiene cerca de 500 productos terminados de línea (no se hacen productos especiales), y más de 400 materias primas. Trabaja con maquiladoras a las cuales envía materias primas y subensambles y sus clientes son distribuidores autorizados por la fábrica. De los productos que maneja, hay algunos que se fabrican sobre pedido y otros que se tienen en inventarios.

Esta fábrica es principalmente una ensambladora, ya que la mayoría de los procesos que requieren sus productos se realizan

fuera de ella. Entre estos procesos se encuentran: rechazado, troquelado, inyección de plástico, extruido (aluminio y plástico), fundición, doblado de lámina, acabado de las piezas (cromado, latonado y pintado), etc. El proceso que se realiza principalmente en la empresa es el armado de las partes, que se hace con punteadora y tornillería diversa.

Para los efectos del programa diseñado, carece de importancia el tipo de procesos que se realicen en la empresa y fuera de ella, así como la proporción de procesos en maquila respecto a los procesos en producción.

Los aspectos que se controlan con el programa son:

+ Inventarios (materias primas, subensambles y productos terminados)
+ Materiales en proceso
+ Pedidos
+ Facturación
+ Estado de cuenta de clientes
+ Estado de cuenta de maquiladores y proveedores

También se deja abierta la posiblidad de implementar nuevas aplicaciones, ya sea dentro del programa o fuera de él (como es el caso de una base de datos, una hoja de cálculo, un procesador de palabras, etc.).

1.3 DESCRIPCION DE LOS CAPITULOS

En el Capítulo II "ELECCION DE ALTERNATIVAS" se ve con qué herramientas se va a resolver el problema. Se exponen varias soluciones alternativas y se elige la solución más adecuada en función de las necesidades más importantes de la empresa.

El Capítulo III "PLANTEAMIENTO DE NECESIDADES Y EL FLUJO DE INFORMACION" plantea las necesidades más importantes a resolver y su repercusión en el flujo de información. Expone todos los puntos a controlar y cómo se afectan entre sí.

En el Capítulo IV "DESARROLLO DE DIAGRAMAS DE FLUJO Y PROGRAMACION" se presentan las estructuras de las bases de datos que se utilizarán, se obtienen los diagramas de flujo y los listados de los programas. El resultado de este capítulo es el sistema integrado para la administración por computadora (SIAC) para la empresa, RIMA, S.A.. Dicho sistema puede ser fácilmente adaptado a otras empresas por una persona con experiencia en programación.

En el último Capítulo "PUESTA EN MARCHA Y EJEMPLIFICACION" se muestran las etapas necesarias para instalar el programa así como ejemplos de su utilización. También se presentan ejemplos de las ventajas de su compatibilidad con el programa dBase III Plus y de la transferencia de datos desde éste hacia otros programas para obtener gráficos y resultados diversos.

CAPITULO II· ELECCION DE ALTERNATIVAS

Al no existir un programa de aplicación general para empresas manufactureras, en esta tesis se ha diseñado uno de aplicación concreta, dejando abierta la posibilidad de adaptarlo a diferentes tipos de empresas.

2.1 POSIBILIDADES PARA EL DESARROLLO DEL PROGRAMA

Es necesario saber a partir de qué se desarrollará el sistema. A continuación se enuncian las posibilidades que se consideraron:

a) Modificar un programa ya existente, enfocado a comercios, para que se adecúe a las necesidades de una empresa manufacturera. Esta posibilidad se considera poco factible porque estos programas al venderse en código ejecutable (ya compilados), son prácticamente imposibles de modificar. Además sería muy difícil adecuar todos los archivos y formatos de pantalla a tan distinto propósito. Por lo anterior se descarta esta posibilidad.

b) Desarrollar un programa que se enlace a los archivos de un programa enfocado a comercios; esta posibilidad es factible y relativamente fácil de realizar. Se tendría por resultado un programa con suficiente versatilidad (tanta como la que el programa adquirido posea), pero con problemas en cuanto a su utilización. Si se elige esta posibilidad, el resultado sería un programa con múltiples "parches" y con alta posibilidad de error.

c) Hacer por completo el programa, logrando como resultado un programa a la medida de la empresa en cuestión, con todos los requisitos deseados y con la mayor eficiencia posible. En contrapartida, se puede caer en el error de generar un programa tan concreto que sea poco versátil, y que, al requerir cambios, se tendría que hacer un trabajo difícil y laborioso.

De las posibilidades antes expuestas se eligió la última, pero se buscará, en la medida de lo posible, cubrir con el requerimiento de versatilidad, para así lograr la mejor opción posible.

--> De esta sección se concluye que la opción más adecuada será desarrollar completamente el programa.

2.2 HERRAMIENTAS A UTILIZAR

El programa se puede realizar por medio de un lenguaje de alto nivel como el BASIC, Pascal, COBOL, FORTRAN, etc. o por medio de un programa de aplicación general.

A) Lenguajes de alto nivel:

Los lenguajes de alto nivel proporcionan las herramientas necesarias para generar un programa que cumpla con todos los requisitos necesarios; su único problema sería la dificultad que presenta el realizar un sistema tan grande, y especialmente uno muy versátil.

De los lenguajes de alto nivel disponibles para el IBM/PC, destacan el BASIC y el Pascal (también el lenguaje C que es muy potente pero de menor nivel que los anteriores; por lo que es más difícil de utilizar) como los más populares y potentes por sus adaptaciones a estas microcomputadoras.

a) El lenguaje BASIC es más fácil de aprender a usar y de aplicar en pequeños proyectos, pero en grandes proyectos como éste, se hace enredoso y confuso.

b) El lenguaje Pascal, a diferencia del BASIC, resulta más difícil de aprender y de aplicar en pequeños proyectos, pero en grandes proyectos es mucho más estructurado y claro.

Por lo tanto, el lenguaje más adecuado para este proyecto es el Pascal.

B) Programas de aplicación general:

Existen en el mercado programas de aplicación general como las hojas electrónicas, las bases de datos y los procesadores de palabras. Algunos de éstos son programables, es decir, tienen su propio lenguaje por medio del cual es posible automatizar sus operaciones.

El programa que se necesita puede realizarse utilizando una hoja electrónica o una base de datos potentes.

a) La hoja electrónica tiene una capacidad muy alta de realizar cálculos matemáticos complejos y de generar gráficas en general, pero muy baja en cuanto al manejo y manipulación de archivos. El aplicarla sería poco adecuado, si tomamos en cuenta que la parte más importante del proceso requerido es el manejo de los archivos y cálculos matemáticos sencillos, por lo que aplicarla sería desperdiciar capacidad en unas cosas y carecer de ésta en otras cosas muy necesarias.

b) Una base de datos tiene una alta capacidad en el manejo de archivos, una no muy elevada en cálculos matemáticos y excluye la posibilidad de gráficos. Tomando en cuenta que el programa que se necesita requiere principalmente del manejo de archivos y cálculos matemáticos sencillos, resulta muy adecuada la aplicación de una base de datos.

En base a los dos incisos anteriores, se justifica la utilización de una base de datos.

Sería bueno destacar que es posible hacer cálculos matemáticos complejos y gráficas mediante la transferencia de información desde la base de datos a la hoja electrónica.

Con base a los análisis anteriores, sólo nos queda elegir entre un lenguaje de programación como el Pascal (A) y una base de datos (B) para el desarrollo del programa.

Con el Pascal se puede hacer prácticamente todo lo que se quiera, pero con la desventaja de que sería muy complejo pretender obtener todos los resultados deseados por medio de él. En contrapartida, una base de datos programable puede resolver el problema en forma mucho más sencilla que el lenguaje Pascal. A primera vista, parecería lo más conveniente el utilizar una base de datos programable, pero si nos adentramos más al problema y observamos que el programa requiere repetidas revisiones de los archivos (por ejemplo, el verificar que las claves de los artículos introducidas por el usuario sean correctas), advertiríamos que lo más conveniente es mantener estos archivos en memoria RAM para una consulta más rápida y para evitar un desgaste excesivo de los discos de archivo, lo cual no es posible de lograr en una base de datos.

Se puede notar que, el elegir entre un lenguaje de programación como Pascal y una base de datos programable para desarrollar el proyecto, no es una tarea fácil, ya que existen múltiples factores como los ya mencionados y otros similares que nos dan pros y contras al elegir cualquiera de ellos.

Lo que se puede hacer es aprovechar al máximo las bondades de cada una de estas dos opciones, es decir, se puede diseñar el sistema de manera que todo aquéllo que la base de datos programable no sea capaz de cumplir con eficiencia, lo realice un programa en Pascal.

Como la sección del programa que requiere mantener en memoria los datos es la de uso más común, se le denominará 'Programa Principal' (el cual se hará en Pascal) y a las otras secciones que estén fuera de él, se les denominará simplemente

'Programa' seguido del nombre asignado. Los programas realizados en lenguaje de base de datos se denominarán como 'LBD'.

Las ventajas de combinar las dos opciones son:

+ Se puede desarrollar cualquier programa deseado.

+ Existe una capacidad muy elevada para el manejo y manipulación de la información.

+ Se tiene una gran flexibilidad en el programa.

+ Resultará sencillo hacer nuevas implementaciones y ajustes al programa de base de datos.

+ Si la base de datos programable que se va a utilizar es muy popular, se presentarán nuevas posibilidades para la ampliación del sistema y una fácil y tal vez directa transferencia de información hacia otros programas, etc.

Las desventajas de combinar las dos opciones son:

+ Se tiene que salir del programa principal para utilizar los otros programas.

+ Hacer cambios en el programa principal no resulta muy sencillo.

+ Y algunas otras desventajas que se verán después.

Al ver las ventajas y desventajas resulta finalmente ser una solución factible, considerando que las desventajas que presenta serán minimizadas en la medida de lo posible. Por ejemplo, si tomamos la primera de ellas que dice que se tiene que salir del programa principal para ejecutar algún otro programa, lo que se puede hacer es que el programa principal cumpla con todos los requisitos para uso diario, y que los otros programas sean de uso esporádico.

—> En esta sección, se determinó que se desarrollará el Programa Principal en Pascal por su alta eficiencia y los otros programas podrán ser en Pascal o en LBD dependiendo de cada aplicación en particular.

2.3 SOLUCION AL PROBLEMA DE LA COMPATIBILIDAD

Para que todos los programas compartan la misma información, existen dos posibilidades:

a) Hacer un programa de transferencia de archivos desde un formato a otro; la ventaja es que de esta forma cada

programa manejará los archivos en su forma estándar, con lo que se tiene acceso a los mismos en la forma más rápida posible. Sin embargo tiene el inconveniente de que es tardado hacer la transferencia del archivo, que se tiene que hacer por lo menos cada vez que haya un cambio en la base de datos y se quiera transferir el control a otro programa.

b) Que todos los programas manejen la información en el mismo formato; este método es muy apropiado para tener versatilidad y compatibilidad general de los archivos, no requiere ninguna transferencia y todos los archivos guardados en disco contienen información actualizada. La desventaja es que el acceso al disco de aquellos programas que se tienen que acoplar al formato elegido será más lento de lo normal.

De las posibilidades anteriores, se elige la segunda, por sus ventajas y por la factibilidad de minimizar las desventajas (se explicará más adelante). Sólo se tomará la primera para aplicaciones especiales como la transferencia de datos a una hoja electrónica.

Como la base de datos guarda sus archivos en un solo formato, el cual es diferente a los archivos estándar del Pascal (debido a la forma muy particular en que el Pascal almacena los datos en disco) y éste último sí puede hacer el acceso a los archivos estándar de cualquier programa, tendrá que ser entonces el programa hecho en Pascal el que no trabaje los archivos en su forma estándar.

Es posible minimizar la desventaja de tener que operar archivos no estándar desde Pascal disminuyendo el número de accesos al disco. Esto se puede lograr haciendo una sola lectura de los datos de uso más común y manteniéndolos en memoria (en la medida de lo posible), y además guardando identificadores de posición de registros del disco para, en caso de un acceso al disco, hacerlo en forma directa (o sea, no tener que buscar registro por registro). Así se lograrán accesos más rápidos al disco que si se hicieran desde un programa en LBD. El costo de esto es un arranque más lento del programa y algunas limitaciones en la capacidad de almacenamiento.

—> En esta sección se determinó que los programas en Pascal deberán de manejar los archivos de la misma manera que la base de datos programable elegida para que haya compatibilidad en los archivos.

2.4 FORMA DE HACER NUEVAS IMPLEMENTACIONES

Para poder hacer nuevas implementaciones al mismo, se proponen las siguientes opciones:

a) Es posible modificar el Programa Principal para que obtenga los nuevos resultados buscados cada vez que surjan nuevas necesidades.

b) En muchos casos se pueden obtener los resultados por medio de otro programa.

De entre estas opciones es difícil elegir alguna como regla general, ya que la elección depende principalmente de las características de cada caso.

Por ejemplo:

+ Si se quisiera hacer un reporte que muestre todos los pedidos pendientes de pago y que ya estén entregados sería más apropiado utilizar LBD, por no representar este reporte un programa de uso diario y porque es más difícil obtener este programa en Pascal que en LBD.

+ Si se quiere hacer una nueva consulta para saber si todavía tiene crédito algún cliente, lo más adecuado sería agregar este programa al Programa Principal porque sería de uso diario.

—> Las nuevas implementaciones al sistema se harán ya sea modificando el programa principal o creando otro externo (en Pascal o LBD), dependiendo de cada caso en particular.

CAPITULO III

PLANTEAMIENTO DE NECESIDADES Y EL FLUJO DE INFORMACION

Para el desarrollo del programa fue necesario definir en forma concreta y exacta todas las funciones que éste debía cumplir. Para lograr lo anterior, se necesitaron conocer las operaciones administrativas que se ejecutaban en la empresa y simplificarlas, en la medida de lo posible, para hacer más sencillo el funcionamiento y creación del programa.

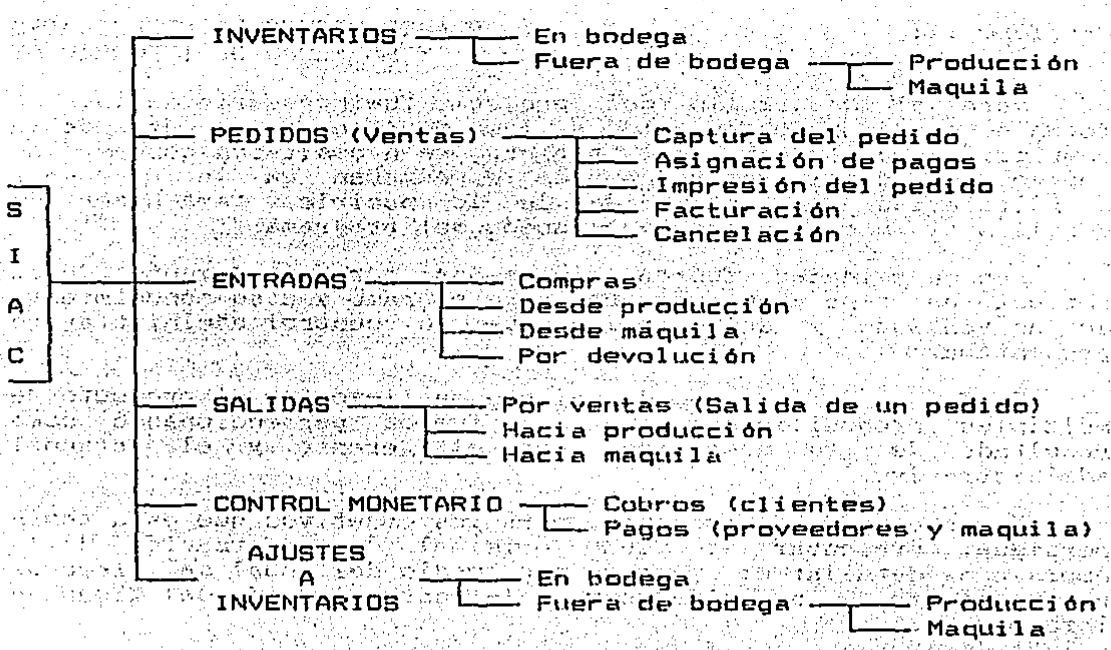
Las necesidades fueron planteadas por el gerente de la empresa, en base a su experiencia en el ramo y a su conocimiento de las ventajas y deficiencias que su control administrativo presentaba.

El flujo de información al que se llegó fue el producto de múltiples proposiciones que se fueron perfeccionando como resultado de su discusión con el gerente y el personal administrativo.

No se consideró necesario, para los objetivos que esta tesis persigue, presentar el flujo original de información de la empresa ni detallar el proceso por medio del cual se llegó al planteamiento de necesidades y a la determinación del flujo de información.

En esta fase del desarrollo del programa se plantean las necesidades más importantes y la forma de resolverlas. Lo que se pretende es enfocar en forma integral el problema con sus respectivas soluciones y enmarcar así los objetivos buscados.

A continuación se establecen los puntos principales que el programa controlará y la explicación de cada uno de ellos. Aún cuando en este capítulo se definen las funciones que debe realizar el programa, su explicación se detalla en el capítulo IV.



INVENTARIOS:

El control de inventarios es uno de los aspectos esenciales del sistema ya que de él se derivan muchos beneficios como lo son:

+ Un buen servicio a los clientes al poderles informar en forma rápida y confiable las cantidades de artículos de que se dispone.

+ La minimización de pérdidas de mercancías por falta de control administrativo.

+ La posibilidad de hacer una planeación de la producción en base a información confiable y actualizada.

Son éstas y muchas otras las razones por las cuales es muy conveniente poner especial atención en el control de los inventarios.

Los inventarios en bodega son la sección de más importancia para el control de inventarios. En la bodega se tienen materiales indispensables para ventas, producción y maquila, por lo que se debe tener un control estricto de ella. Por esta razón, se

considera la bodega el sistema central a controlar por el programa.

En la bodega se almacenarán toda clase de materiales previamente inspeccionados que, de ésta, se enviarán oportunamente a su destino. Para efectos del programa no será necesario hacer una distinción especial entre los diferentes tipos de bodega con que se cuenta.

Los inventarios fuera de la bodega se deben controlar para saber cuáles materiales están en producción y cuáles en maquila. Esta información es de utilidad para evitar pérdidas de materiales, para la planeación de los próximos envios, para la fecha de entrega de pedidos, etc.

Los inventarios se alteran por medio de la sección de pedidos, entradas, salidas y ajustes, es decir, la sección de inventarios no es una sección que haga movimientos, sino que es una sección de consulta.

PEDIDOS:

Por las características de la empresa, es necesario trabajar con pedidos para después facturarlos. Esto se debe a la existencia de artículos que se producen sólamente cuando son requeridos por un cliente. Por medio de los pedidos, entonces, es posible saber qué artículos han solicitado nuestros clientes y, a partir de esto y otros datos, hacer una planeación eficaz de las compras, producción y maquila.

Para controlar los pedidos se requiere de varias secciones que comprenden casi todo su manejo, y que son:

Captura del pedido; es en esta sección donde se inicia un pedido; lo que se hace es introducir todos los datos indispensables para definir al pedido. Estos son:

Folio; es el número de referencia que identifica al pedido. Este número se asigna en forma automática por la computadora.

Fecha de pedido; esta fecha nos indica el día en que fue elaborado el pedido. La fecha será asignada en forma automática por la computadora.

Datos del cliente; éstos son: clave, nombre, dirección, teléfono, ciudad, estado, código postal, destino del pedido (embárquese a), condiciones de pago (días de crédito), descuento (en porcentaje), descuento por pronto pago (en porcentaje). De todos estos datos, lo único que no

se puede variar para un mismo cliente de un pedido a otro, será la clave del cliente.

Sabiendo que los clientes son cautivos (es decir, que no varían regularmente), se llevará un control completo de cada cliente con todos los datos anteriormente mostrados. También es necesario saber el saldo actualizado del cliente y su límite de crédito.

Existe la posibilidad de venta a clientes esporádicos, por lo que se deberá tener la alternativa de operar con clientes eventuales, de los cuales no se llevará un control tan exacto. La venta a clientes eventuales será poco común en la operación normal de la empresa.

Partidas del pedido; en las partidas se registran los artículos requeridos por el cliente. Los conceptos para cada partida son: número de partida, clave del artículo, cantidad, descripción y precio unitario. La descripción del artículo puede ser modificada en algún pedido si se desean hacer aclaraciones adicionales. A partir de la cantidad y el precio unitario se puede calcular el importe de la partida (que es el producto de la cantidad por el precio unitario).

El número de partidas por pedido se fijará en un máximo de diez. La suma de los importes de las partidas constituye el total del pedido. A este total se le aplicará el descuento autorizado para el cliente, lo cual dará por resultado el subtotal del pedido. Por último, a este subtotal se le aplica el I.V.A., obteniendo así el importe del pedido.

El descuento por pronto pago no se aplica al importe del pedido, sino que se considera como un pago realizado por el cliente, que obtiene una nota de crédito expedida por la empresa. Este punto se detallará más adelante.

Los artículos que pueden especificarse en un pedido son: materias primas, subensambles y productos terminados. Sólo en el caso de estos últimos será necesario conocer el número de piezas disponibles de cada artículo, que equivale a la diferencia entre las existencias totales y los pedidos pendientes de entrega de cada artículo. Es por esto que se puede tener una cantidad disponible negativa de cualquier artículo.

El importe de los pedidos será modificado en caso de un incremento en los precios de venta de los artículos, excepto cuando se haya entregado la mercancía o bien se haya recibido un anticipo o el pago total del pedido para asegurar el precio de la misma. Esta modificación del importe del pedido se hará en la sección de asignación de pagos a pedidos.

Fecha esperada de entrega; indica la fecha prometida de entrega del pedido. Esta será especificada por la persona de ventas en forma manual pues, como existen muchas variables que pueden alterar la fecha de entrega, resultaría impráctico introducirlas en el programa.

Fecha de cobro; esta fecha indica el día en que se debe cobrar el pedido, y se determina en forma manual en el momento de la entrega. Para su cálculo, se requiere sumar, a la fecha en que se entregó el pedido, el número de días de crédito del cliente (condiciones de pago).

Asignación de pagos; los pagos de los clientes se asignarán a los pedidos a criterio del operario. No necesariamente se pagarán primero los pedidos iniciales, sino que se tendrá la libertad de seleccionar cualquiera de ellos. Se podrán recibir un máximo de dos pagos para cada pedido.

La asignación de pagos a pedidos no puede hacerse directamente a partir del pago físico recibido. Esto se debe a que los clientes pueden cubrir en un solo pago diversos pedidos e incluso dejar algún remanente o hacer varios pagos para un solo pedido. Lo que se hará es llevar un estado de cuenta de cada cliente en donde se registren los pagos recibidos y su designación a los diversos pedidos. Estos estados de cuenta se cerrarán periódicamente y serán entregados al cliente.

Anticipo; se puede recibir un anticipo con el fin de fijar el precio de la mercancía. El anticipo no quedará restringido a un porcentaje, sino que el mínimo anticipo para cada pedido se establece a criterio del vendedor, tomando en cuenta la política de la empresa y las circunstancias particulares del momento. Cuando se recibe un anticipo, éste se registra y se reduce el saldo del cliente.

Pago total; éste deberá cubrir el importe del pedido. En el caso de que ya haya un anticipo, el pago total será la diferencia necesaria para la cobertura total del pedido. Al igual que en el caso del anticipo, se registra el pago del pedido y se resta el importe del pago del saldo del cliente.

Cuando se asigna el pago total a un pedido que no tiene anticipo y no ha sido entregado, debe revisarse que no haya habido cambio en los precios de la mercancía; en caso contrario, se actualizarán los precios y el importe del pedido antes de que se registre el pago. La operación de actualización de precios podrá ser cancelada por una persona autorizada.

Si el pago se hace en fecha anterior o igual a la de la entrega del pedido, y éste tiene descuento por pronto pago, se hará lo siguiente antes de que se registre el pago:

+ Se expedirá una nota de crédito a favor del cliente por una cantidad igual al descuento por pronto pago.

+ Se abona la nota de crédito al saldo del cliente.

Fecha de entrega; es la fecha en que se entregó la mercancía que indica el pedido; se registra en forma automática desde el momento en que se hace la salida de la mercancía desde la sección de salidas. Por lo tanto este dato no se registra desde la sección de pedidos.

Impresión del pedido; los pedidos se podrán imprimir en cualquier momento y tantas veces como lo desee el usuario.

Facturación; el momento de facturación del pedido quedará a criterio del operario. La única restricción es que no se permitirá al operario la reimpresión de facturas, sólo se dejará abierta esta posibilidad a una persona responsable que verifique las razones por las cuales es necesaria esa reimpresión. La fecha en que se imprimió la factura se registra automáticamente.

Cancelación del pedido; los pedidos se pueden cancelar si no ha sido entregada la mercancía ni emitida la factura y si los artículos que lo forman no son artículos especiales (aquéllos que sólo se fabrican sobre pedido).

Al cancelar un pedido, se imprimirá una nota de crédito a favor del cliente por el importe de los pagos hechos al mismo. Si no es un cliente eventual, la nota de crédito se abonará automáticamente al saldo del cliente y se registrará dicho pago.

Cuando se hace una cancelación se debe de dar la razón por la cual se cancela el pedido. Esta razón será un número codificado por la empresa. Por ejemplo, la razón 5 puede ser 'cancelado por error en la captura del pedido'.

ENTRADAS:

Para lograr un adecuado control de inventarios es indispensable controlar las entradas a la bodega. Se entenderá por entrada todo ingreso de materiales a bodega (materias primas, subensambles o productos terminados). Existen diversas razones por las cuales se reciben mercancías, por lo que se dividirá a la sección de entradas en diferentes subsecciones según el origen de la entrada. Antes de pasar a explicar cada una de estas subsecciones, se definirán los puntos generales que definen una entrada.

Folio; todas las entradas estarán foliadas. El folio servirá para llevar un control más seguro de las entradas y para diferenciar más fácilmente una entrada de otra. La asignación del folio se hará en forma automática por la computadora.

Origen de la entrada; este dato nos indica el tipo de entrada del que se trata, es decir, si es por compra, por maquila, por producción o por devolución de un cliente.

Clave de procedencia; esta clave servirá para identificar la fuente de la mercancía.

Fecha; ésta indicará el día en que fue recibida la mercancía. Esta fecha será asignada automáticamente por la computadora.

Fecha esperada de pago; es la fecha en que se debe pagar la mercancía que se recibe. Esta fecha se especifica por el usuario a partir de las condiciones de pago en cada caso.

Fecha de pago; es la fecha en que se pagó la mercancía que se recibió.

Partidas de la entrada; éstas representan los artículos que ingresan a bodega. Las partidas para cada entrada serán como máximo diez. Los conceptos para cada partida son: número de partida, artículo, cantidad y precio unitario. Como las entradas son para control interno de la empresa, se consideró innecesaria la descripción del artículo.

Las subsecciones de las que hablábamos anteriormente utilizan parcial o totalmente los datos anteriores. Ahora pasaremos a definir cada tipo de entrada:

Compras; éstas son las entradas de materias primas a la empresa, que se generan a partir de una orden de compra. No se deben de relacionar las órdenes de compra con la recepción de mercancías ya que, en general, los proveedores entregan en forma parcial cada orden de compra. El precio unitario que se especifica es el costo real de las materias primas recibidas (ya aplicando descuentos y sin tomar en cuenta el descuento por pronto pago si lo hay). Al recibir la empresa materias primas, se actualizarán los costos de las mismas en función de su precio unitario, se incrementará el inventario de materia prima y se disminuirá la cantidad pedida a proveedores (la cual aumenta con las órdenes de compra).

Las entradas por compras comprenden todos los puntos generales para una entrada.

+ El origen de la entrada será por compras.

+ La clave del que envía la mercancía es la clave del proveedor.

+ Los otros puntos se aplican de acuerdo con su definición.

Desde producción; estas entradas son el resultado de una orden de producción. No se debe de relacionar en forma directa la entrada con la orden de producción. Al registrarse la entrada desde producción, se debe de descontar del inventario de materiales en proceso y aumentar en el inventario en bodega. El precio unitario que se registrará será el costo del material que ingresa a bodega (incluyendo costo de las partes y procesos).

Para las entradas por concepto de producción no se necesitarán todos los datos generales de las entradas.

+ El origen de la entrada será por producción.

+ La clave del que envia la mercancía es la clave que se le asigne a la propia empresa.

+ La fecha esperada de pago y la fecha de pago son irrelevantes por lo que se asignará automáticamente la fecha en que se hace la entrada.

+ Los puntos restantes se aplican según su definición.

Desde maquila; éstas son el producto de una orden de maquila. No se debe de relacionar en forma directa la entrada con la orden de maquila. Al registrarse la entrada desde maquila hay que descontarla del inventario de materiales en maquila y aumentarla en el de materiales en bodega. Los precios que se especificarán en dicha entrada serán los costos de maquila y con éstos se actualizará automáticamente el registro de costos de maquila.

Al igual que en las entradas por compras, el registro de las entradas por maquila utiliza todos los puntos generales de una entrada.

+ El origen de la entrada será por maquila.

+ La clave del que envía la mercancía es la clave del maquilador.

+ Los puntos restantes se aplican según su definición.

Por devolución; se puede presentar el caso de una devolución de cualquier artículo (materia prima, subensamble, producto terminado) por parte de un cliente. Al recibir la mercancía, se registra la entrada a bodega y se hace una nota de crédito (que se abona automáticamente al saldo del cliente) por el importe de la devolución. El precio unitario al que se recibe cada artículo se especificará en forma manual.

Este tipo de entrada utiliza algunos de los puntos generales de una entrada y es como sigue:

+ El origen de la entrada será por devolución.

+ La clave del que envía la mercancía será la clave del cliente.

+ La fecha esperada de pago y la fecha de pago coincidirán con la fecha en la que se hace la devolución.

+ Los puntos restantes se aplican según su definición.

SALIDAS:

Se entenderá por salida a todo egreso de mercancías de la bodega.

Las salidas estarán formadas por los siguientes conceptos:

Folio; todas las salidas estarán foliadas. La asignación del folio se hará en forma automática por la computadora.

Origen de la salida; este dato nos indica el tipo de salida del que se trata, es decir, si es por venta, hacia maquila o hacia producción.

Clave del destinatario; esta clave servirá para identificar el destino de dicha mercancía.

Fecha; ésta indicará el día en que salió la mercancía, y será asignada automáticamente por la computadora.

Partidas de la salida; éstas representan los artículos que salen de bodega. Las partidas para cada salida serán como máximo diez. Los conceptos de las partidas son: número de partida, artículo, cantidad y precio unitario. No es necesaria la descripción del artículo ya que las salidas son para control interno de la empresa.

Estas salidas sólo pueden darse por tres conceptos:

Por ventas; cuando se va a entregar la mercancía a un cliente se hará una orden de salida a partir del pedido en cuestión. Esta orden autoriza la salida de mercancías de la bodega para ser enviadas a ventas.

Todas las partidas que forman el pedido deben de ser entregadas en una sola operación, es decir, no se permiten entregas parciales. En caso de que se haga una entrega parcial, será necesario cancelar el pedido original y hacer un nuevo pedido para los artículos que se van a entregar y otro para los artículos pendientes.

Para poder dar salida a la mercancía, la computadora revisará que el pedido ya esté pagado o

bien que el cliente no haya sobrepasado su límite de crédito.

Hacia producción; para poder producir es necesario que los materiales que componen un artículo se envíen de la bodega de materiales hacia producción. Para lograrlo, se dará la salida de las partes necesarias para ensamblar el producto y una orden de producción en donde se indique el producto a fabricar. Sólo pueden salir por este concepto materias primas y subensambles.

Hacia maquila; en forma análoga, se debe de hacer una salida de las partes que forman el artículo que se quiere enviar a maquilar y, además, una orden de maquila en donde se le pida al maquilador que produzca los artículos deseados. Al igual que en el punto anterior, sólo pueden salir por este concepto materias primas y subensambles.

CONTROL MONETARIO:

El control monetario es la sección en la que se tomará nota de los pagos hechos por la empresa y los cobros a clientes. El control monetario tiene dos secciones, que son:

Cobros; la sección de cobros es esencial en toda empresa y por ello se debe de lograr un sistema eficiente y confiable para el control de esta área. Sólo se recibirán cobros de clientes, ya sea por un pago físico o por una nota de crédito otorgada por la empresa.

Como se indicó en la sección de pagos de pedidos, la recepción de los pagos será asignada a un estado de cuenta y de éste se asignarán los pagos a los pedidos que se desee. El saldo del cliente siempre será acreedor.

Pagos; éstos se harán a proveedores y maquiladores bajo los mismos principios con los que trabaja el control monetario para cobranza. El saldo de la empresa con sus proveedores será siempre acreedor, e irá en decremento a medida que se utilice para cubrir las entradas de mercancías. De lo anterior se deriva la necesidad de especificar, en el momento de registrar una entrada por compra o por maquila, la fecha esperada de pago, el importe a pagar y la fecha real en que se pagó.

AJUSTES A INVENTARIOS:

Los inventarios físicos que existen en la empresa pueden no coincidir con los que indica la computadora, debido a errores humanos como el conteo equivocado de productos que entran o salen, el desperdicio de partes, la pérdida de materiales en maquila, en producción, o en bodega, etc. Por esto es necesario permitir la alteración positiva y negativa de los inventarios guardando un registro de los ajustes hechos al mismo.

Los ajustes pueden aplicarse a los inventarios de bodega o fuera de ella. Los ajustes fuera de bodega pueden ser ajustes en maquila o bien en producción. Se dejará libre el aumento de los inventarios y sólo se permitirá la alteración negativa de los mismos a una persona autorizada.

De esta manera quedan expuestas las necesidades a solucionar por el programa y en el siguiente capítulo se obtiene su solución concreta.

CAPITULO IV DESARROLLO DEL PROGRAMA

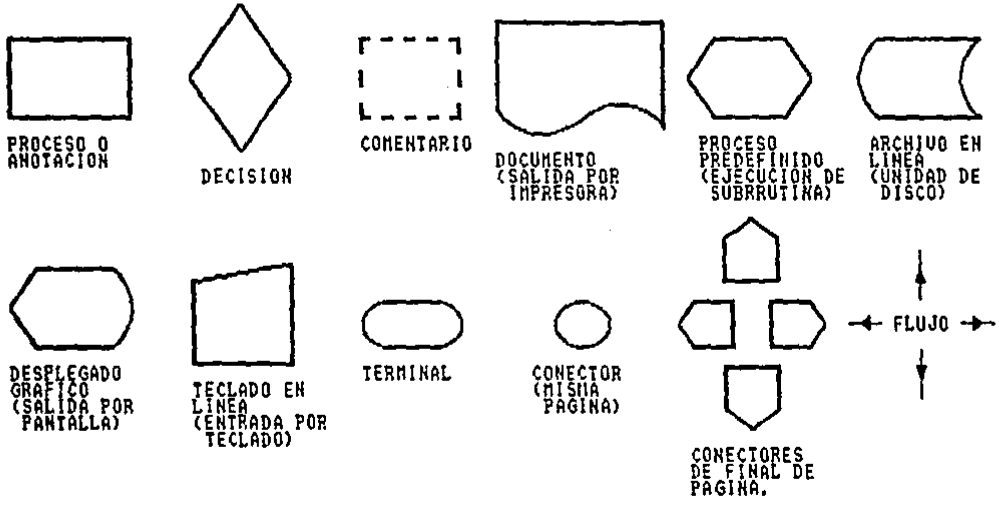
Es en este capitulo donde culmina el análisis anterior y se obtiene el programa completo para su adaptación a la empresa en cuestión.

En primer lugar se mostrará el programa por medio de los diagramas de flujo de programación, en los cuales se esquematiza en forma clara y sencilla el funcionamiento general del programa. La siguiente sección muestra las bases de datos que se utilizarán, y por último, en base a lo anterior, se elabora el programa principal denominado S.I.A.C., que es el objetivo principal de esta tesis.

4.1 DIAGRAMAS DE FLUJO DE PROGRAMACION

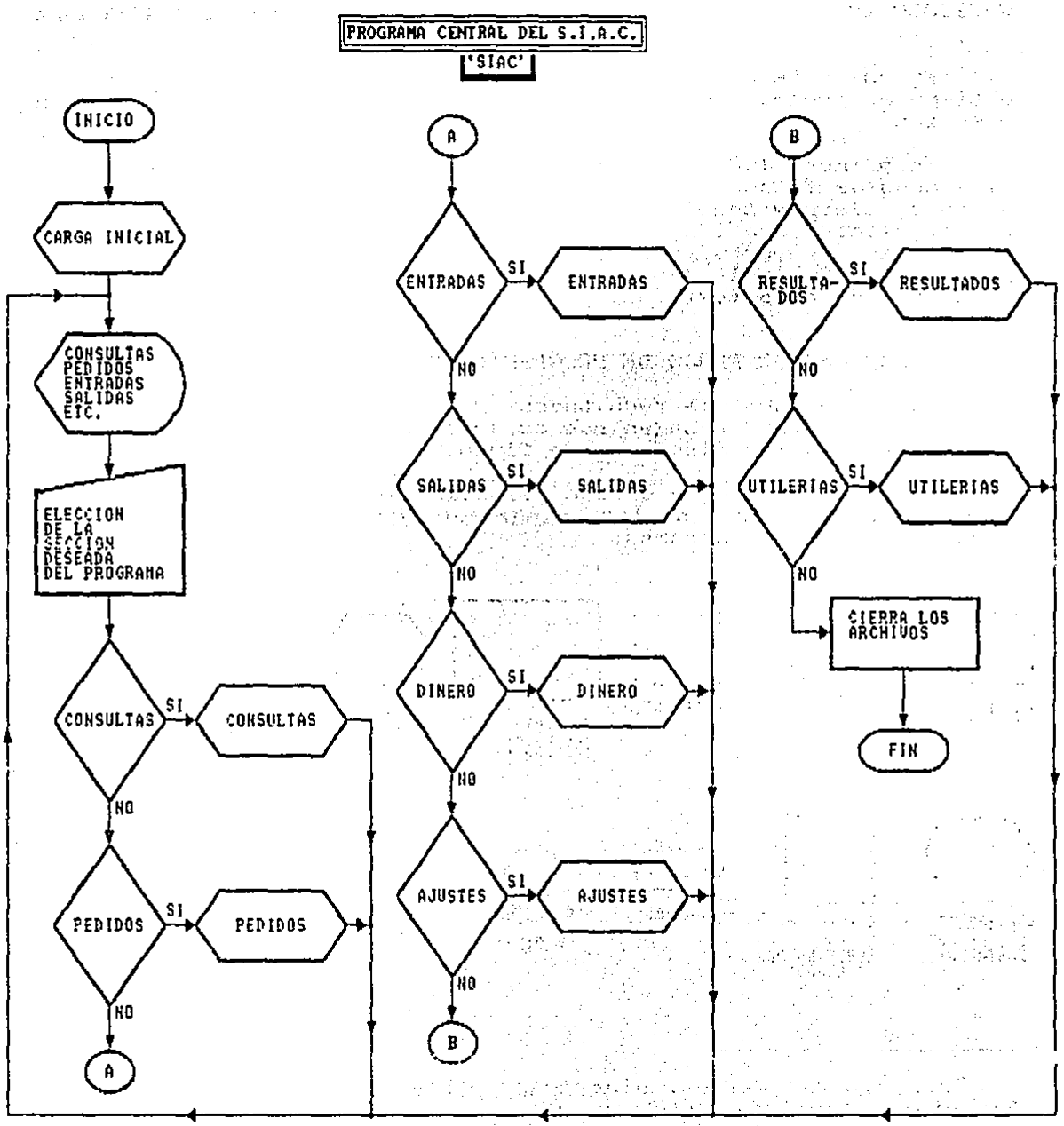
Las secciones de resultados y utilerías se dejan abiertas para que en ellas se hagan nuevos desarrollos y, por ello, no se presentarán en los diagramas de flujo.

A continuación se define el significado de cada uno de los signos[1] utilizados en los diagramas de flujo y después se presentan dichos diagramas.

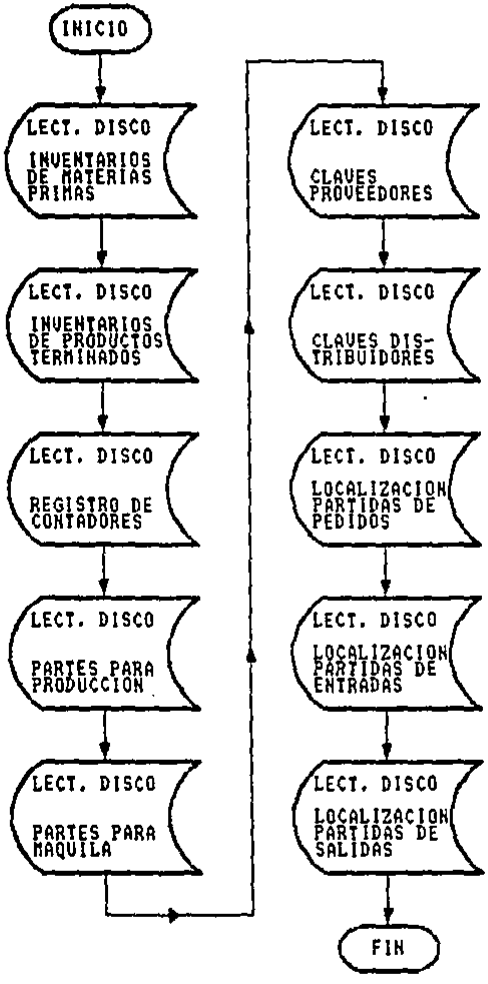


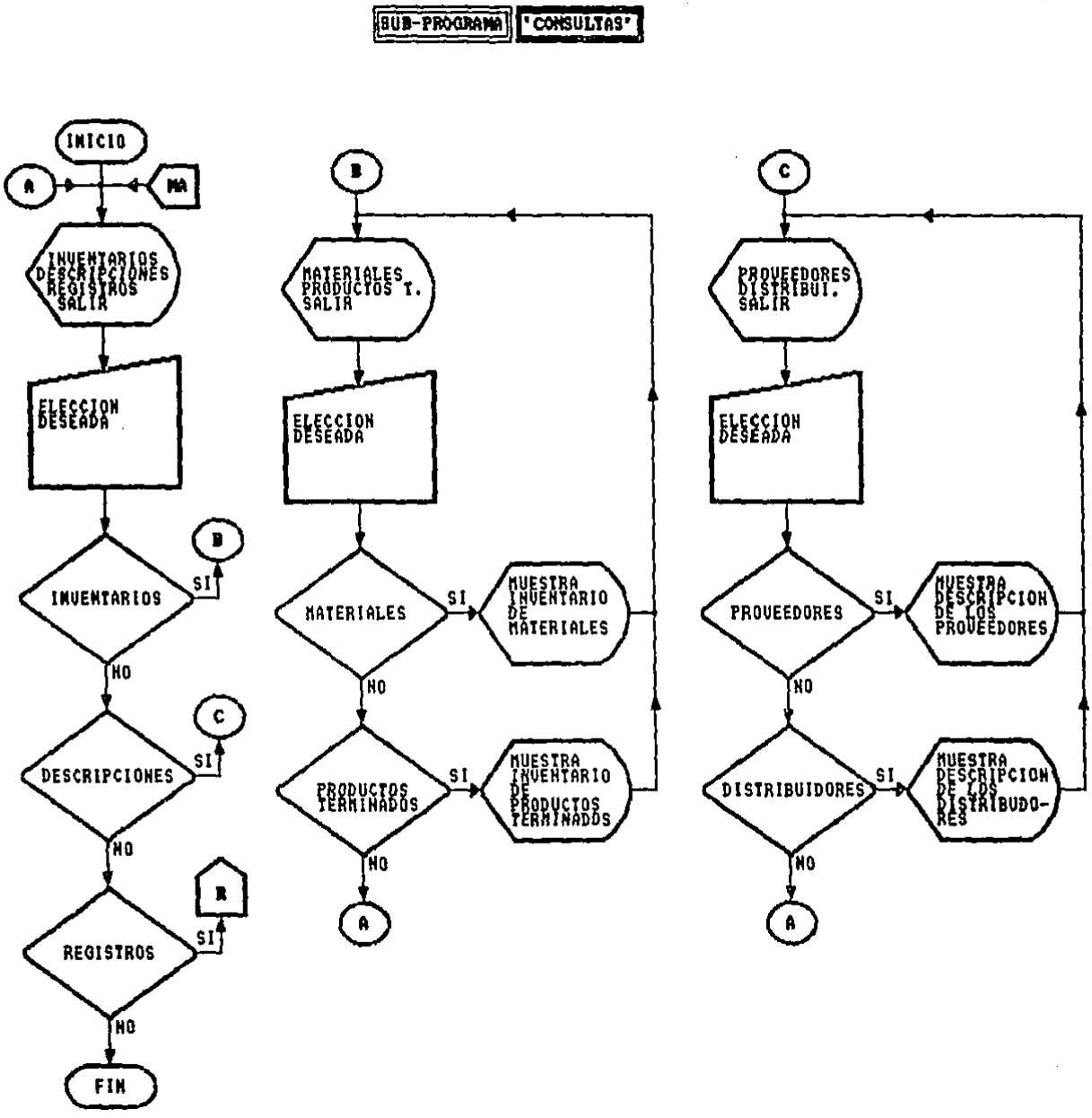
---

[1] Mario V. Farina, Diagramas de Flujo, 19a edición (México: Ed. Diana, Abril de 1987), p. 13.

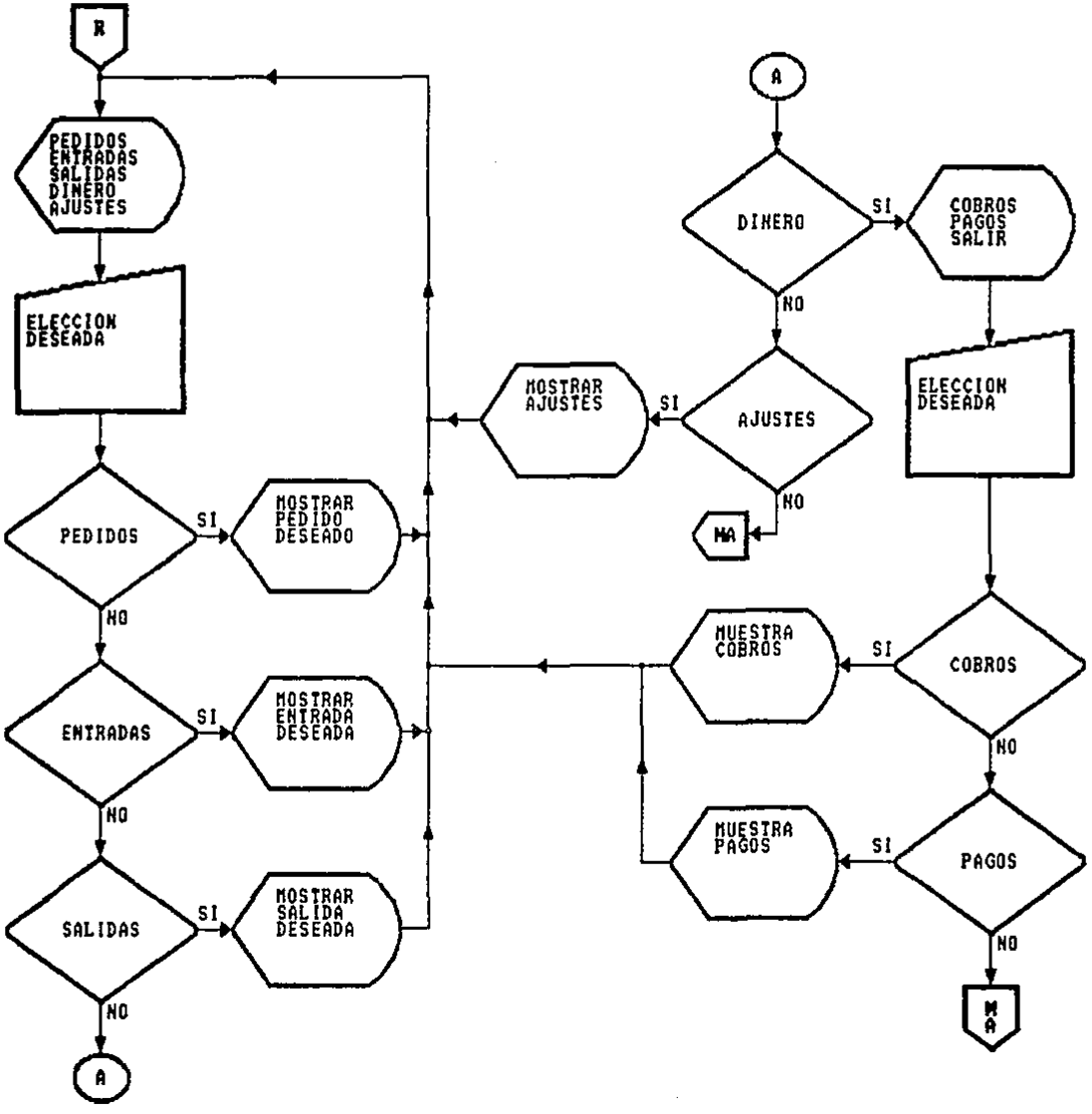

PROGRAMA CENTRAL DEL S.I.A.C.
'SIAC'
INICIO
CARGA INICIAL
CONSULTAS
PEDIDOS
ENTRADAS
SALIDAS
ETC.
ELECCION DE LA SECCION DESEADA DEL PROGRAMA
CONSULTAS
SI
CONSULTAS
NO
PEDIDOS
SI
PEDIDOS
NO
A
A
ENTRADAS
SI
ENTRADAS
NO
SALIDAS
SI
SALIDAS
NO
DINERO
SI
DINERO
NO
AJUSTES
SI
AJUSTES
NO
B
B
RESULTA-DOS
SI
RESULTADOS
NO
UTILERIAS
SI
UTILERIAS
NO
CIERRA LOS ARCHIVOS
FIN

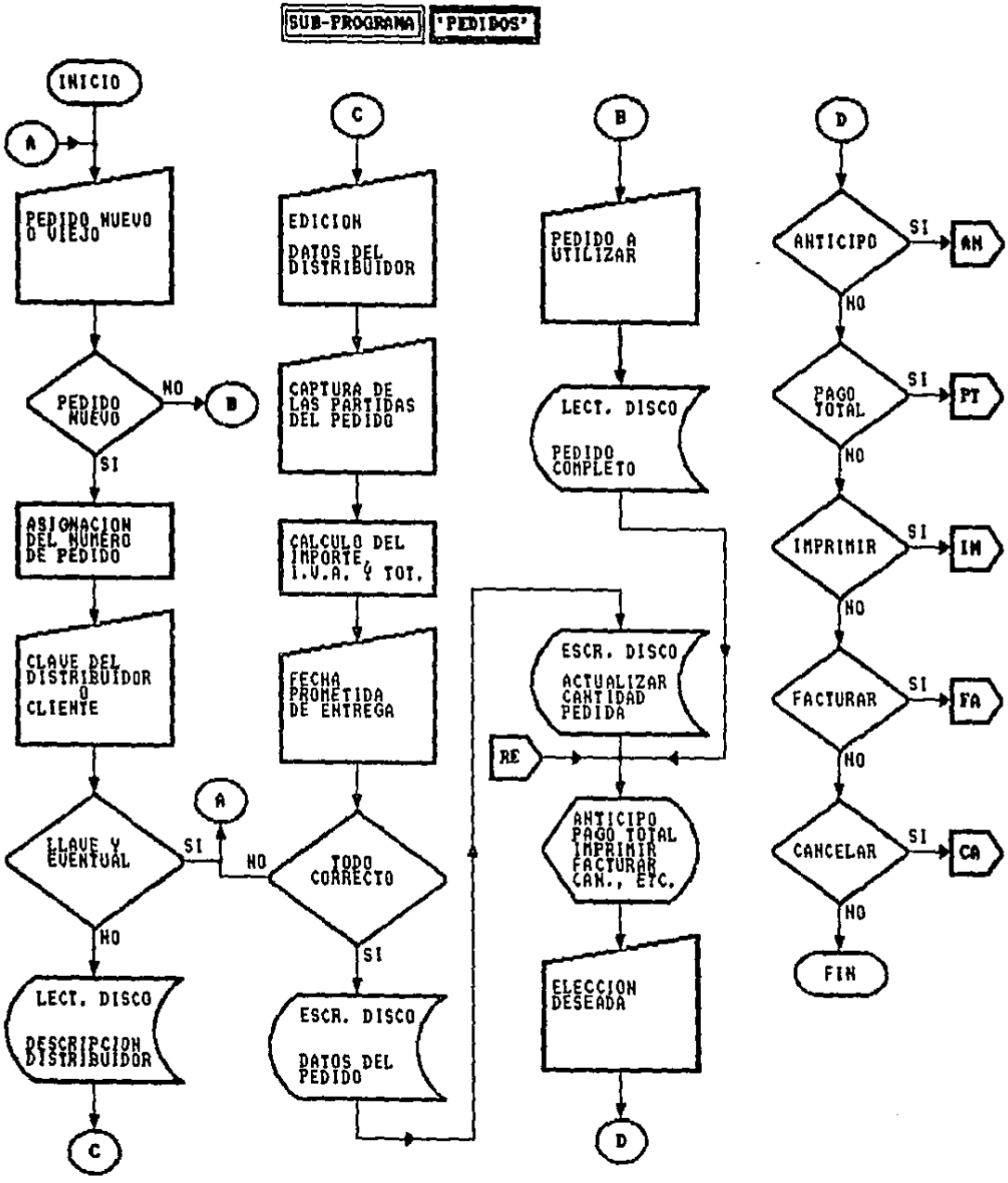
SUB-PROGRAMA 'CARGA INICIAL'



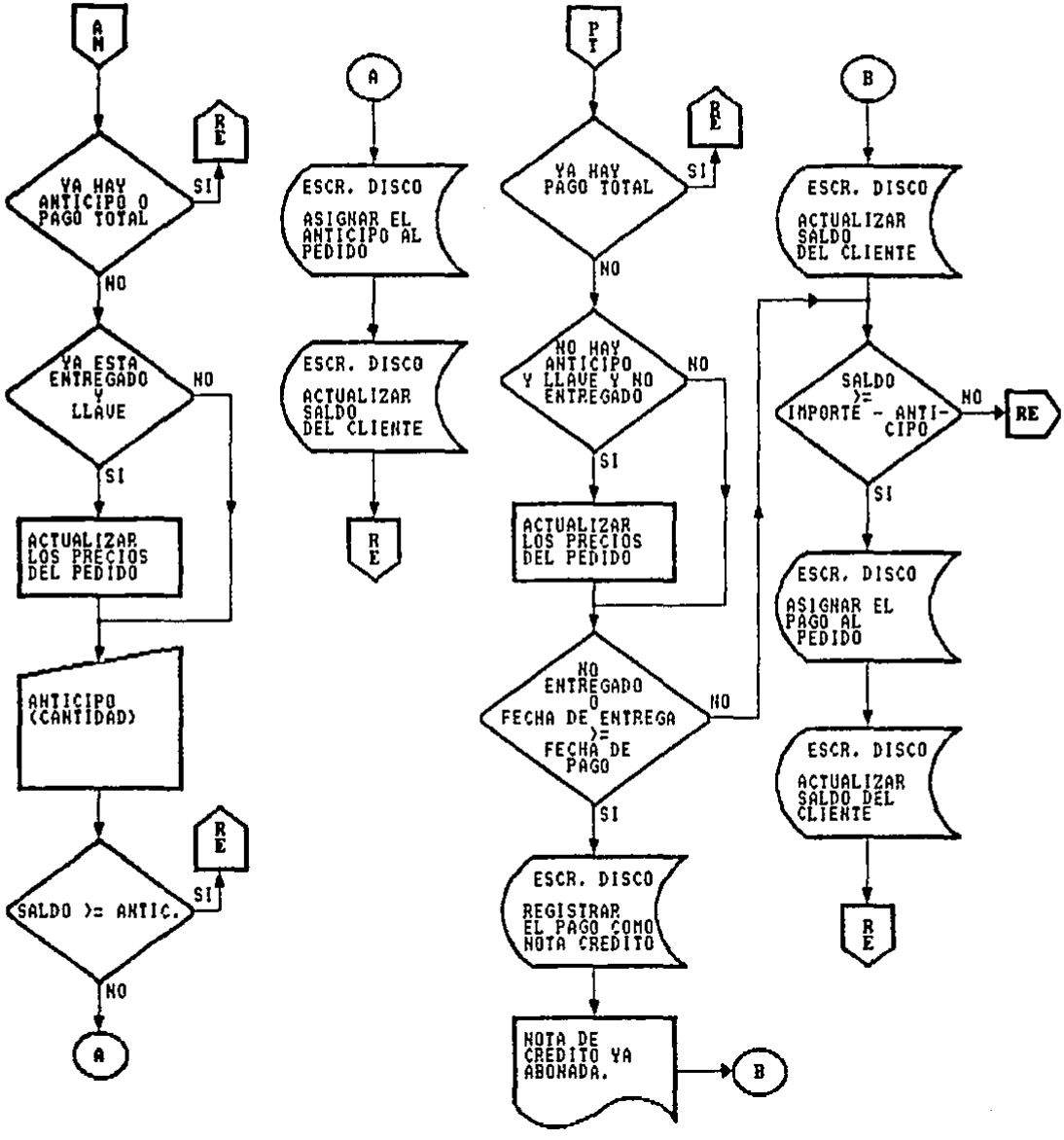

SUB-PROGRAMA "CONSULTAS"
INICIO
A
MA
INVENTARIOS DESCRIPCIONES REGISTROS SALIR
ELECCION DESEADA
INVENTARIOS
SI
B
NO
DESCRIPCIONES
SI
C
NO
REGISTROS
SI
R
NO
FIN
B
MATERIALES PRODUCTOS T. SALIR
ELECCION DESEADA
MATERIALES
SI
MUESTRA INVENTARIO DE MATERIALES
NO
PRODUCTOS TERMINADOS
SI
MUESTRA INVENTARIO DE PRODUCTOS TERMINADOS
NO
A
C
PROVEEDORES DISTRIBUI. SALIR
ELECCION DESEADA
PROVEEDORES
SI
MUESTRA DESCRIPCION DE LOS PROVEEDORES
NO
DISTRIBUIDORES
SI
MUESTRA DESCRIPCION DE LOS DISTRIBUDORES
NO
A

CONTINUACION SUB-PROGRAMA 'CONSULTAS'

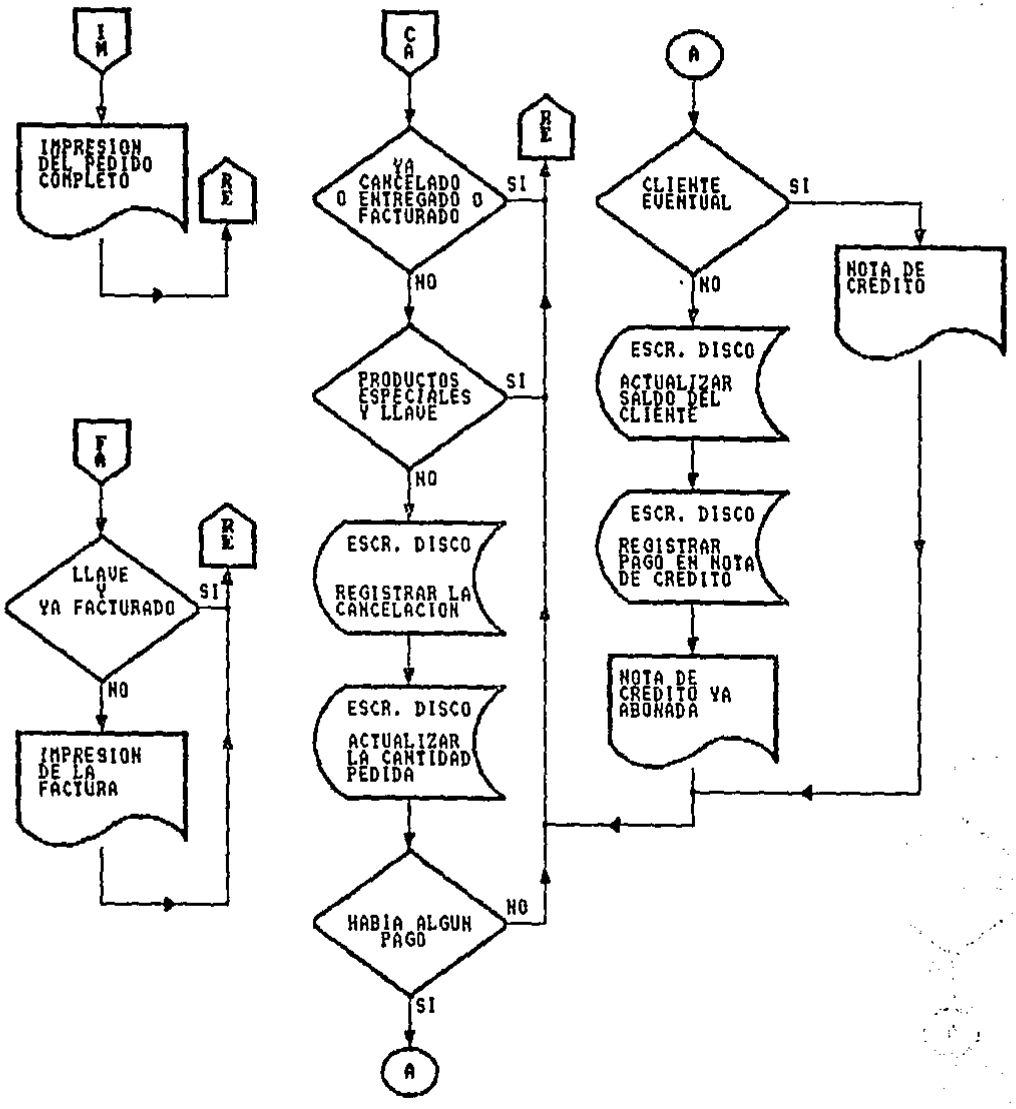



SUB-PROGRAMA 'PEDIDOS'
INICIO
A
PEDIDO NUEVO O VIEJO
PEDIDO NUEVO
NO
B
SI
ASIGNACION DEL NUMERO DE PEDIDO
CLAVE DEL DISTRIBUIDOR O CLIENTE
LLAVE Y EVENTUAL
SI
A
NO
NO
LECT. DISCO
DESCRIPCION DISTRIBUIDOR
C
C
EDICION
DATOS DEL DISTRIBUIDOR
CAPTURA DE LAS PARTIDAS DEL PEDIDO
CALCULO DEL IMPORTE, I.V.A. Y TOT.
FECHA PROMETIDA DE ENTREGA
TODO CORRECTO
SI
ESCR. DISCO
DATOS DEL PEDIDO
B
PEDIDO A UTILIZAR
LECT. DISCO
PEDIDO COMPLETO
ESCR. DISCO
ACTUALIZAR CANTIDAD PEDIDA
RE
ANTICIPO PAGO TOTAL IMPRIMIR FACTURAR CAN., ETC.
ELECCION DESEADA
D
D
ANTICIPO
SI
AN
NO
PAGO TOTAL
SI
PT
NO
IMPRIMIR
SI
IM
NO
FACTURAR
SI
FA
NO
CANCELAR
SI
CA
NO
FIN

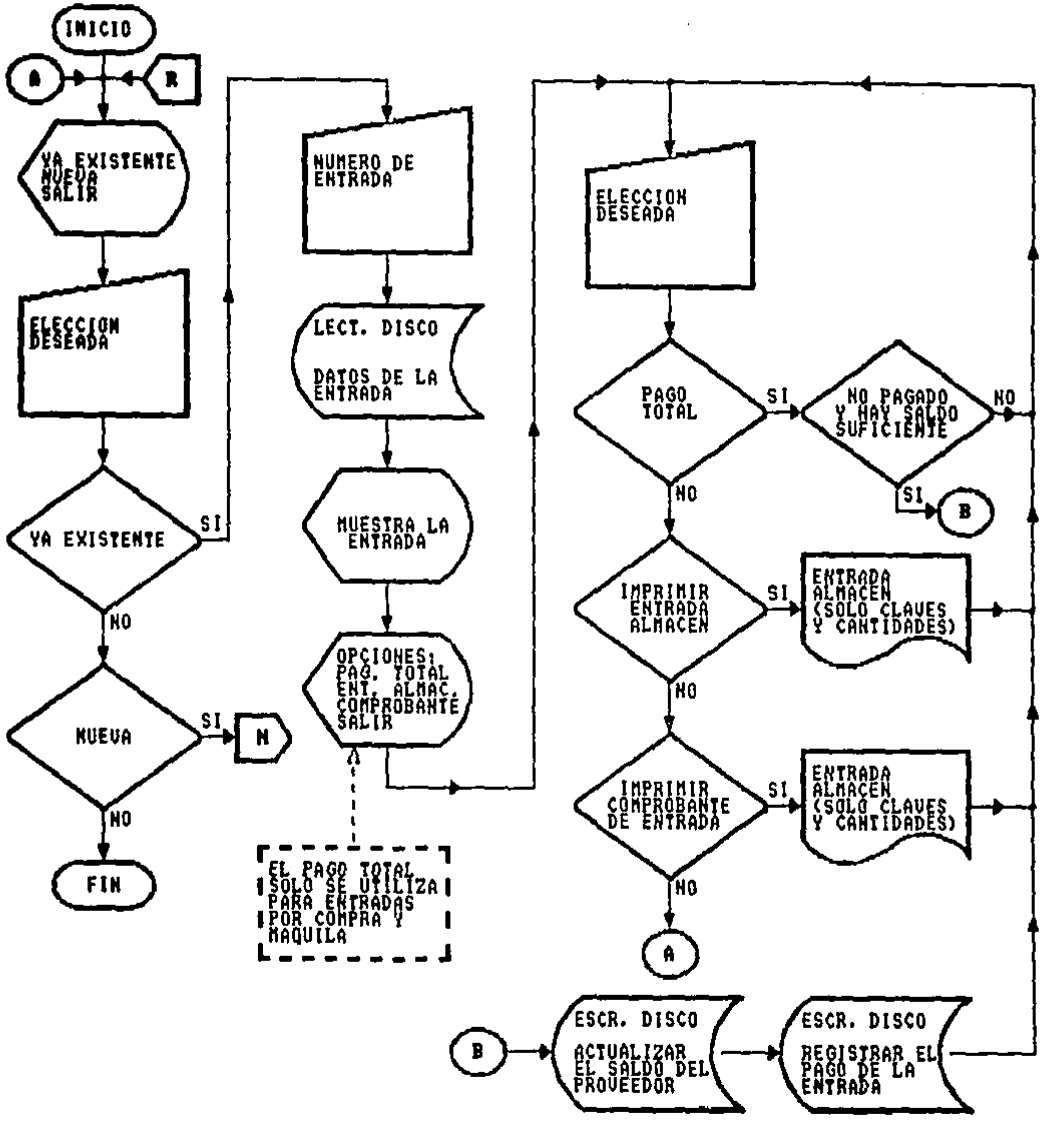
CONTINUACION SUB-PROGRAMA 'PEDIDOS'



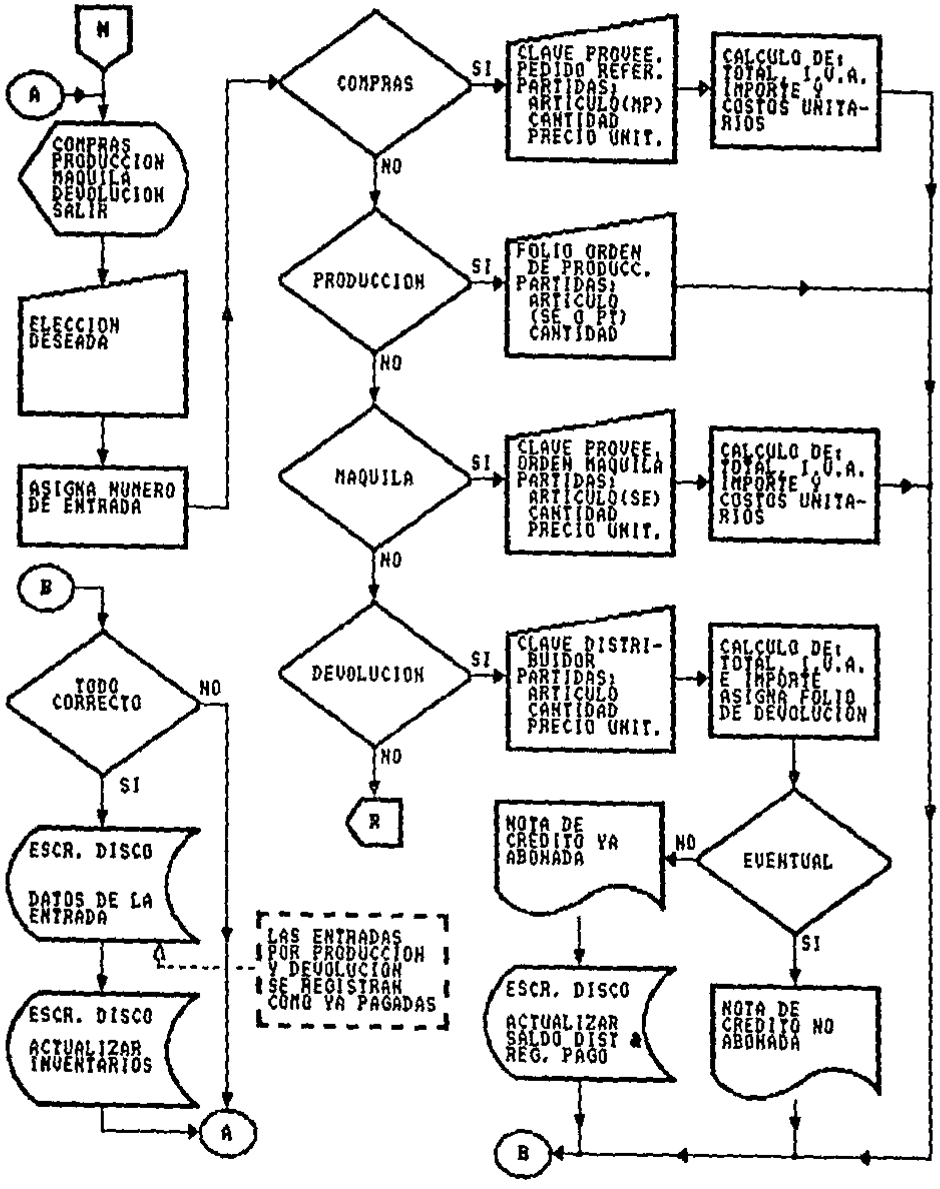
CONTINUACION SUB-PROGRAMA 'PEDIDOS'



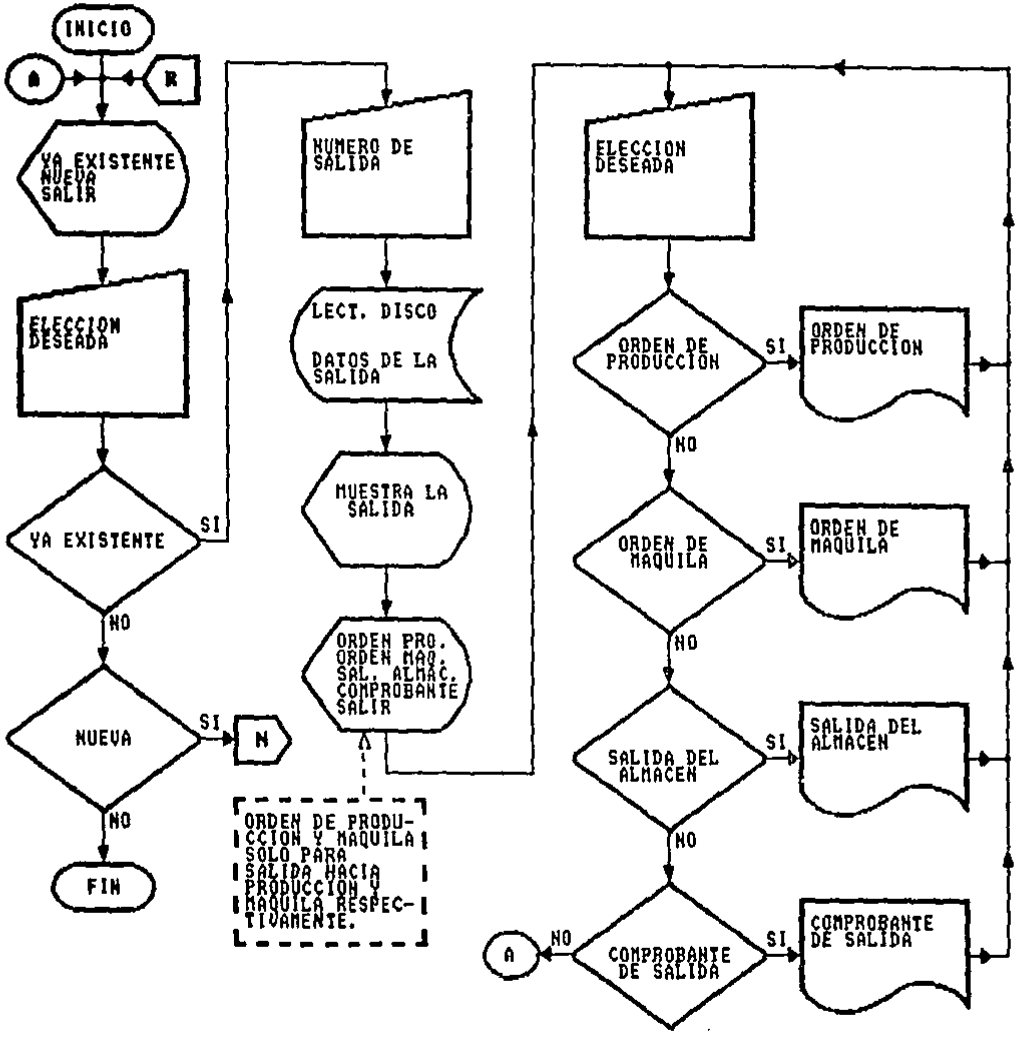
SUB-PROGRAMA 'ENTRADAS'



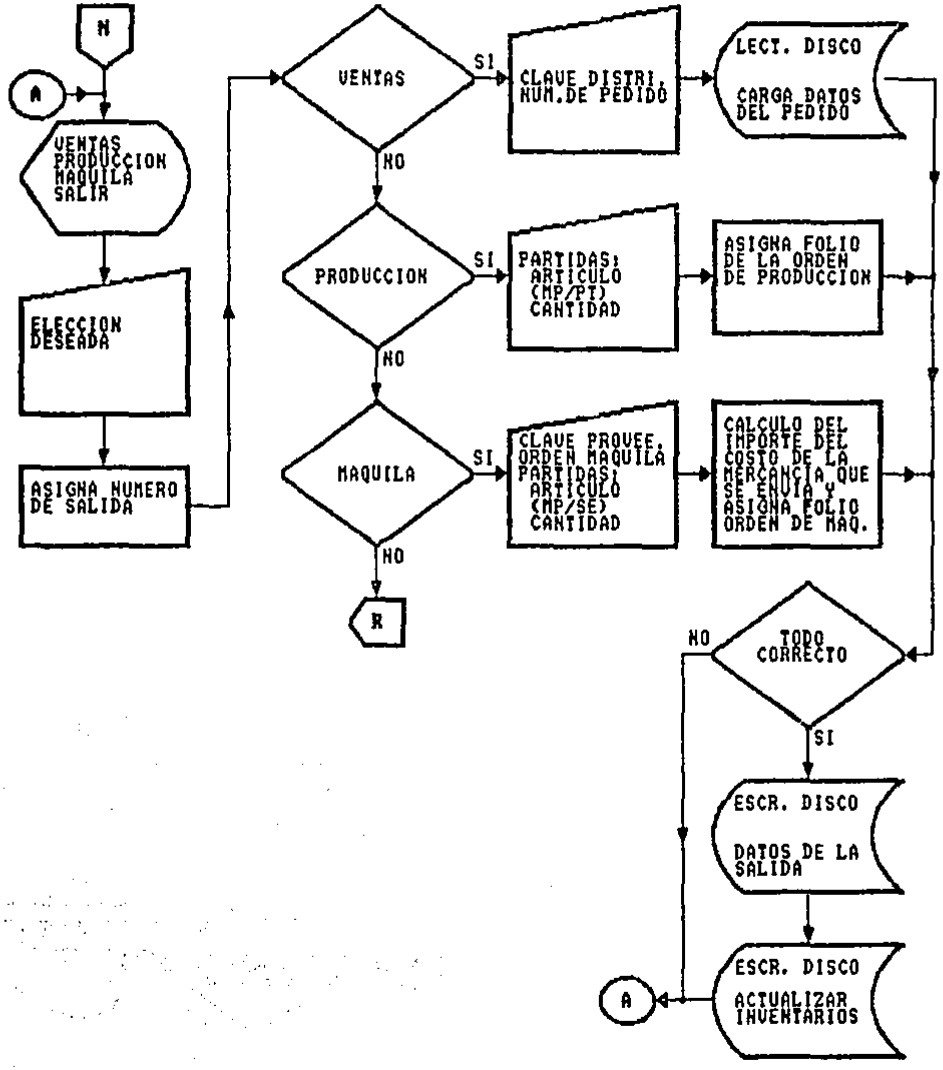
CONTINUACION SUB-PROGRAMA 'ENTRADAS'



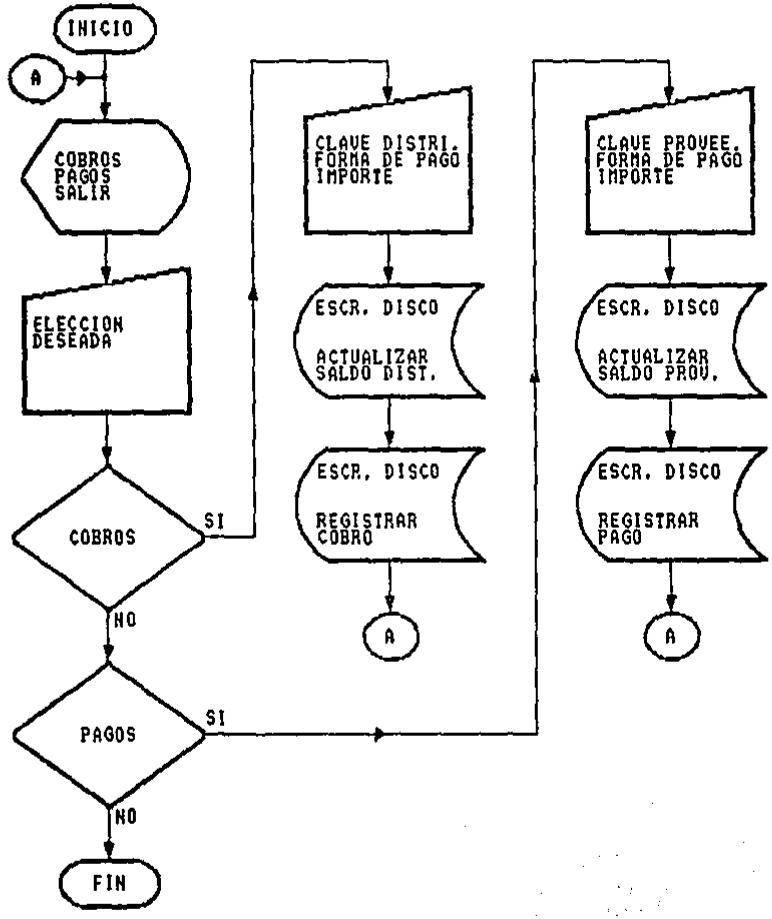
SUB-PROGRAMA 'SALIDAS'



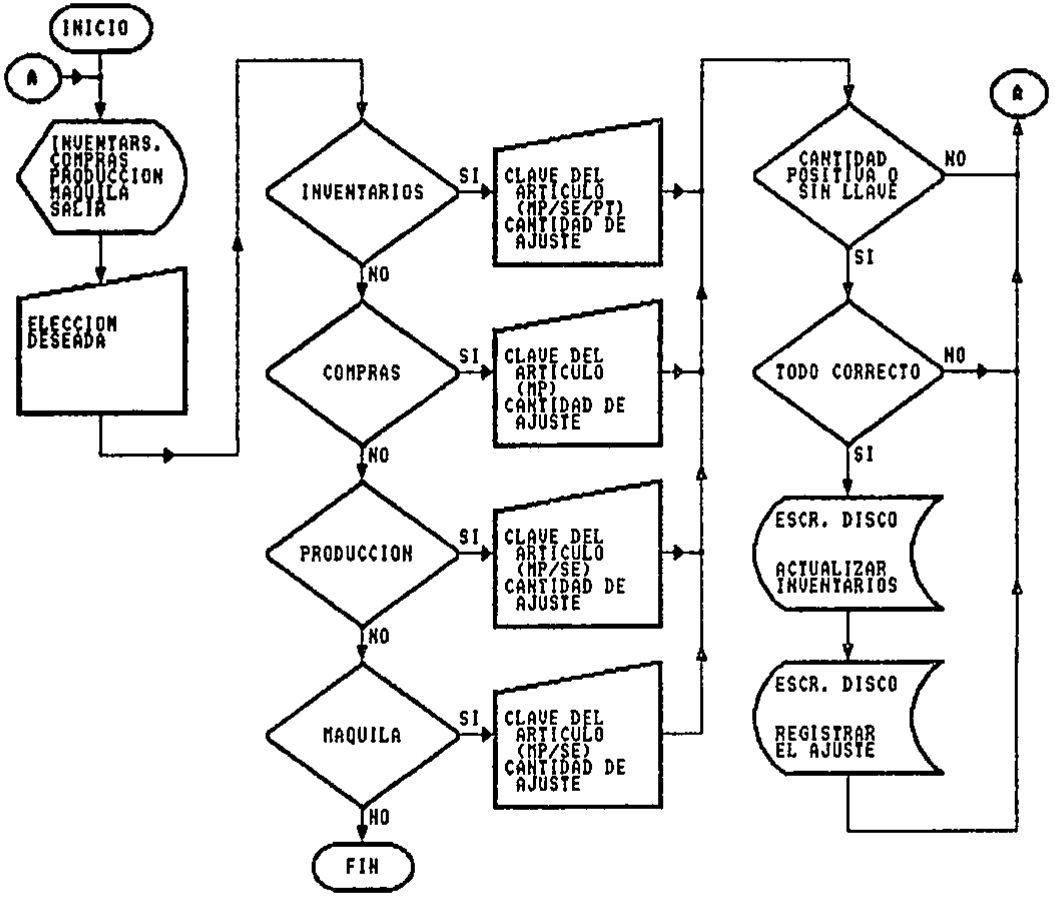
CONTINUACION SUB-PROGRAMA 'SALIDAS'



SUB-PROGRAMA "DINERO"



SUB-PROGRAMA 'AJUSTES'



4.2 ESTRUCTURA DE LAS BASES DE DATOS

En esta sección se muestran las bases de datos que se crearon para tener un control óptimo de todos los puntos anteriormente expuestos. Estas bases de datos se crearon por medio del dBASE III y se operan con el mismo dBASE III y también con el programa principal realizado en Pascal.

A partir de las necesidades planteadas en el capítulo anterior fue posible definir las bases de datos requeridas. Se buscó minimizar la longitud de los registros de las bases de datos por razones de velocidad y de espacio en memoria y en disco.

Para cada base de datos se indica:

+ Nombre (nombre del archivo de disco y nombre completo).

+ Los campos que la forman.
    + Campo (número de campo).
    + Nom. Campo (nombre del campo).
    + Tipo (es el tipo de campo: Carácter, Fecha, Lógico, Numérico o texto).
    + Ancho (número de caracteres que lo forman).
    + Dec (número de decimales en el caso de un campo numérico).

+ El número total de caracteres necesarios para cada registro de la base de datos.

Estas especificaciones son necesarias para definir perfectamente la estructura de las bases de datos a utilizar. Además, la última columna da una breve explicación de cada campo.

Se sugiere revisar brevemente cada base de datos para tener una idea más concreta de las mismas y poder entender satisfactoriamente la siguiente sección.

Estructura de la base de datos: CONT.dbf MEMORIA DE LOS CONTADORES

Campo	Nom. Campo	Tipo	Ancho	Dec	
1	DESC	Carácter	20		Descripción del contador.
2	CONT	Numérica	4		Conteo.
** Total **			25		

Estructura de la base de datos: INV_MP.dbf INVENTARIO DE MATERIA PRIMA

Campo	Nom. Campo	Tipo	Ancho	Dec	
1	ARTICULO	Carácter	7		Clave del artículo.
2	EXIST	Numérico	5		Existencias del artículo.
3	CA_PED	Numérico	5		Cantidad pedida (lo que se tiene pedido a compras, producción, maquila).
4	MIN	Numérico	5		Mínimo deseable en bodega.
5	MAX	Numérico	5		Máximo deseable en bodega.
6	RED	Numérico	1		Número de cifras de redondeo.
7	DESCR	Carácter	20		Descripción del artículo.
8	CLAVE	Carácter	7		Clave del artículo.
9	PRECIO_V	Numérico	12	2	Precio unitario de venta del artículo.
10	PRECIO_ULT	Numérico	12	2	Ultimo costo del artículo.
11	FECHA	Fecha	8		Fecha del último costeo del artículo.
** Total **			88		

Estructura de la base de datos: INV_PT.dbf INVENTARIO DE PRODUCTO TERMINADO

Campo	Nom. Campo	Tipo	Ancho	Dec	
1	ARTICULO	Carácter	7		Clave del artículo.
2	EXIST	Numérico	5		Existencias en bodega.
3	CA_DIS	Numérico	6		Cantidad disponible (EXIST - lo que se haya pedido), puede ser negativo.
4	CA_PED	Numérico	5		Cantidad pedida (lo que se tiene en producción).
5	MIN	Numérico	5		Mínimo deseable en bodega.
6	MAX	Numérico	5		Máximo deseable en bodega.
7	RED	Numérico	1		Número de cifras de redondeo.
8	DESCR	Carácter	20		Descripción del artículo.
9	PRECIO_V	Numérico	12	2	Precio de venta.
10	PRECIO_ULT	Numérico	12	2	Ultimo costo del artículo.
11	FECHA	Fecha	8		Fecha del último costeo del artículo.
** Total **			87		

Estructura de la base de datos: PROD.dbf PARTES PARA PRODUCCION

Campo	Nom. Campo	Tipo	Ancho	Dec	
1	ARTICULO	Carácter	7		Clave del artículo.
2	PARTES	Carácter	75		Partes que lo forman.
** Total **					

Estructura de la base de datos: MAQ.dbf PARTES PARA MAQUILA

Campo	Nom. Campo	Tipo	Ancho	Dec	
1	ARTICULO	Carácter	7		Clave del articulo.
2	PARTES	Carácter	75		Partes que lo forman.
3	PRECIO_ULT	Numérico	12	2	Ultimo precio unitario de la maquila.
4	FECHA	Fecha	8		Fecha del último envio.
** Total **			103		

Estructura de la base de datos: DES_PRO.dbf DESCRIPCION DE PROVEEDORES

Campo	Nom. Campo	Tipo	Ancho	Dec	
1	CLAVE	Carácter	7		Clave del proveedor.
2	NOMBRE	Carácter	40		Nombre.
3	DIRECCION	Carácter	20		Dirección.
4	TELEFONO	Carácter	10		Teléfono.
5	CIUDAD	Carácter	20		Ciudad.
6	ESTADO	Carácter	20		Estado.
7	C_P	Carácter	5		Código postal.
8	CDP	Numérico	3		Condiciones de pago (dias).
9	SALDO	Numérico	12	2	Saldo actual de la empresa (acreedor).
10	SALDO_ANT	Numérico	12	2	Saldo al último cierre.
11	FECHA_SA	Fecha	8		Fecha del último cierre.
** Total **			158		

Estructura de la base de datos: DES_DIS.dbf DESCRIPCION DE DISTRIBUIDORES

Campo	Nom. Campo	Tipo	Ancho	Dec	
1	CLAVE	Carácter	7		Clave del distribuidor.
2	NOMBRE	Carácter	40		Nombre.
3	DIRECCION	Carácter	20		Dirección.
4	TELEFONO	Carácter	10		Teléfono.
5	CIUDAD	Carácter	20		Ciudad.
6	ESTADO	Carácter	20		Estado.
7	C_P	Carácter	5		Código postal.
8	EMB_A	Carácter	20		Embárquese a.
9	CDP	Numérico	3		Condiciones de pago (Dias).
10	D	Numérico	2		Descuento.
11	DP	Numérico	2		Descuento por pronto pago.
12	SALDO	Numérico	12	2	Saldo actual (siempre a favor del distribuidor).
13	L_CREDITO	Numérico	12	2	Limite de crédito.
14	CREDITO	Numérico	12	2	Crédito actual otorgado.
15	SALDO_ANT	Numérico	12	2	Saldo al último cierre.
16	FECHA_SA	Fecha	8		Fecha del último cierre.
** Total **			206		

Estructura de la base de datos: R_PED.dbf REGISTRO DE PEDIDOS

Campo	Nom. Campo	Tipo	Ancho	Dec	
1	PEDIDO	Numérico	4		Folio del pedido.
2	C	Carácter	1		Cerrado 'S' o 'N'.
3	FECHA	Fecha	8		Fecha de pedido.
4	CAN	Numérico	3		Código de cancelación.
5	FECHA_CAN	Fecha	8		Fecha de cancelación.
6	FECHA_HE	Fecha	8		Fecha esperada de entrega.
7	FECHA_ENT	Fecha	8		Fecha en que se entregó.
8	CLAVE	Carácter	7		Clave del distribuidor.
9	NOMBRE	Carácter	40		Nombre del cliente.
10	DIRECCION	Carácter	20		Dirección (calle y número).
11	TELEFONO	Carácter	10		Teléfono.
12	CIUDAD	Carácter	20		Ciudad.
13	ESTADO	Carácter	20		Estado.
14	C_P	Carácter	5		Código postal.
15	EMB_A	Carácter	20		Embárquese a.
16	COP	Numérico	3		Condiciones de pago (en días).
17	D	Numérico	2		Descuento asignado.
18	DP	Numérico	2		Descuento por pronto pago.
19	FECHA_COB	Fecha	8		Fecha en que se debería cobrar.
20	AP	Carácter	1		Se aplicó el descuento por pronto pago ('S' o 'N').
21	IMPORTE	Numérico	12	2	Importe del pedido (ya con I.V.A. para pedidos por facturar).
22	PAG_ANT	Numérico	12	2	Importe del anticipo.
23	FECHA_ANT	Fecha	8		Fecha en que se recibió el anticipo.
24	PAG_TOT	Numérico	12	2	Importe de pago total.
25	FECHA_PT	Fecha	8		Fecha de pago total.
26	FACTURA	Numérico	4		Número de factura (<0 si es sin I.V.A.).
27	FECHA_FAC	Fecha	8		Fecha en que se facturó.
28	N_P	Numérico	2		Número de partidas (Máximo 10).
** Total **			265		

Estructura de la base de datos: R_PED_P.dbf REGISTRO DE PEDIDOS POR PARTIDA

Campo	Nom. Campo	Tipo	Ancho	Dec	
1	PEDIDO	Numérico	4		Folio del pedido.
2	ARTICULO	Carácter	7		Clave del artículo.
3	CANTIDAD	Numérico	5		Número de unidades.
4	DESCR	Carácter	20		Descripción del artículo.
5	PRECIO	Numérico	12	2	Precio unitario del artículo.
** Total **			49		

Estructura de la base de datos: R_ENT.dbf REGISTRO DE ENTRADAS

Campo	Nom. Campo	Tipo	Ancho	Dec	
1	ENTRADA	Numérico	4		Folio de la entrada.
2	ORIGEN	Carácter	1		Origen de la entrada ('C' por compra,'P' por producción, 'M' por maquila y 'D' por devolución de un cliente).
3	RAZON	Numérico	3		Código de razón.
4	CLAVE	Carácter	7		Clave del proveedor o distribuidor.
5	PEDIDO	Numérico	4		Número de pedido de referencia.
6	FECHA	Fecha	8		Fecha en que se hizo la entrada.
7	FECHA_PAG	Fecha	8		Fecha esperada de pago.
8	FECHA_PT	Fecha	8		Fecha en que se pagó.
9	IMPORTE	Numérico	12	2	Importe de la entrada.
10	N_P	Numérico	2		Número de partidas de la entrada (Máximo 10).
** Total **			58		

Estructura de la base de datos: R_ENT_P.dbf REGISTRO DE ENTRADAS POR PARTIDA

Campo	Nom. Campo	Tipo	Ancho	Dec	
1	ENTRADA	Numérico	4		Folio de la entrada.
2	ARTICULO	Carácter	7		Clave del artículo.
3	CANTIDAD	Numérico	5		Número de unidades.
4	PRECIO	Numérico	12	2	Precio unitario.
** Total **			29		

Estructura de la base de datos: R_SAL.dbf REGISTRO DE SALIDAS

Campo	Nom. Campo	Tipo	Ancho	Dec	
1	SALIDA	Numérico	4		Folio de la salida.
2	ORIGEN	Carácter	1		Origen de la salida ('V' por ventas, 'P' por producción o 'M' por maquila).
3	RAZON	Numérico	3		Código de razón.
4	CLAVE	Carácter	7		Clave del proveedor o distribuidor.
5	PEDIDO	Numérico	4		Pedido de referencia.
6	FECHA	Fecha	8		Fecha de la salida.
7	IMPORTE	Numérico	12	2	Importe de la salida.
8	N_P	Numérico	2		Número de partidas (Máximo 10).
** Total **			42		

Estructura de la base de datos: R_SAL_P.dbf REGISTRO DE SALIDAS POR PARTIDA

Campo	Nom. Campo	Tipo	Ancho	Dec	
1	SALIDA	Numérico	4		Folio de la salida.
2	ARTICULO	Carácter	7		Clave del artículo.
3	CANTIDAD	Numérico	5		Número de unidades.
4	PRECIO	Numérico	12	2	Precio unitario del artículo.
** Total **			29		

Estructura de la base de datos: R_COB.dbf REGISTRO DE COBROS

Campo	Nom. Campo	Tipo	Ancho	Dec	
1	CLAVE	Carácter	7		Clave del distribuidor.
2	FECHA	Fecha	8		Fecha en que se recibió el cobro.
3	FORM_PAG	Carácter	7		Forma de cobro.
4	IMPORTE	Numérico	12	2	Importe del cobro.
** Total **			35		

Estructura de la base de datos: R_PAG.dbf REGISTRO DE PAGOS

Campo	Nom. Campo	Tipo	Ancho	Dec	
1	CLAVE	Carácter	7		Clave del proveedor.
2	FECHA	Fecha	8		Fecha en que se hizo el pago.
3	FORM_PAG	Carácter	7		Forma de pago.
4	IMPORTE	Numérico	12	2	Importe del pago.
** Total **			35		

Estructura de la base de datos: R_AJU.dbf REGISTRO DE AJUSTES

Campo	Nom. Campo	Tipo	Ancho	Dec	
1	AJUSTE	Numérico	4		Folio del ajuste.
2	ORIGEN	Carácter	1		Inventario que se afectó ('I', 'P', 'M', 'C').
3	RAZON	Numérico	3		Código de razón.
4	ARTICULO	Carácter	7		Clave del artículo.
5	CANTIDAD	Numérico	5		Número de unidades de ajuste (puede ser negativo).
6	PRECIO	Numérico	12	2	Precio unitario.
7	FECHA	Fecha	8		Fecha de ajuste.
** Total **			41		

4.3 PROGRAMACION

Esta es la última etapa del desarrollo del programa principal que se realiza en lenguaje Pascal en su versión "Turbo Pascal 3.01A", creado por Borland International.

Al desarrollar el código fuente para Pascal surgen nuevas necesidades y detalles sutiles que no se preveían en el momento de la planeción general del programa, por lo que en esta sección se aclaran con todo detalle hasta las últimas ambigüedades que pudieran existir.

En primer lugar se hará la presentación general del programa y a continuación, se mostrarán las definiciones y declaraciones del programa S.I.A.C. con sus respectivos comentarios; después, se presentarán los procedimientos y funciones de propósito general y, en seguida, las rutinas que aparecen repetidamente a lo largo del programa. Por último, se presentarán los subprogramas que cumplen con lo expuesto en los diagramas de flujo.

4.3.1 PRESENTACION GENERAL DEL PROGRAMA

Para lograr una mayor claridad en la exposición del programa, se mostrarán primero en forma jerárquica, las rutinas que componen al programa en su totalidad.

Notas:

- El signo '~' (tilde) nos indica que dicha rutina es utilizada en otras secciones del programa y que es siervo de otra u otras rutinas.
- El signo '<' (menor que) nos indica que dicha rutina es siervo de otra u otras rutinas.
- Los nombres de los procedimientos y funciones no se acentuarán ya que esta versión del Turbo Pascal no lo permite.

En lo sucesivo, los comentarios explicativos se pondrán entre corchetes y los niveles se esquematizarán por indentación.

```
S_I_A_C                                  ( Programa central del S.I.A.C. )

  ( Primeros procedimientos de propósito general. )
  <Alarma                                          ( Suena una alarma. )
  <Timbre                                          ( Suena un timbre. )
  MensajeErr                          ( Presenta los mensajes de error. )
  Fechar                              ( Carga la fecha y hora del sistema. )
    <CambioTipo          ( Convierte un número de dos digitos en una cadena. )
  JusCad                       ( Justifica una cadena a una longitud dada. )
  JusEnte              ( Da la presentación a una variable de tipo entera. )
```

```
JusReal                  ( Da la presentación a una variable de tipo real. )
JusFolio               ( Da la presentación de folio a una variable entera. )
SigFolio                                ( Calcula el siguiente folio. )
AntFolio                                ( Calcula el folio anterior. )
OkFolio                          ( Verifica que el folio sea correcto. )
Solapar                                 ( Solapa una cadena con otra. )
Posibilidad                    ( Presenta una posibilidad de elección. )

( Subprograma de entrada de datos por teclado. )
<Editar                          ( Sección general de edición de entradas. )
PregByte                 ( Edición de entrada para una variable tipo byte. )
PregEnte                ( Edición de entrada para una variable tipo entera. )
PregReal                ( Edición de entrada para una variable tipo real. )
PregCar              ( Edición de entrada para una variable tipo carácter. )
PregCadMay    ( Edición de entrada para mayúsculas de una variable cadena. )
PregCad               ( Edición de entrada para una variable tipo cadena. )
PregOpcion          ( Permite la entrada de las teclas que se especifiquen. )

( Subprograma de acceso a los archivos del dBase III. )
Iniciacion            ( Inicializa la variable de control de base de datos. )
<naValido                  ( Verifica que el número de archivo sea válido. )
<LecturaCar                              ( Lee un carácter del archivo. )
AbrirArchivo                             ( Abre el archivo deseado. )
  <Saltar                ( Salta cierto número de bytes del archivo. )
  <LecturaI                          ( Lee un número entero del archivo. )
<YaAbierto                    ( Verifica que el archivo ya esté abierto. )
<Compactar    ( Elimina los espacios en blanco por la derecha e izquierda. )
<PosRegCamp              ( Da la posición del registro y campo especificado. )
  <VerifRegCamp           ( Verifica que el registro y campo sean válidos. )
    <AgRegArch                ( Agrega un registro en blanco al archivo. )
<Leer                             ( Lee del archivo el campo encontrado. )
<Escribir                  ( Escribe una cadena en el campo encontrado. )
  <FormatoC             ( Da el formato apropiado a campos de tipo carácter. )
LeerByte                      ( Lee una variable tipo byte del archivo. )
LeerEnte                   ( Lee una variable tipo entera del archivo. )
LeerReal                      ( Lee una variable tipo real del archivo. )
LeerCar                    ( Lee una variable tipo carácter del archivo. )
LeerCad                     ( Lee una variable tipo cadena del archivo. )
LeerFecha             ( Lee una variable tipo cadena de un campo de fechas. )
EscribirByte                ( Escribe una variable tipo byte en el archivo. )
EscribirEnte             ( Escribe una variable tipo entera en el archivo. )
EscribirReal                ( Escribe una variable tipo real en el archivo. )
EscribirCar              ( Escribe una variable tipo carácter en el archivo. )
EscribirCad               ( Escribe una variable tipo cadena en el archivo. )
EscribirFecha          ( Escribe una variable cadena en un campo de fechas. )
CerrarArchivo                            ( Cierra el archivo deseado. )

( Ultimos procedimientos de propósito general. )
OkPrepararImpresora          ( Verifica que la impresora esté preparada. )
Terminar                  ( Cierra los archivos para salir del programa. )
```

```
( Sección de programas centrales del S.I.A.C. )
CargaInicial                    ( Carga los datos que permanecerán en memoria. )
Consultas                       ( Pasa a la sección de consultas. )
  ^CargaProveedor               ( Lee del disco la descripción del proveedor. )
  ^CargaDistribuidor            ( Lee del disco la desc. del distribuidor. )
  Inventarios                   ( Consulta a inventarios. )
    Materiales                  ( Consulta al inventario de materiales (m.p. y s.e.). )
    Productos                   ( Consulta al inventario de productos terminados. )
  Descripciones                 ( Consulta a descripciones. )
    Proveedores                 ( Consulta a descripción de proveedores. )
    Distribuidores              ( Consulta a descripción de distribuidores. )
  Registros                     ( Consulta a registros. )
    Pedidos                     ( Consulta al registro de pedidos. )
      ^CargaPedido              ( Lee del disco el pedido especificado. )
      MostrarCabezaPedido       ( Muestra datos del cliente del pedido. )
      ^MostrarPartidas          ( Muestra las partidas del pedido. )
      ^MostrarOtros             ( Muestra los datos del estado del pedido. )
    Entradas                    ( Consulta al registro de entradas. )
      ^CargaEntrada             ( Lee del disco la entrada especificada. )
      ^MostrarEntrada           ( Muestra los datos de la entrada. )
    Salidas                     ( Consulta al registro de salidas. )
      ^CargaSalida              ( Lee del disco la salida especificada. )
      ^MostrarSalida            ( Muestra los datos de la salida. )
    Ajustes                     ( Consulta al registro de ajustes. )
      <CargaAjuste              ( Lee del disco el ajuste especificado. )
    Dinero                      ( Consulta a los registros de dinero. )
      Cobros                    ( Consulta al registro de cobros. )
        <CargaCobro             ( Lee del disco el cobro especificado. )
      Pagos                     ( Consulta al registro de pagos. )
        <CargaPago              ( Lee del disco el pago especificado. )
Pedidos                         ( Pasa a la sección de pedidos. )
  ^OkDistribuidor               ( Verifica que esté registrado el distribuidor. )
  ^CargaDistribuidor            ( Lee del disco la descripción del dist. )
  ^OkMaterial                   ( Verifica que sea un material registrado. )
  ^OkProducto                   ( Verifica que sea un producto terminado registrado. )
  ^CargaPedido                  ( Lee del disco el pedido especificado. )
  ^MostrarCabezaPedidoExtra     ( Muestra datos del cliente del pedido. )
  ^MostrarPartidas              ( Muestra las partidas del pedido. )
  ^MostrarOtros                 ( Muestra los datos del estado del pedido. )
  ^OkCambioPrecios              ( Verifica que los precios estén correctos. )
  ^CambiarPrecios               ( Actualiza los precios del pedido. )
  <OkPrimerosDatos              ( Verifica que se introduzcan los datos iniciales. )
  Anticipo                      ( Asigna un anticipo al pedido. )
  PagoTotal                     ( Asigna el pago total al pedido. )
    CadenaExterna               ( Inserta caracteres a un formato de impresión. )
    ^OkImprimirForma            ( Manda a impresión un formato dado. )
  ImprimirPedido                ( Imprime el pedido. )
    CadenaExterna               ( Inserta caracteres a un formato de impresión. )
    ^OkImprimirForma            ( Manda a impresión un formato dado. )
  Facturar                      ( Factura el pedido. )
    CadenaExterna               ( Inserta caracteres a un formato de impresión. )
    ^OkImprimirForma            ( Manda a impresión un formato dado. )
```

```
  Cancelar                                                    { Cancela el pedido. }
    CadenaExterna                { Inserta caracteres a un formato de impresión. }
    ^OkImprimirForma                      { Manda a impresión un formato dado. }
Entradas                                                  { Pasa a la sección de entradas. }
  ^OkMaterial                          { Verifica que sea un material registrado. }
  ^OkProducto              { Verifica que sea un producto terminado registrado. }
  ^OkProveedor                        { Verifica que esté registrado el proveedor. }
  ^OkDistribuidor          { Verifica que esté registrado el distribuidor. }
  Nuevas                                        { Para registrar una nueva entrada. }
    Compras                                  { Registro de una entrada por compras. }
    Producción                          { Registro de una entrada desde producción. }
      ^OkSubensamble             { Verifica que sea un subensamble registrado. }
      ^OkDesProduccion           { Verifica que exista la descripción de partes. }
      ^OkPartesProduccion        { Verifica y encuentra las partes integrantes. }
    Maquila                             { Registro de una entrada desde maquila. }
      ^OkMaquila                 { Verifica que sea una maquila registrada. }
      ^OkDesMaquila              { Verifica que exista la descripción de partes. }
      ^OkPartesMaquila           { Verifica y encuentra las partes integrantes. }
    Devolucion                          { Registro de una entrada por devolución. }
      ^CargaDistribuidor         { Lee del disco la descripción del distribuidor. }
      CadenaExterna              { Inserta caracteres a un formato de impresión. }
      ^OkImprimirForma                    { Manda a impresión un formato dado. }
  YaExistentes                      { Para trabajar con una entrada ya existente. }
      ^CargaEntrada                    { Lee del disco la entrada especificada. }
      ^MostrarEntrada                     { Muestra los datos de la entrada. }
      CadenaExterna              { Inserta caracteres a un formato de impresión. }
      ^OkImprimirForma                    { Manda a impresión un formato dado. }
      ImprimirEntrada                  { Imprime el comprobante de entrada. }
      ImprimirEntradaAlmacen                 { Imprime la entrada al almacén. }
      PagoTotal                        { Asigna el pago total a la entrada. }
Salidas                                                   { Pasa a la sección de salidas. }
  ^OkMaterial                          { Verifica que sea un material registrado. }
  ^OkProducto              { Verifica que sea un producto terminado registrado. }
  ^OkProveedor                        { Verifica que esté registrado el proveedor. }
  ^OkDistribuidor          { Verifica que esté registrado el distribuidor. }
  Nuevas                                        { Para registrar una nueva salida. }
    Ventas                                    { Registra una salida por ventas. }
      ^CargaDistribuidor     { Lee del disco la descripción del distribuidor. }
      ^CargaPedido                    { Lee del disco el pedido especificado. }
      ^MostrarCabezaPedidoExtra       { Muestra datos del cliente del pedido. }
      ^MostrarPartidas                      { Muestra las partidas del pedido. }
      ^MostrarOtros                  { Muestra los datos del estado del pedido. }
      ^OkCambioPrecios            { Verifica que los precios estén correctos. }
      ^CambiarPrecios                       { Actualiza los precios del pedido. }
      <RegSalida                                      { Registra la salida. }
    Produccion                          { Registra una salida hacia producción. }
      ^OkSubensamble             { Verifica que sea un subensamble registrado. }
      ^OkDesProduccion           { Verifica que exista la descripción de partes. }
      ^OkPartesProduccion             { Verifica y encuentra partes integrantes. }
    Maquilas                                 { Registra una salida hacia maquila. }
      ^OkMaquila                     { Verifica que sea una maquila registrada. }
      ^OkDesMaquila              { Verifica que exista la descripción de partes. }
```

```
      ^OkPartesMaquila                ( Verif. y encuentra partes integrantes. )
  YaExistentes                      ( Para trabajar con una salida ya existente. )
    ^CargaSalida                      ( Lee del disco la salida especificada. )
    ^CargaProveedor                 ( Lee del disco la descripción del proveedor. )
    ^MostrarSalida                      ( Muestra los datos de la salida. )
    CadenaExterna                 ( Inserta caracteres a un formato de impresión. )
    ^OkImprimirForma                  ( Manda a impresión un formato dado. )
    ImprimirSalida                    ( Imprime el comprobante de salida. )
    ImprimirSalidaAlmacen                 ( Imprime la salida de almacén. )
    SalidaAlmacenProduccion           ( Imprime salida almacén hacia prod. )
      ^OkSubensamble                ( Verifica que sea un subensamble registrado. )
      ^OkDesProduccion              ( Verifica que exista la descripción de partes. )
      ^OkPartesProduccion           ( Verifica y encuentra las partes integrantes. )
    SalidaAlmacenMaquila              ( Imprime salida de almacén hacia maquila. )
      ^OkMaquila                      ( Verifica que sea una maquila registrada. )
      ^OkDesMaquila                 ( Verifica que exista la descripción de partes. )
      ^OkPartesMaquila              ( Verifica y encuentra las partes integrantes. )
Dinero                                    ( Pasa a la sección de pagos y cobros. )
  Cobros                                      ( Pasa a la sección de cobros. )
    ^OkDistribuidor                 ( Verifica que esté registrado el distribuidor. )
  Pagos                                       ( Pasa a la sección de pagos. )
    ^OkProveedor                    ( Verifica que esté registrado el proveedor. )
Ajustes                                     ( Pasa a la sección de ajustes. )
  ^OkMaterial                       ( Verifica que sea un material registrado. )
  ^OkProducto                   ( Verifica que sea un producto terminado registrado. )
  Inventarios                         ( Ajustes en los inventarios en bodega. )
  Compras                         ( Ajustes a cantidad pedida de materias primas. )
  Produccion                    ( Ajustes al inventario de producción en proceso. )
  Maquila                         ( Ajustes al inventario de maquila en proceso. )
    CadenaExterna                 ( Inserta caracteres a un formato de impresión. )
    ^OkImprimirForma                    ( Manda a impresión un formato dado. )
    ^OkDesMaquila                 ( Verifica que exista la descripción de partes. )
    ^OkProveedor                    ( Verifica que esté registrado el proveedor. )
    ^CargaProveedor               ( Lee del disco la descripción del proveedor. )
Resultados                                  ( Pasa a la sección de resultados. )
Utilerias                                   ( Pasa la sección de utilerías. )
```

Después de revisar detenidamente esta sección se tendrá una idea clara de la jerarquía y localización de cada procedimiento y función utilizados en el desarrollo del programa.

4.3.2 DEFINICIONES Y DECLARACIONES DEL S.I.A.C.

En esta sección se presenta el código fuente del programa S.I.A.C., con los siguientes elementos:

+ La cabecera.
+ Definición de constantes.
+ Definición de tipos.
+ Declaración de variables.
+ Declaración de subprogramas (procedimientos y funciones).
  Nota: los subprogramas aparecen como ficheros incluidos y los listados de cada uno de ellos se presentarán más adelante.
+ Conjunto de sentencias.

Las extensiones de los nombres de ficheros de interés se interpretan como sigue:

EXTENSION	DESCRIPCION
PAS	Código fuente de un programa en Pascal.
INC	Código fuente de un fichero incluido.
OVL	Código fuente de un fichero de solapamiento.
BBL	Código fuente de un fichero utilizado como biblioteca de procedimientos y funciones.
COM	Código objeto ejecutable (programa compilado).
XXX	En donde las X's representan números cualesquiera. Código objeto de un fichero de solapamiento (solapamiento compilado).

A continuación se presenta el listado del programa con sus respectivos comentarios explicativos.

SIAC.PAS (Nombre del fichero que contiene el código fuente que se muestra.)

```
PROGRAM S_I_A_C;                                    ( PROGRAMA CENTRAL DEL S.I.A.C. )
|
CONST                                               ( DEFINICION DE CONSTANTES. )
|
| NEGRO       = 0;   AZUL          = 1;             ( Definición de colores para monitor en color RGB. )
| VERDE       = 2;   CIAN          = 3;
| ROJO        = 4;   MAGENTA       = 5;
| MARRON      = 6;   BLANCO        = 7;
| GRIS        = 8;   AZUL_CLARO    = 9;
| VERDE_CLARO = 10;  CIAN_CLARO    = 11;
| AMARILLO    = 12;  BLANCO_INTENSO = 13;
|
| LINEA     = #95;                                  ( Carácter de subrayado '_'. )
| COMILLA   = #39;                                  ( Carácter comilla. )
| ESC       = #27;                                  ( Carácter de escape. )
| CR        = #13;                                  ( Carácter de retorno de carro. )
```

```
  ENTER     = #13;                    { Carácter de retorno de carro utilizado por 'Editar'. }
  CLA_MAQ   = '^';                    { Clave para subensambles de maquila. }
  CLA_SUBE  = '[';                    { Clave para subensambles de producción. }
  EMPRESA   = 'RIMA';                 { Clave de la compañía que utiliza el programa. }
  LLAVE     = 'JDF83+';               { Seis caracteres que componen la llave del sistema. }
  MAX_LONG  = 78;                     { Máxima longitud de la visualización en pantalla. }
  MININT    = -32767;                 { Entero mínimo. }
  IVA       = 0.15;                   { Fracción que compone el I.V.A. }

  { Mandos de impresora. }
  LF  = #10;                          { Avance de linea en impresora. }
  FF  = #12;                          { Avance de página en impresora. }

  { Para el manejo de archivos }
  NUM_MAX_ARCH = 5;                   { Número máximo de archivos abiertos simultáneamente. }
  LONG_NOM_ARCH = 40;                 { Longitud máxima del nombre del archivo incluyendo el camino. }
  LONG_MAX_CAMP = 255;                { Longitud máxima de los campos. }
  NUM_MAX_CAMP = 128;                 { Número máximo de campos en una base de datos. }
  RETORNO = #13;                      { Carácter de retorno de carro. }

  { Límites de datos mantenidos en memoria }
  MAX_M_P    = 100;                   { Máximo de materias primas. }
  MAX_P_T    = 1000;                  { Máximo de productos terminados. }
  MAX_PROD   = 100;                   { Máximo de productos y subensambles manufacturados en la planta. }
  MAX_MAQ    = 100;                   { Máximo de productos de maquila. }
  MAX_PARTES = 50;                    { Máximo de componentes de una maquila o subensamble. }
  MAX_DIS    = 1000;                  { Máximo de distribuidores. }
  MAX_PRO    = 200;                   { Máximo de proveedores. }
  MAX_PED    = 5000;                  { Máximo de pedidos controlados por el programa. }
  MAX_ENT    = 2000;                  { Máximo de entradas controladas por el programa. }
  MAX_SAL    = 2000;                  { Máximo de salidas controladas por el programa. }
  MAX_N_P    = 10;                    { Máximo de partidas. Nota: NO ALTERARLO. }

  { Configuración de las bases de datos. }
  TAM_CLAVE    = 7;                   { Longitud máxima para las claves. }
  TAM_ARTIC    = 7;                   { Longitud máxima para las claves de los artículos. }
  TAM_NOMBRE   = 40;                  { Longitud máxima para el nombre del cliente o distribuidor. }
  TAM_DIREC    = 20;                  { Longitud máxima para la dirección. }
  TAM_TELEF    = 10;                  { Longitud máxima para el número telefónico. }
  TAM_CIUDAD   = 20;                  { Longitud máxima para el nombre de la ciudad. }
  TAM_ESTADO   = 20;                  { Longitud máxima para el nombre del estado. }
  TAM_C_P      = 5;                   { Longitud máxima para el código postal. }
  TAM_ENB_A    = 20;                  { Longitud máxima para el lugar de destino. }
  TAM_FORM_PAG = 7;                   { Longitud máxima para la descripción de la forma de pago. }
  TAM_DESC     = 20;                  { Longitud máxima para la descripción del artículo. }
  TAM_PARTES   = 75;                  { Longitud máxima para la descripción de partes que componen un artículo. }

  { Variables inicializadas. }
  espacio: Char   = #32;              { Carácter de espacio. }
  espacios : String[MAX_LONG] =  '                                                                              ';
  lineas   : String[MAX_LONG] =  '______________________________________________________________________________';
  diagonales: String[20]      =  '////////////////////';
```

```
con_llave: Boolean = TRUE;                    { Se inicializa la variable llave a verdadero. }

TYPE                                          { DEFINICION DE TIPOS. }

  Tipos   = (T_BYTE,T_ENTERO,T_REAL,          { Especifica los diferentes tipos de variables para los procedimientos }
             T_CAR,T_CAD_MAY,T_CAD   );       { de edición de entrada por teclado. }

  { Tipos de cadenas de uso general. }
  Cad4    = String[4];                        { Cadena de cuatro caracteres. }
  Cad78   = String[MAX_LONG];                 { Cadena de setenta y ocho caracteres. }
  Cad255  = String[255];                      { Cadena de doscientos cincuenta y cinco caracteres. }
  TFecha  = String[8];                        { Cadena de ocho caracteres necesarios para fechas. }
  TTecla  = Array[1..2] OF Char;              { Rango de dos caracteres para identificar cualquier tecla. }
  TPant = Array[1..2000] OF RECORD            { Rango para almacenar en memoria toda una pantalla con sus atributos. }
                              caracter: Char; { Carácter de pantalla. }
                              atributo: Byte; { Atributo de color e intensidad. }
                            END;

  { Para el manejo de archivos. }
  Nom = String[LONG_NOM_ARCH];                { Tipo nombre de archivo. }
  Base = RECORD                               { Tipo base de datos y su descripción. }
           arch:      File OF Char;               { Acceso al disco carácter a carácter. }
           abierto:   Boolean;                    { Indica si está abierto o cerrado el archivo. }
           nom_arch:  Nom;                        { Nombre del archivo incluyendo el camino. }
           num_regs:  Integer;                    { Indica el número de registros que contiene la base de datos. }
           tam_desc:  Integer;                    { Número de caracteres utilizados para la descripción de la b. d. }
           long_reg:  Integer;                    { Indica la longitud del registro. }
           num_camps: Integer;                    { Indica el número de campos que contiene la base de datos. }
           tipo_camp: Array[1..NUM_MAX_CAMP] OF Char;     { Indica el tipo de cada campo. }
           long_camp: Array[1..NUM_MAX_CAMP] OF Byte;     { Indica la longitud de cada campo. }
           pos_camp:  Array[1..NUM_MAX_CAMP] OF Integer;  { Indica la posición del campo. }
           num_dec:   Array[1..NUM_MAX_CAMP] OF Byte;     { Indica el número de decimales. }
         END;

  { Datos leídos del disco. }
  TClave   = String[TAM_CLAVE];               { Tipo clave. }
  TArtic   = String[TAM_ARTIC];               { Tipo artículo. }
  TNombre  = String[TAM_NOMBRE];              { Tipo nombre. }
  TDirec   = String[TAM_DIREC];               { Tipo dirección. }
  TTelef   = String[TAM_TELEF];               { Tipo número telefónico. }
  TCiudad  = String[TAM_CIUDAD];              { Tipo nombre de ciudad. }
  TEstado  = String[TAM_ESTADO];              { Tipo nombre de estado. }
  TEmb_A   = String[TAM_EMB_A];               { Tipo lugar de embarque. }
  TC_P     = String[TAM_C_P];                 { Tipo código postal. }
  TFormPag = String[TAM_FORM_PAG];            { Tipo descripción de la forma de pago. }
  TDesc    = String[TAM_DESC];                { Tipo descripción de un artículo. }
  TPartes  = String[TAM_PARTES];              { Tipo partes que componen un artículo. }

  { Tipos definidos con el fin de almacenar datos en la memoria dinámica. }
  TPntMP = ^TMP;                              { Tipo puntero de materia prima. }
  TMP = RECORD                                { Tipo materia prima. }
          artic:     TArtic;                     { Clave del artículo. }
```

```
        exist:      Integer;                ( Existencias. )
        ca_ped:     Integer;                ( Cantidad pedida. )
        min:        Integer;                ( Mínimo requerido. )
        max:        Integer;                ( Máximo requerido. )
        red:        Byte;                   ( Número de cifras de redondeo. )
        desc:       TDesc;                  ( Descripción del artículo. )
        clave:      TClave;                 ( Clave del proveedor. )
        precio_v:   Real;                   ( Precio de venta. )
        precio_ult: Real;                   ( Ultimo costo calculado. )
        fech:       TFecha;                 ( Fecha del último costeo. )
      END;
TPntPT = ^TPT;                          ( Tipo puntero de producto terminado. )
TPT = RECORD                            ( Tipo producto terminado. )
        artic:      TArtic;                 ( Clave del artículo. )
        exist:      Integer;                ( Existencias. )
        ca_dis:     Integer;                ( Cantidad disponible. )
        ca_ped:     Integer;                ( Cantidad pedida. )
        min:        Integer;                ( Mínimo requerido. )
        max:        Integer;                ( Máximo requerido. )
        red:        Byte;                   ( Número de cifras de redondeo. )
        desc:       TDesc;                  ( Descripción del artículo. )
        precio_v:   Real;                   ( Precio de venta. )
        precio_ult: Real;                   ( Ultimo costo calculado. )
        fech:       TFecha;                 ( Fecha del último costeo. )
      END;
TPntProd = ^TProd;                      ( Tipo puntero de partes para la producción. )
TProd = RECORD                          ( Tipo partes para la producción. )
          artic:    TArtic;                 ( Clave del artículo. )
          partes:   TPartes;                ( Partes que componen el artículo. )
        END;
TPntMaq = ^TMaq;                        ( Tipo puntero de partes para la maquila. )
TMaq  = RECORD                          ( Tipo partes para la maquila. )
          artic:      TArtic;               ( Clave del artículo. )
          partes:     TPartes;              ( Partes que componen el artículo. )
          precio_ult: Real;                 ( Ultimo costo de la maquila. )
          fech:       TFecha;               ( Fecha del último costeo de la maquila. )
        END;
TDesPro = RECORD                        ( Tipo descripción del proveedor. )
            clave:  TClave;               ( Clave del proveedor. )
            nombre: TNombre;              ( Nombre del proveedor. )
            direc:  TDirec;               ( Dirección del proveedor. )
            telef:  TTelef;               ( Teléfono del proveedor. )
            ciudad: TCiudad;              ( Ciudad de residencia. )
            estado: TEstado;              ( Estado de residencia. )
            c_p:    TC_P;                 ( Código postal. )
            cdp:    Byte;                 ( Condiciones de pago. )
            saldo:  Real;                 ( Saldo acreedor con el proveedor. )
          END;
TDesDis = RECORD                        ( Tipo descripción de distribuidor. )
            clave:  TClave;               ( Clave del distribuidor. )
            nombre: TNombre;              ( Nombre del distribuidor. )
            direc:  TDirec;               ( Dirección del distribuidor. )
```

```
          telef:  TTelef;                    { Teléfono del distribuidor. }
          ciudad: TCiudad;                   { Ciudad de residencia. }
          estado: TEstado;                   { Estado de residencia. }
          c_p:    TC_P;                      { Código postal. }
          emb_a:  TEmb_A;                    { Lugar de embarque. }
          cdp:    Byte;                      { Condiciones de pago. }
          d:      Byte;                      { Descuento asignado. }
          dp:     Byte;                      { Descuento por pronto pago asignado. }
          saldo:  Real;                      { Saldo deudor con el distribuidor. }
          l_cred: Real;                      { Límite de crédito otorgado al distribuidor. }
          cred:   Real;                      { Crédito utilizado. }
        END;
TPed = RECORD                              { Tipo pedido. }
        num:      Integer;                     { Folio del pedido. }
        cerr:     Char;                        { Indica si el pedido está cerrado; N = ABIERTO; S = CERRADO. }
        fech:     TFecha;                      { Fecha del pedido. }
        can:      Byte;                        { Clave de cancelación; 0 = NO CANCELADO; 1-255 = CANCELADO. }
        fech_can: TFecha;                      { Fecha de cancelación. }
        fech_he:  TFecha;                      { Fecha hipotética de entrega. }
        fech_ent: TFecha;                      { Fecha de entrega. }
        clave:    TClave;                      { Clave del distribuidor. }
        nombre:   TNombre;                     { Nombre del distribuidor. }
        direc:    TDirec;                      { Dirección del distribuidor. }
        telef:    TTelef;                      { Teléfono del distribuidor. }
        ciudad:   TCiudad;                     { Ciudad de residencia. }
        estado:   TEstado;                     { Estado de residencia. }
        c_p:      TC_P;                        { Código postal. }
        emb_a:    TEmb_A;                      { Lugar de embarque. }
        cdp:      Byte;                        { Condiciones de pago. }
        d:        Byte;                        { Descuento. }
        dp:       Byte;                        { Descuento por pronto pago. }
        fech_cob: TFecha;                      { Fecha de cobro. }
        ap:       Char;                        { Indica si se aplicó el descuento por pronto pago. }
        importe:  Real;                        { Importe del pedido. }
        pag_ant:  Real;                        { Importe del pago anticipado. }
        fech_pa:  TFecha;                      { Fecha de pago anticipado. }
        pag_tot:  Real;                        { Importe del pago total. }
        fech_pt:  TFecha;                      { Fecha del pago total. }
        factura:  Integer;                     { Folio de la factura. }
        fech_fac: TFecha;                      { Fecha de facturación. }
        n_p:      Integer;                     { Número de partidas. }
        part: Array[1..MAX_N_P] OF RECORD      { Partidas del pedido. }
                                    artic: TArtic;    { Clave del artículo. }
                                    cant:  Integer;   { Cantidad de unidades. }
                                    desc:  TDesc;     { Descripción del artículo. }
                                    precio: Real;     { Precio del artículo. }
                                  END;
      END;
TEnt = RECORD                              { Tipo entrada. }
        num:    Integer;                       { Número de entrada. }
        origen: Char;                          { Origen de la entrada. }
        razon:  Byte;                          { Razón que origina la entrada. }
```

```
        clave:    TClave;                    { Clave de la fuente de la entrada. }
        pedido:   Integer;                   { Folio del pedido o referencia. }
        fech:     TFecha;                    { Fecha de la entrada. }
        fech_pag: TFecha;                    { Fecha prometida de pago. }
        importe:  Real;                      { Importe de la entrada. }
        fech_pt:  TFecha;                    { Fecha de pago total. }
        n_p:      Integer;                   { Número de partidas de la entrada. }
        part: Array[1..MAX_N_P] OF RECORD    { Partidas de la entrada. }
                                   artic:  TArtic;    { Clave del artículo. }
                                   cant:   Integer;   { Cantidad de unidades. }
                                   precio: Real;      { Precio unitario. }
                                 END;
      END;
TSal = RECORD                                { Tipo salida. }
        num:      Integer;                   { Folio de la salida. }
        origen:   Char;                      { Origen de la salida. }
        razon:    Byte;                      { Razón que origina la salida. }
        clave:    TClave;                    { Clave del destinatario de la salida. }
        pedido:   Integer;                   { Folio del pedido o referencia. }
        fech:     TFecha;                    { Fecha de salida. }
        importe:  Real;                      { Importe de la salida. }
        n_p:      Integer;                   { Número de partidas de la salida. }
        part: Array[1..MAX_N_P] OF RECORD    { Partidas de la salida. }
                                   artic:  TArtic;    { Clave del artículo. }
                                   cant:   Integer;   { Cantidad de unidades. }
                                   precio: Real;      { Precio unitario. }
                                 END;
      END;
TPag = RECORD                                { Tipo pago. }
        clave:    TClave;                    { Clave del proveedor. }
        fech:     TFecha;                    { Fecha del pago. }
        fora_pag: TForaPag;                  { Forma de pago. }
        importe:  Real;                      { Importe del pago. }
      END;
TCob = RECORD                                { Tipo cobro. }
        clave:    TClave;                    { Clave del distribuidor. }
        fech:     TFecha;                    { Fecha del cobro. }
        fora_pag: TForaPag;                  { Forma de pago. }
        importe:  Real;                      { Importe del cobro. }
      END;
TAju = RECORD                                { Tipo ajuste. }
        num: Integer;                        { Folio del ajuste. }
        origen: Char;                        { Origen del ajuste. }
        razon:  Byte;                        { Razón del ajuste. }
        artic:  TArtic;                      { Clave del artículo ajustado. }
        cant:   Integer;                     { Cantidad de ajuste. }
        precio: Real;                        { Precio en el momento del ajuste. }
        fech:   TFecha;                      { Fecha del ajuste. }
      END;
```

```
  TPartUsad = Array[0..MAX_PARTES] OF RECORD     { Tipo partes usadas para formar otro artículo. }
                                       numero: Integer;   { Número de parte necesaria. }
                                       cant:   Integer;   { Cantidad necesaria. }
                                     END;

VAR

  { Variables de uso general. }
  un_byte: Byte;                               { Una variable tipo byte. }
  un_ente, num_err: Integer;                   { Una variable entera y el número de error recogido. }
  un_real: Real;                               { Una variable tipo real. }
  un_car: Char;                                { Una variable tipo carácter. }
  una_cad: Cad255;                             { Una variable tipo cadena de 255 caracteres. }
  ok: Boolean;                                 { Variable de verificación. }
  pantalla: TPant Absolute $B800:0000;         { Pantalla actual del PC $B000:0000 B/N y $B800:0000 Color. }

  { Para el procedimiento fechar. }
  horas: String[7];                            { Almacenará la hora del sistema. }
  hora, minuto: Byte; meridiano: String[4];
  fecha: TFecha;
  dia, mes, ano: Byte;                         { Almacenará la fecha del sistema. }

  { Para el manejo de archivos. }
  bas: Array[1..NUM_MAX_ARCH] OF Base;         { Datos de cada base de datos. }
  na: Byte;                                    { Número de archivo en uso. }
  car1, car2: Char;                            { Definidos como generales por }
  reg: Integer;                                { cuestiones de velocidad y para }
  camp: Byte;                                  { su posible consulta y uso desde }
  posicion: Real;                              { cualquier sitio del programa. }

  { Variables punteros para localizar información almacenada en la memoria dinámica. }
  pnt_mp:   Array[1..MAX_M_P]  OF TPntMP;      { Variable que contiene los punteros de materias primas. }
  pnt_pt:   Array[1..MAX_P_T]  OF TPntPT;      { Variable que contiene los punteros de productos terminados. }
  pnt_prod: Array[1..MAX_PROD] OF TPntProd;    { Variable que contiene los punteros de las partes para producción. }
  pnt_maq:  Array[1..MAX_MAQ]  OF TPntMaq;     { Variable que contiene los punteros de las partes para maquila. }

  { Variables para datos residentes en memoria. }
  dis:  Array[1..MAX_DIS] OF TClave;
  pro:  Array[1..MAX_PRO] OF TClave;
  ped_part: Array[1..MAX_PED] OF Integer;      { Almacena el número de la primera partida correspondiente a cada pedido. }
  ent_part: Array[1..MAX_ENT] OF Integer;      { Almacena el número de la primera partida correspondiente a cada entrada. }
  sal_part: Array[1..MAX_SAL] OF Integer;      { Almacena el número de la primera partida correspondiente a cada salida. }

  { Variables para lectura de archivos. }
  num_mp:   Integer;                           { Número de materias primas. }
  num_pt:   Integer;                           { Número de productos terminados. }
  num_prod: Integer;                           { Número de descripciones de producción. }
  num_maq:  Integer;                           { Número de descripciones de maquila. }
  num_dis:  Integer;                           { Número de distribuidores. }
  num_pro:  Integer;                           { Número de proveedores. }

  { Valores de folios iniciales y finales. }
```

```
  num_ent_i, num_ent_f, num_ent:  Integer;      { Número de entrada inicial, final, y total de entradas. }
  num_sal_i, num_sal_f, num_sal:  Integer;      { Número de salida inicial, final, y total de salidas. }
  num_ped_i, num_ped_f, num_ped:  Integer;      { Número de pedido inicial, final, y total de pedidos. }
  num_aju_i, num_aju_f, num_aju:  Integer;      { Número de ajuste inicial, final, y total de ajustes. }
  num_devol,                                    { Número de devolución. }
  num_fact,                                     { Número de factura. }
  num_not_cred,                                 { Número de nota de crédito. }
  num_ord_prod,                                 { Número de orden de producción. }
  num_ord_maq: Integer;                         { Número de orden de maquila. }

  { Nombres de los archivos de base de datos. }
  nom_arch_des_pro,                             { Nombre del archivo de descripciones de proveedores. }
  nom_arch_des_dis,                             { Nombre del archivo de descripciones de distribuidores. }
  nom_arch_ped, nom_arch_ped_p,                 { Nombre de los archivos de pedidos (datos generales y sus partidas). }
  nom_arch_ent, nom_arch_ent_p,                 { Nombre de los archivos de entradas (datos generales y sus partidas). }
  nom_arch_sal, nom_arch_sal_p,                 { Nombre de los archivos de salidas (datos generales y sus partidas). }
  nom_arch_aju,                                 { Nombre del archivo de ajustes. }
  nom_arch_pag,                                 { Nombre del archivo de pagos. }
  nom_arch_cob,                                 { Nombre del archivo de cobros. }
  nom_arch_inv_mp,                              { Nombre del archivo de materias primas. }
  nom_arch_inv_pt,                              { Nombre del archivo de productos terminados. }
  nom_arch_prod,                                { Nombre del archivo de producción. }
  nom_arch_maq,                                 { Nombre del archivo de maquila. }
  nom_arch_cont: Nom;                           { Nombre del archivo de contadores. }

  { para el uso directo del programa base. }
  opcion: TTecla;                               { Almacenamiento de la tecla oprimida. }

  { Directivos al compilador. }
{$V-}                                           { No verificar tamaño de las cadenas. }

  { Grupo de ficheros incluidos. }
{$I INCLUIDO\VARIOS.INC}      { Contiene los primeros procedimientos y funciones de propósito general. }

{$I INCLUIDO\PREG.INC}        { Procedimientos y funciones para la entrada de datos por teclado. }

{$I INCLUIDO\ACCDBASE.INC}    { Procedimientos y funciones para el acceso a los archivos del dBase III. }

{$I INCLUIDO\VARIOS2.INC}     { Contiene los últimos procedimientos y funciones de propósito general. }

{$I INCLUIDO\CARGA_IN.OVL}    { Contiene: 'CargaInicial'. }

{$I INCLUIDO\CO.OVL}          { Sección inicial CONSULTAS: 1ª parte 'Consultas'. }
  {$I INCLUIDO\CI.INC}        { Sección CONSULTAS/INVENTARIOS: todo el procedimiento 'Inventarios'. }
  {$I INCLUIDO\CD.INC}        { Sección CONSULTAS/DESCRIPCIONES: todo el procedimiento 'Descripciones'. }
  {$I INCLUIDO\CR.INC}        { Sección CONSULTAS/REGISTROS: todo el procedimiento 'Registros'. }
{$I INCLUIDO\C1.OVL}          { Sección final CONSULTAS: 2ª parte 'Consultas'. }

{$I INCLUIDO\PO.OVL}          { Sección inicial PEDIDOS: 1ª parte 'Pedidos'. }
  {$I INCLUIDO\PA.INC}        { Sección PEDIDOS/ANTICIPO: todo el procedimiento 'Anticipo'. }
  {$I INCLUIDO\PP.INC}        { Sección PEDIDOS/PAGO TOTAL: todo el procedimiento 'PagoTotal'. }
  {$I INCLUIDO\PI.INC}        { Sección PEDIDOS/IMPRIMIR PEDIDO: todo el procedimiento 'ImprimirPedido'. }
```

```
{  ($I INCLUIDO\PF.INC)       { Sección PEDIDOS/FACTURAR: todo el procedimiento 'Facturar'. }
{  ($I INCLUIDO\PC.INC)       { Sección PEDIDOS/CANCELAR: todo el procedimiento 'Cancelar'. }
{($I INCLUIDO\PI.OVL)         { Sección final PEDIDOS: 2ª parte 'Pedidos'. }
{
{($I INCLUIDO\EO.OVL)         { Sección inicial ENTRADAS: 1ª parte 'Entradas'. }
{  ($I INCLUIDO\EC.INC)       { Sección ENTRADAS/NUEVAS/COMPRAS: todo el procedimiento 'Compras'. }
{  ($I INCLUIDO\EP.INC)       { Sección ENTRADAS/NUEVAS/MAQUILA: todo el procedimiento 'Maquila'. }
{  ($I INCLUIDO\EM.INC)       { Sección ENTRADAS/NUEVAS/PRODUCCION: todo el procedimiento 'Produccion'. }
{  ($I INCLUIDO\ED.INC)       { Sección ENTRADAS/NUEVAS/DEVOLUCION: todo el procedimiento 'Devolucion'. }
{($I INCLUIDO\EI.OVL)         { Sección final ENTRADAS: 2ª parte 'Entradas' y todo el procedimiento 'YaExistentes'. }
{
{($I INCLUIDO\SO.OVL)         { Sección inicial SALIDAS: 1ª parte 'Salidas'. }
{  ($I INCLUIDO\SV.INC)       { Sección SALIDAS/NUEVAS/VENTAS: todo el procedimiento 'Ventas'. }
{  ($I INCLUIDO\SP.INC)       { Sección SALIDAS/NUEVAS/PRODUCCION: todo el procedimiento 'Produccion'. }
{  ($I INCLUIDO\SM.INC)       { Sección SALIDAS/NUEVAS/MAQUILA: todo el procedimiento 'Maquila'. }
{($I INCLUIDO\SI.OVL)         { Sección final SALIDAS: 2ª parte 'Salidas' y todo el procedimiento 'YaExistentes'. }
{
{($I INCLUIDO\D.OVL)          { Sección DINERO: todo el procedimiento 'Dinero'. }
{
{($I INCLUIDO\A.OVL)          { Sección AJUSTES: todo el procedimiento 'Ajustes'. }
{
{($I INCLUIDO\R.OVL)          { Sección RESULTADOS: todo el procedimiento 'Resultados'. }
{
{($I INCLUIDO\U.OVL)          { Sección UTILERIAS: todo el procedimiento 'Utilerias'. }
{
```

```
BEGIN
  TextColor(VERDE);
  CargaInicial;                                   { Hace la carga inicial desde el disco. }
  Fechar;                                         { Lee la fecha del sistema. }
  REPEAT
    ClrScr;                                       { Presenta el encabezado del menú principal. }
    GotoXY(26, 1); Write('M E N U   P R I N C I P A L ');
    GotoXY(70, 1); Write(fecha);
    Posibilidad(30,  5, 'C O N S U L T A S');     { Presenta las posibilidades de elección. }
    Posibilidad(30,  7, 'P E D I D O S');
    Posibilidad(30,  9, 'E N T R A D A S');
    Posibilidad(30, 11, 'S A L I D A S');
    Posibilidad(30, 13, 'D I N E R O');
    Posibilidad(30, 15, 'A J U S T E S');
    Posibilidad(30, 17, 'R E S U L T A D O S');
    Posibilidad(30, 19, 'U T I L E R I A S');
    Posibilidad(30, 21, 'T E R M I N A R');
    GotoXY(60,23); Write('¿Qué elección? ');
    PregOpcion(opcion,'CPESDARUT',' ');           { Pide que se seleccione una de las opciones. }
    CASE opcion[1] OF                             { Ejecuta el procedimiento seleccionado. }
      'C' : Consultas;
      'P' : Pedidos;
      'E' : Entradas;
      'S' : Salidas;
      'D' : Dinero;
```

```
    'A' : Ajustes;
    'R' : Resultados;
    'U' : Utilerias;
    'T' : BEGIN                                          ( Pide que se confirme la elección de terminar. )
            GotoXY(1,23); ClrEol;
            GotoXY(22,23); Write('¿ SEGURO DE QUERER TERMINAR ? (S/N) ');
            PregOpcion(opcion,'SN',' ');
            IF opcion[1] = 'S' THEN opcion[1] := 'T';
          END;
  END;
UNTIL opcion[1] = 'T';                          ( Se repite hasta que se tome y ratifique la opción 'T'. )
Terminar;                                       ( Cierra los archivos abiertos. )
END.
```

Como se puede observar en el listado mostrado, se hace uso de procedimientos de solapamiento, lo cual se debe a que esta versión del Turbo Pascal tiene limitada la capacidad para código objeto a 64 *KBytes*, por lo que no es posible mantener simultáneamente en memoria RAM todas las secciones del programa.

Los procedimientos de solapamiento, que se presentarán y discutirán después de los que no lo sean, son los siguientes:

+ CargaInicial
+ Consultas
+ Pedidos
+ Entradas
    + Nuevas
    + YaExistentes
+ Salidas
    + Nuevas
    + YaExistentes
+ Ajustes
+ Dinero
+ Resultados
+ Utilerias

Hay procedimientos y funciones que aparecen repetidamente a lo largo del programa (ver sección 4.3.1) debido a la limitación en el tamaño del código objeto. El único procedimiento que no obedece a esta razón es OkImprimirForma, el cual llama al procedimiento CadenaExterna que en cada caso es diferente.

En el almacenamiento de datos también se tiene una limitante de 64 *KBytes*, lo cual se resuelve utilizando la memoria dinámica, haciendo uso de punteros.

El conjunto de sentencias del programa ejecuta primero el procedimiento de solapamiento CargaInicial para que los datos sean leídos del disco. Ya con los datos en la memoria del

computador, se despliega el menú principal del programa, en el cual se presentan las diferentes posibilidades de elección. Cuando se elige alguna de ellas, se ejecuta el procedimiento de solapamiento seleccionado y cuando termina la ejecución del procedimiento seleccionado, vuelve a presentar el menú principal, a excepción de la opción "Terminar", en cuyo caso terminará la ejecución del programa.

4.3.3 PROCEDIMIENTOS Y FUNCIONES DE USO GENRALIZADO.

Estos ficheros contienen los procedimientos y funciones de propósito general. Estos permanecerán en todo momento en la memoria RAM del computador y pueden ser utilizados desde cualquier parte del programa. Dichos procedimientos cumplen con tareas que presentan alta incidencia en su utilización, por lo que es adecuado que permanezcan en memoria.

El espacio utilizado en memoria por el código objeto de estas rutinas resulta ser bastante grande. De aquí que los procedimientos y funciones que les sigan serán declarados como de solapamiento.

PRIMEROS PROCEDIMIENTOS DE PROPOSITO GENERAL.

En este fichero están contenidos la mayor parte de los procedimientos y funciones de propósito generalizado. Los que se presentarán son utilizados, a su vez, por otros procedimientos.

VARIOS.INC (Nombre del fichero que contiene el código fuente que se muestra.)

```
PROCEDURE Alarma(veces: Integer);              ( Procedimiento que hace sonar una alarma el número de 'veces' deseado. )

  CONST
    PASO = 50;                                 ( Paso del contador de tonos. )

  VAR
    cont1, cont2, tono: Integer;               ( Contadores y el tono a ejecutar. )

  BEGIN
    FOR cont1 := 1 TO veces DO                 ( Ejecuta la rutina 'veces' número de veces. )
      BEGIN
        tono := 1500;                          ( Se inicia con el tono número 1,500. )
        FOR cont2 := 0 TO 20 DO                ( Hace sonar una serie de veinte tonos aumentados por 'PASO' cada vez y )
          BEGIN                                ( con una duración de siete milisegundos cada uno. )
            Sound(tono); Delay(7);
            tono := tono + PASO;
          END;
        FOR cont2 := 0 TO 20 DO                ( Hace sonar una serie de veinte tonos disminuidos por 'PASO' cada vez y )
          BEGIN                                ( con una duración de siete milisegundos cada uno. )
            Sound(tono); Delay(7);
            tono := tono  - PASO;
          END;
      END;
    Nosound;
  END;
```

```
PROCEDURE Timbre;                          { Hace sonar un timbre. }

  BEGIN
    Sound(150); Delay(200); NoSound;       { Ejecuta el tono ciento cincuenta durante doscientos milisegundos. }
  END;
```

```
PROCEDURE MensajeErr(letrero: Cad78);      { Presenta en la última línea de la pantalla un mensaje de error haciendo }
                                           { sonar el timbre. }
  VAR
  x, y: Byte;                              { Localización del cursor. }
  tecla: Char;                             { Primer carácter de la tecla presionada. }

  BEGIN
  x := WhereX; y := WhereY;
  GotoXY(1,24); Write(espacios); GotoXY(1,24);          { Borra la última línea de la pantalla. }
  letrero := Copy(letrero,1,68);                        { Sólo permite los primeros sesenta y ocho caracteres. }
  Timbre;                                               { Hace sonar el timbre. }
  Write(letrero,' <Enter>');
  REPEAT                                                { Presenta el letrero y espera hasta que se apriete la }
    GotoXY(Length(letrero) + 9,24); Read(Kbd,tecla);    { tecla "RETURN". }
  UNTIL tecla = #13;
  GotoXY(1,24); Write(espacios);                        { Borra la última línea de la pantalla. }
  GotoXY(x,y);                                          { Regresa el cursor a su posición original. }
  END;
```

```
PROCEDURE Fechar;                          { Procedimiento utilizado para depositar en dos variables cadena }
                                           { la fecha y hora del reloj interno de la computadora. }
  TYPE                                     { Este procedimiento se desarrolló a partir de otro muy similar que }
    Cad2 = String[2];                      { aparece en el disco "Turbo Tutor" de esta versión del Turbo Pascal }
                                           { desarrollado por Borland International, Inc. }
  VAR
    regs: RECORD CASE Integer OF                    { Solamente para MS-DOS }
                  1: (AX,BX,CX,DX,BP,DI,SI,DS,ES,Flags: Integer);
                  2: (AL,AH,BL,BH,CL,CH,DL,DH: Byte);
                END;
```

```
  FUNCTION CambioTipo(valor: Byte): Cad2;  { Convierte un número de dos cifras tipo byte a una cadena de dos carac. }

    BEGIN
      Str(valor:2,una_cad);
      IF una_cad[1] = ' ' THEN una_cad[1] := '0';
      CambioTipo := una_cad;
    END;
```

```
  BEGIN
```

```
  WITH regs DO                          { Hace un llamado al sistema operativo para obtener la hora del sistema. }
    BEGIN                               { Esta se coloca en la variable 'horas' (que es general) en la forma }
      AH := $2C;                        { "hora:minutos meridiano". }
      Flags := 0;
      MsDos(regs);
      meridiano := 'am';
      IF CH > 11 THEN
        BEGIN
          CH := CH - 12;
          meridiano := 'pm';
        END;
      IF CH = 0 THEN CH := 12;
      hora := CH;
      minuto := CL;
      Str(hora:2,horas);
      horas := horas + ':' + CambioTipo(minuto) + meridiano;

      AH:=$2A;                          { Hace un llamado al sistema operativo para obtener la fecha del sistema. }
      Flags:=0;                         { Esta se coloca en la variable 'fecha' (que es general) en la forma }
      MsDos(regs);                      { "dia/mes/año". }
      dia := DL;
      mes := DH;
      ano := CX Mod 100;
      fecha := CambioTipo(dia) + '/' + CambioTipo(mes) + '/' + CambioTipo(ano);
    END;
END;
```

```
FUNCTION JusCad(entr: Cad78;            { Obtiene una cadena de longitud 'long' justificando a la izquierda }
                long: Byte): Cad78;     { el contenido de la cadena de entrada 'entr'. }
                                        { Ej: JusCad('Las Palmas #351',20) -> "Las Palmas #351     ". }
  VAR
    cad: Cad78;                         { Cadena de trabajo. }
    len: Byte;                          { Longitud de la cadena a acotar. }

  BEGIN
    len := Length(entr);
    IF len = long                       { Si la cadena 'entr' no tiene la longitud deseada, se acotará por la }
      THEN cad := entr                  { derecha si es mayor, o se le agregarán espacios en blanco si es menor. }
      ELSE
        IF len > long
          THEN cad := Copy(entr,1,long)
          ELSE cad := entr + Copy(espacios,1,long - len);
    JusCad := cad;
  END;
```

```
FUNCTION JusEnte(entr: Integer;                  { Obtiene una cadena de longitud 'long' justificando a la derecha y }
                 long: Byte): Cad78;             { dando formato al entero de entrada 'entr'. }
                                                 { Ej: JusEnte(7421,5) -> "7,421". }
  VAR
    cad: Cad78;                                  { Cadena de trabajo. }
    len,                                         { Longitud del número a justificar. }
    des: Byte;                                   { Número de comas a insertar para la justificación. }

  BEGIN
    Str(entr,cad);
    len := Length(cad);
    des := (len - Ord(entr < 0) - 1) DIV 3;
    IF len + des > long                          { Si la longitud del número ya formateado sobrepasa la longitud máxima }
      THEN cad := Copy(diagonales,1,long)        { se obtendrán diagonales para indicarlo. En caso contrario se procede a }
      ELSE                                       { formatear el número con una coma para los miles y el signo de menos }
        BEGIN                                    { inmediatamente antes del primer número para negativos. }
          IF len - Ord(entr < 0) > 3 THEN
            BEGIN
              Insert(',',cad,len - 2);
              len := len + 1;
            END;
          IF len < long THEN cad := Copy(espacios,1,long - len) + cad;
        END;
    JusEnte := cad;
  END;
```

```
FUNCTION JusReal(entr: Real;                     { Obtiene una cadena de longitud 'long' justificando a la derecha y }
                 long,                           { dando formato a 'dec' número de decimales al real de entrada 'entr'. }
                 dec: Byte): Cad78;              { Ej: JusReal(1345232.2,14,2) -> "  1'345,232.20". }

  VAR
    cad: Cad78;                                  { Cadena de trabajo. }
    len,                                         { Longitud del real a justificar. }
    posi,                                        { Posición del punto. }
    num_ent,                                     { Número de posiciones enteras requeridas. }
    des:       Byte;                             { Lugares de desplazamiento necesarios para la justificación. }

  BEGIN
    Str(entr:0:dec,cad);
    len := Length(cad);
    des := (len - Ord(entr < 0) - Ord(dec>0) * (dec + 1) - 1) DIV 3;
    posi := Pos('.',cad);
    IF dec > 0                                   { Evalúa el número de posiciones enteras a utilizar. }
      THEN num_ent := posi - Ord(entr < 0) - 1
      ELSE num_ent := len - Ord(entr < 0);
    IF len + des > long                          { Si la longitud del número ya formateado sobrepasa la longitud máxima }
      THEN cad := Copy(diagonales,1,long)        { se obtendrán diagonales para indicarlo. En caso contrario se procede a }
      ELSE                                       { formatear el número con una comilla para millones, una coma para los }
        BEGIN                                    { miles y el signo de menos inmediatamente antes del primer número para }
```

```
        IF num_ent > 3 THEN                                    { los negativos. }
          Insert(',',cad,len - Ord(dec > 0) * (dec + 1) - 2);
        IF num_ent > 6 THEN Insert(COMILLA,cad,len - dec - 6);
        len := Length(cad);
        IF len < long THEN cad := Copy(espacios,1,long - len) + cad;
      END;
    JusReal := cad;
  END;
```

```
FUNCTION JusFolio(entr: Integer): Cad78;        { Obtiene una cadena en donde se da el formato de folio al entero 'entr'. }
                                                { Ej: JusFolio(13) -> "0013". }
  VAR
    cadena: Cad78;                              { Cadena de trabajo. }

  BEGIN
    Str(entr,cadena);
    WHILE Length(cadena) < 4 DO                 { Cada vez que la longitud de la cadena que contiene el número enviado sea }
      Insert('0',cadena,1);                     { menor a cuatro, se insertará un cero en la 1ª posición de la misma. }
    JusFolio := cadena;
  END;
```

```
FUNCTION SigFolio(entr: Integer): Integer;      { Función que proporciona el folio siguiente al enviado. }

  BEGIN
    entr := Succ(entr);                         { Se obtiene el número consecutivo del recibido; si el número obtenido }
    IF entr > 9999 THEN entr := 1;              { sobrepasa los cuatro dígitos, se vuelve a iniciar el conteo desde uno. }
    SigFolio := entr;
  END;
```

```
FUNCTION AntFolio(entr: Integer): Integer;      { Función que calcula el folio anterior al enviado. }

  BEGIN
    entr := Pred(entr);                         { Se obtiene el número que precede al enviado; si el número obtenido }
    IF entr < 1 THEN entr := 9999;              { es menor que uno, se convierte en el folio más grande a obtener. }
    AntFolio := entr;
  END;
```

```
FUNCTION OkFolio(   num: Integer;               { Función booleana que indica si el folio en cuestión ('num') se }
                  desde: Integer;               { encuentra entre el primero ('desde') y el último ('hasta') folios. }
                  hasta: Integer   ): Boolean;

  BEGIN
    IF num > 0                                  { El folio enviado tiene que ser distinto de cero. Si el folio 'desde' }
      THEN                                      { es mayor que 'hasta', se incrementará este último folio en 9999. Y si }
        BEGIN                                   { al mismo tiempo 'num' es menor que 'desde', se incrementará el primero }
```

```
        IF desde > hasta THEN                ( en 9999. Ya hechos estos ajustes, el folio estará correcto si se )
          BEGIN                              ( encuentra en el intervalo cerrado de 'desde' a 'hasta'. )
            hasta := hasta + 9999;
            IF num < desde THEN num := num + 9999;
          END;
        OkFolio := (num >= desde) AND  (num <= hasta);
      END
    ELSE
      OkFolio := FALSE;
  END;
```

```
PROCEDURE Solapar(     externa: Cad255;      ( Procedimiento que solapa completamente la cadena 'externa' en la cadena )
                  VAR  cadena: Cad255;       ( 'cadena' a partir de la posición 'pos'. )
                       posi: Integer );
  VAR
    cont,                                    ( Contador del carácter a solapar. )
    long,                                    ( Longitud de la cadena 'externa'. )
    tam_min: Byte;                           ( Tamaño mínimo que debe tener la cadena 'cadena' para poder ser solapada. )

  BEGIN
    long := Length(externa);
    IF (long > 0) AND (posi > 0) THEN        ( Si 'externa' contiene caracteres y 'pos' es válido se puede hacer el )
      BEGIN                                  ( solapamiento. El tamaño mínimo para 'cadena' será la longitud de )
        tam_min := long + posi - 1;          ( 'externa' más el número de caracteres de 'cadena' a la izquierda del )
        WHILE Length(cadena) < tam_min DO    ( solapamiento. De ser necesario, se ajusta 'cadena' a dicha longitud y )
          cadena := cadena + ' ';            ( después se lleva a cabo el solapamiento. )
        FOR cont := 1 TO long DO
          cadena[posi + cont - 1] := externa[cont];
      END;
  END;
```

```
PROCEDURE Posibilidad(   x, y: Byte;       ( Presenta en un lugar de la pantalla ('x','y') una posibilidad de )
                       cadena: Cad78 );    ( elección ('cadena'), recalcando su primera letra. )
  BEGIN
    GotoXY(x,y);                                          ( Ubica el cursor en la posición indicada, )
    TextColor(VERDE_CLARO); Write(cadena[1]); Delete(cadena,1,1);   ( se presenta en color claro el primer )
    TextColor(VERDE); Write(cadena);                      ( carácter del letrero enviado. Elimina ese )
  END;                                                    ( carácter y presenta el resto del letrero. )
```

Alarma; que hace sonar una alarma a manera de sirena. El argumento que requiere es un entero positivo, que indica el número de veces que va a subir y bajar el tono del altavoz. Este procedimiento es ejecutado cuando se intenta violar la llave del sistema.

Timbre; ejecuta un tono grave constante durante aproximadamente 200 milisegundos. Este procedimiento es ejecutado cuando se presenta un mensaje de error.

MensajeErr; presenta en pantalla la cadena enviada como argumento en la última línea de visualización y hace sonar el timbre. Para que el mensaje desaparezca de la pantalla y el proceso continúe, es necesario presionar la tecla de retorno de carro en señal de enterado.

Fechar; procedimiento que introduce en las variables generales tipo cadena fecha y horas, la fecha y hora del sistema, respectivamente. Este procedimiento fue una adaptación del código fuente del programa DATE.PAS contenido en el disco "Turbo Tutor" desarrollado por Borland International Inc. Este procedimiento se ejecuta al principio del programa.

CambioTipo; procedimiento que pertenece al procedimiento Fechar, y cuya función es únicamente la de convertir una variable tipo *byte* de dos cifras a una cadena de dos caracteres. Esta conversión se requiere para pasar a una variable cadena los minutos y el mes que se reciben del sistema operativo.

JusCad; función tipo cadena que obtiene la justificación de una cadena a una longitud dada. El propósito de esta función es obtener una cadena que contenga estrictamente el número de caracteres deseados. Para su utilización se coloca como primer parámetro la cadena que se va a justificar y como segundo un número que indique la longitud de justificación deseada.

```
Ej: JusCad('123456789  ',5);  --> "12345".
Ej: JusCad('123456789  ',15); --> "123456789      ".
```

JusEnte; función tipo cadena que justifica un entero (o *byte*) a la derecha y a un longitud dada, colocando una coma para separar los miles. Para utilizarlo se coloca como primer parámetro el entero a justificar y como segundo parámetro un número que indique la longitud de justificación. La longitud solicitada podrá ser mayor a la mínima necesaria, pero si es menor, se presentarán únicamente líneas diagonales para indicarlo.

```
Ej: JusEnte(-22312,10); --> "   -22,312".
Ej: JusEnte(-22312,6);  --> "//////".
```

JusReal; función tipo cadena que justifica un número tipo real a la derecha a una longitud dada, colocando una coma para los miles y una comilla para los millones. Para utilizar este procedimiento, se colocan como parámetros: el número tipo real a justificar, un número que indique la

longitud de justificación deseada y, por último, un número que indique las posiciones decimales a utilizar.
Ej: JusReal(78198343,15,2); --> " 78'198,343.00".

JusFolio; función tipo cadena que da la presentación de folio (de cuatro dígitos) a un número tipo entero. Para utilizar dicha función, se envía como único parámetro el entero a presentar como folio.
Ej: JusFolio(34); --> "0034".

SigFolio; función tipo entera que calcula el folio siguiente al enviado como parámetro. El único parámetro necesario para utilizar dicha función es el número de folio del cual se quiere obtener el folio consecutivo.
Ej: SigFolio(2671); --> 2672.
Ej: SigFolio(9999); --> 1.

AntFolio; función tipo entera que calcula el folio anterior al enviado como único parámetro.
Ej: AntFolio(391); --> 390.
Ej: AntFolio(1); --> 9999.

OkFolio; función tipo *booleana* que verifica que un folio se encuentre en el intervalo cerrado de otros dos folios. Para utilizar la función se requiere de tres parámetros; el folio del que se tiene la duda, el inicial y el final.
Ej: OkFolio(500,1,100); --> FALSE.
Ej: OkFolio(50,9000,100); --> TRUE.

Solapar; procedimiento que solapa (sobrepone) una cadena en otra a partir de cierta posición. Los parámetros necesarios son: la cadena fuente, la cadena destino y la posición inicial.
Ej: Solapar('Fuente','Destino',10);
--> "Destino Fuente".

Posibilidad; procedimiento que presenta en pantalla una cadena en la posición de la pantalla que se desee. El primer carácter de la cadena aparecerá con un color claro para indicar que con dicho carácter se puede seleccionar esa opción. Se envian como parámetros; la localización horizontal y vertical, y la cadena a presentar.
Ej: Presentar en la posición horizontal dos y vertical treinta de la pantalla respectivamente el letrero "PEDIDOS".
Posibilidad(2,30,'PEDIDOS');

SUBPROGRAMA DE ENTRADA DE DATOS POR TECLADO.

Este grupo de procedimientos y funciones se utiliza para la entrada general de datos por teclado. En seguida se expone el fichero del código fuente que contiene este subprograma.

PREG.INC (Nombre del fichero que contiene el código fuente que se muestra.)

```
PROCEDURE Editar(VAR entrada  : Cad78;          { Procedimiento que edita las entradas por teclado. En donde el parámetro }
                 tipo         : Tipos;          { var 'entrada' contiene la entrada por defecto, 'tipo' es el tipo de }
                 longitud,                      { variable, 'longitud' es la longitud de la entrada, 'decimales' es el }
                 decimales    : Byte;           { número de decimales esperadas y 'negativo' se utiliza para aceptar }
                 negativo     : Boolean );      { negativos. }

  VAR
    entrada_nueva: Cad78;                       { Para nuevas entradas por teclado. }
    contador,                                   { Contador para la posición del cursor en la cadena de entrada. }
    pos_x,                                      { Posición del cursor en el eje de las x's. }
    pos_y,                                      { Posición del cursor en el eje de las y's. }
    loc_punto: Byte;                            { Localización a lo largo de la entrada del punto decimal. }
    caracter,                                   { Código ASCII del carácter introducido por el teclado. }
    funcion: Char;                              { Código de la tecla de función utilizada. }
    ok,                                         { Variable booleana para identificación de errores. }
    insertar,                                   { Verdadero cuando se está en modo insertar. }
    especial,                                   { Verdadero cuando se espera un carácter especial Ej: á. }
    signo: Boolean;                             { Verdadero si la cifra introducida es negativa. }
    numero: Real;                               { Para verificar que el número introducido sea correcto. }

  BEGIN
    pos_x := WhereX; pos_y := WhereY;           { Memoriza la posición del cursor. }
    IF longitud + pos_x > MAX_LONG THEN         { Revisa que la longitud de la entrada no sobrepase la pantalla. }
      BEGIN
        longitud := MAX_LONG + 1 - pos_x;
        MensajeErr('La entrada sobrepasa la pantalla. Será acotada.');
      END;
    IF Length(entrada) > longitud THEN                      { Verifica la longitud de la entrada. }
      entrada := Copy(entrada, 1, longitud);
    caracter := ESC; funcion := #227;                       { Da valor inicial a las variables. }
    insertar := FALSE; signo := FALSE;
    especial := FALSE;
    IF (tipo = T_REAL) OR (tipo = T_ENTERO) THEN            { Para las entradas de variables tipo real y entera: }
      IF Pos('-',entrada) > 0 THEN                          { En caso de encontrar un signo negativo en la cadena: }
        BEGIN
          Delete(entrada,1,1);                              { Elimina el signo de la cadena. }
          IF negativo THEN                                  { En caso de que se permitan negativos, se presentará }
            BEGIN                                           { el signo en la pantalla. }
              signo := TRUE;
              GotoXY(pos_x - 1,pos_y); Write('-');
            END;
        END;
```

```
  REPEAT
    ok := TRUE;                          { Da valor inicial a la variable booleana 'ok' como verdadero. }
    IF caracter = ESC
      THEN                               { En caso de uso de funciones. }
        CASE funcion OF
          #227   : BEGIN                 { 'Esc' recupera la entrada original. }
                     entrada_nueva := entrada;
                     GotoXY(pos_x,pos_y); Write(Copy(lineas,1,longitud));
                     GotoXY(pos_x,pos_y); Write(entrada_nueva);
                     contador := 1;
                   END;
          #71    : contador := 1;                                  { 'Home' Cursor al principio de la entrada. }
          #75    : IF contador > 1 THEN contador := Pred(contador); { '<-' Cursor a la izquierda. }
          #77    : IF contador < Length(entrada_nueva) + 1 THEN     { '->' Cursor a la derecha. }
                     contador := Succ(contador);
          #79    : contador := Length(entrada_nueva) + 1;          { 'End' Cursor al final. }
          #82    : BEGIN                                           { 'Ins' Bandera de inserción de caracteres. }
                     insertar := NOT(insertar);
                     GotoXY(pos_x + longitud,pos_y);
                     IF insertar
                       THEN Write(#26)
                       ELSE Write(ESPACIO);
                   END;
          #83    : IF contador <= Length(entrada_nueva) THEN       { 'Del' Borrar carácter sobre el cursor. }
                     BEGIN
                       Delete(entrada_nueva,contador,1);
                       Write(Copy(entrada_nueva,contador,longitud));
                       Write(LINEA);
                     END;
          #208   : IF contador > 1 THEN                            { '<-' Borrar carácter a la izq. del cursor. }
                     BEGIN
                       contador := Pred(contador);
                       Delete(entrada_nueva,contador,1);
                       GotoXY(pos_x + contador - 1,pos_y);
                       Write(Copy(entrada_nueva,contador,longitud));
                       Write(LINEA);
                     END;
          #255   : BEGIN                                           { '^Del' Borra toda la entrada. }
                     contador := 1; entrada_nueva := '';
                     GotoXY(pos_x,pos_y); Write(Copy(lineas,1,longitud));
                     GotoXY(pos_x,pos_y);
                   END;
          ENTER  : CASE Tipo OF                                    { 'Enter' Terminación de la entrada. }
                                                                   { Hace un análisis para autorizar cada tipo }
                     T_BYTE    : BEGIN                           { de entrada. }
                                   IF Length(entrada_nueva) = 0 THEN
                                     entrada_nueva := '0';
                                   VAL(entrada_nueva,numero,num_err);
                                   IF (numero > 255) THEN
                                     BEGIN
                                       ok := FALSE;
                                       MensajeErr('No se acepta un valor mayor a 255.');
```

```
                        END;
                    END;

            T_ENTERO :  BEGIN
                        IF Length(entrada_nueva) = 0 THEN entrada_nueva := '0';
                        IF signo THEN Insert('-',entrada_nueva,1);
                        VAL(entrada_nueva,numero,num_err);
                        IF (numero < MININT) OR (numero > MAXINT) THEN
                          BEGIN
                            ok := FALSE;
                            MensajeErr('Valor fuera del rango -32767..32767.');
                          END;
                        END;

            T_REAL   :  BEGIN
                        IF Length(entrada_nueva) = 0 THEN entrada_nueva := '0';
                        IF signo THEN Insert('-',entrada_nueva,1);
                        VAL(entrada_nueva,numero,num_err);
                        Str(numero:0:decimales,entrada_nueva);
                        END;

            T_CAR,
            T_CAD,
            T_CAD_MAY : BEGIN
                        WHILE entrada_nueva[Length(entrada_nueva)] = ESPACIO DO
                        Delete(entrada_nueva,Length(entrada_nueva),1);
                        END;

          END;
    END
  ELSE                                                  { En caso de caracteres de impresión. }
    IF contador > longitud
      THEN ok := FALSE                  { Si ya no cabe el carácter introducido, se hace falsa la variable 'ok'. }
      ELSE                              { Si si cabe el carácter introducido: }
        BEGIN
          CASE Tipo OF                                  { Verifica la validez del carácter introducido. }

            T_BYTE   : IF NOT(caracter IN [#48..#57]) THEN ok := FALSE;          { Si no es un número no se acepta. }

            T_ENTERO : IF NOT(caracter IN [#48..#57]) THEN                       { Si no es un número: }
                       BEGIN
                         ok := FALSE;                                            { No se acepta en la cadena. }
                         IF (caracter = '-') AND (negativo = TRUE) THEN          { Si es el signo negativo y si }
                           BEGIN                                                 { se aceptan negativos: }
                             signo := NOT(signo);
                             GotoXY(pos_x - 1,pos_y);                            { Coloca el cursor en posición. }
                             IF signo                   { Si la variable signo es: }
                               THEN Write('-')          { Verdadera; presenta el signo "-" en la pantalla. }
                               ELSE Write(espacio);     { Falsa; presenta un espacio en blanco en la pantalla. }
                           END;
                       END;
```

```
T_REAL   : IF (caracter = '-') AND (negativo = TRUE)
             THEN                              { Si es el carácter negativo y se aceptan negativos: }
               BEGIN
                 ok := FALSE;                  { No se acepta en la cadena. }
                 signo := NOT(signo);
                 GotoXY(pos_x - 1,pos_y);      { Coloca el cursor en posición. }
                 IF signo                      { Si la variable signo es: }
                   THEN Write('-')             { Verdadera; presenta el signo "-" en la pantalla. }
                   ELSE Write(espacio);        { Falsa; presenta un espacio en blanco en la pantalla. }
               END
             ELSE                              { No es el carácter negativo y/o no se aceptan neg. }
               IF decimales > 0
                 THEN                          { Si la entrada es con decimales: }
                   BEGIN
                     loc_punto := Pos('.',entrada_nueva);
                     IF caracter IN [#48..#57]
                       THEN                    { Si el carácter es un número: }
                         BEGIN
                           IF (contador + decimales = longitud)      { No aceptará el carácter }
                              AND ((loc_punto = 0)                   { numérico en caso de que }
                                   OR (loc_punto = contador))        { sobrepase el máximo de }
                             THEN ok := FALSE;                       { posiciones enteras. }
                           IF loc_punto = 0 THEN
                             loc_punto := Length(entrada_nueva) + 1;
                           IF insertar AND                           { No insertará el carácter }
                              (loc_punto + decimales = longitud ) AND { numérico si sobrepasa el }
                              (contador <= loc_punto)                 { máximo de posiciones enteras. }
                             THEN ok := FALSE;
                         END
                       ELSE                                          { El carácter no es numérico: }
                         IF NOT((caracter = '.') AND                 { Lo aceptará sólo en el caso }
                            ((loc_punto = 0)                         { de que sea el punto decimal }
                              OR (loc_punto = contador)))            { y que éste no esté ya antes }
                           THEN ok := FALSE;                         { puesto. }
                   END
                 ELSE                                                { Si la entrada es sin decimales: }
                   IF NOT(caracter IN [#48..#57]) THEN               { No lo acepta si no es un número. }
                     ok := FALSE;

T_CAD_MAY : IF caracter IN [#32..#255]                               { Si es cualquier carácter alfa- }
              THEN                                                   { numérico: }
                BEGIN
                  IF (caracter = ESPACIO) AND (contador = 1)         { No lo acepta si es un espacio al }
                    THEN ok := FALSE;                                { principio de la entrada. }
                  caracter := UpCase(caracter);                      { Conversión a mayúscula }
                  IF caracter = COMILLA
                    THEN                                             { Si es el carácter 'comilla': }
                      IF especial                                    { En espera de carácter especial: }
                        THEN especial := FALSE                       { Verdadero -> Se cancela }
                        ELSE                                         { Falso: }
                          BEGIN
                            especial := TRUE;                        { Esperará el carácter especial. }
```

```
                        ok := FALSE;                        ( No entra a la cadena. )
                       END
                    ELSE                                    ( Si no es la 'comilla'; )
                      IF especial THEN                      ( Se espera un carácter especial: )
                        BEGIN
                          CASE caracter OF                  ( Conversión para caracteres del )
                            'N' : caracter := 'Ñ';          ( español. )
                            '?' : caracter := '¿';
                            '!' : caracter := '¡';
                            'U' : caracter := 'Ü';
                          END;
                          especial := FALSE;                ( Cancela la espera de carac. esp. )
                        END;
                  END
                 ELSE ok := FALSE;                      ( No se aceptan caracteres del 00 al 031. )

    T_CAR,
    T_CAD    : IF caracter IN [032..0255]
                 THEN                                   ( Si es cualquier carácter alfanumérico: )
                  BEGIN
                    IF (caracter = ESPACIO) AND (contador = 1)      ( No lo acepta si es un espacio )
                     THEN ok := FALSE;                  ( al principio de la entrada. )
                    IF caracter = COMILLA               ( Si es el carácter 'comilla' )
                     THEN
                       IF especial                      ( En espera de carácter especial: )
                         THEN especial := FALSE         ( Verdadero-) Se cancela )
                         ELSE                           ( Falso: )
                           BEGIN
                             especial := TRUE;          ( Esperará el carácter especial. )
                             ok := FALSE;               ( No entra a la cadena. )
                           END
                     ELSE                               ( Si no es la 'comilla': )
                       IF especial THEN                 ( Se espera un carácter especial: )
                         BEGIN
                           CASE caracter OF             ( Conversión para caracteres del )
                             'a' : caracter := 'á';     ( español. )
                             'e' : caracter := 'é';
                             'i' : caracter := 'í';
                             'o' : caracter := 'ó';
                             'u' : caracter := 'ú';
                             'n' : caracter := 'ñ';
                             'N' : caracter := 'Ñ';
                             '?' : caracter := '¿';
                             '!' : caracter := '¡';
                             'v' : caracter := 'ü';
                             'U' : caracter := 'Ü';
                           END;
                           especial := FALSE;           ( Cancela la espera de carac. esp. )
                         END;
                  END
                 ELSE ok := FALSE;                      ( No se aceptan caracteres del 00 al 031. )
    END;
```

```
            IF ok THEN                                    { Si se aceptó como parte de la cadena de entrada: }
              BEGIN
                Write(caracter);                          { Presenta el carácter en la pantalla. }
                IF insertar
                  THEN                                    { Si está en insertar: }
                    BEGIN
                      Delete(entrada_nueva,longitud,1);           { Borra el carácter de la posición longitud. }
                      Insert(caracter,entrada_nueva,contador);    { Inserta el carácter en la posición actual. }
                      Write(Copy(entrada_nueva,contador + 1,longitud));  { Presenta en pantalla los caracteres ya }
                    END                                           { corridos posteriores al cursor. }
                  ELSE                                    { Si no está en insertar: }
                    IF contador > Length(entrada_nueva)
                      THEN entrada_nueva := entrada_nueva + caracter   { Si está en la última posición lo agrega. }
                      ELSE entrada_nueva[contador] := caracter;        { Si no está en la última pos. lo solapa. }
                contador := Succ(contador);               { Se incrementa el contador en uno. }
              END;
            END;

      GotoXY(pos_x + contador - 1, pos_y);      { Coloca el cursor en la nueva posición. }
      IF NOT((funcion = ENTER) AND ok) THEN     { Si no se ha aceptado la entrada: }
        BEGIN                                   { Sección de lectura del teclado. }
          ok := FALSE;
          funcion := #0;
          Read(Kbd, caracter);                          { Lee directo del teclado. }
          IF (KeyPressed AND (caracter = ESC))
            THEN Read(Kbd,funcion)                      { Si fue una tecla de función la lee. }
            ELSE                                        { Si no fue una tecla de función: }
              IF caracter IN [#1..#31] THEN             { Si es un carácter de control: }
                BEGIN
                  funcion := CHR(ORD(caracter) + 200);  { Lo pasa a función corriéndola 200 caracteres. }
                  caracter := ESC;                      { Indica que es carácter de función. }
                END;
        END;
      UNTIL (funcion = ENTER) AND ok;           { Se repite hasta que se presione la tecla ENTER y se acepte. }
    entrada := entrada_nueva;                   { Se actualiza la variable entrada con la nueva entrada. }
    GotoXY(pos_x,pos_y);                        { Coloca el cursor en la posición de envío. }
    Write(Copy(espacios,1,longitud + 1));       { Borra el espacio utilizado para la entrada. }
    GotoXY(pos_x,pos_y);                        { Coloca el cursor en la posición de envío. }
  END;
```

```
PROCEDURE PregByte(VAR ent_byte: Byte;          { Edición de entrada de una variable tipo byte. El parámetro var }
                   longitud    : Byte );        { 'ent_byte' recibe la entrada por defecto y envía la entrada hecha }
  VAR                                           { de longitud máxima 'longitud'. }
    cadena: Cad78;                              { Cadena para poder utilizar el procedimiento 'Editar'. }

  BEGIN
    IF longitud < 1 THEN                        { Verifica que la longitud especificada sea al menos de uno. }
      BEGIN
        longitud := 1;
        MensajeErr('Longitud menor que 1. Se fijará en 1');
      END;
```

```
  IF longitud > 3 THEN                    { Verifica que la longitud no sobrepase el máximo. }
    BEGIN
      longitud := 3;
      MensajeErr('Longitud para entrada byte fuera de rango. Se fijará en 3.');
    END;
  Str(ent_byte,cadena);
  Editar(cadena,T_BYTE,longitud,0,FALSE);  { Se hace la entrada por teclado con todas las especificaciones. }
  VAL(cadena,un_ente,num_err);
  ent_byte := un_ente;
  Write(ent_byte:longitud);               { Presenta en pantalla el número introducido. }
END;
```

```
PROCEDURE PregEnte(VAR ent_ente: Integer;     { Edición de entrada de una variable tipo entera. El parámetro var }
                   longitud    : Byte;        { 'ent_ente' recibe la entrada por defecto, envia la entrada hecha }
                   negativo    : Boolean );   { de longitud máxima 'longitud' y acepta negativos si 'negativo' es }
VAR                                           { verdadero. }
  cadena: Cad78;                              { Cadena para poder utilizar el procedimiento 'Editar'. }
  des: Byte;                                  { Indica el número de comas a insertar. }

BEGIN
  IF negativo THEN                            { En caso de que se acepten negativos, reserva un espacio para el signo. }
    BEGIN
      Write(' ');
      longitud := Pred(longitud);
    END;
  IF longitud < 1 THEN                        { Verifica que la longitud especificada sea al menos de uno. }
    BEGIN
      longitud := 1;
      MensajeErr('Longitud menor que 1. Se fijará en 1.');
    END;
  IF longitud > 3 THEN des := 1;              { Calcula el número de espacios a reservar. }
  IF longitud > 6 THEN                        { Verifica que no se exceda la longitud máxima. }
    BEGIN
      longitud := 6;
      MensajeErr('Longitud para variable entera fuera de rango. Se fijará en 6.');
    END;
  Str(ent_ente,cadena);                                  { Convierte la entrada por defecto en una cadena. }
  Write(Copy(espacios,1,longitud1));                     { Borra los espacios necesarios. }
  GotoXY(WhereX - longitud,WhereY);                      { Regresa el cursor a la posición original. }
  Editar(cadena,T_ENTERO,longitud - des,0,negativo);     { Se hace la entrada por teclado. }
  VAL(cadena,ent_ente,num_err);
  IF negativo THEN                                       { Si se aceptan negativos corre el cursor a la }
    BEGIN                                                { izquierda y restaura la longitud original. }
      Write(^H);
      longitud := Succ(longitud);
    END;
  Write(JusEnte(ent_ente,longitud));          { Presenta el número en la pantalla. }
END;
```

```
PROCEDURE PregReal(VAR ent_real: Real;          { Edición de entrada de una variable tipo real. El parámetro var }
                   longitud,                    { 'ent_real' recibe la entrada por defecto y envia la entrada hecha }
                   decimales   : Byte;          { de longitud máxima 'longitud', con 'decimales' número de decimales y }
                   negativo    : Boolean );     { acepta negativos si 'negativo' es verdadero. }
  VAR
    cadena: Cad78;                              { Cadena para poder utilizar el procedimiento 'Editar'. }
    des: Byte;                                  { Indica el número de cosas a insertar. }

  BEGIN
    IF negativo THEN                            { En caso de que se acepten negativos, reserva un espacio para el signo. }
      BEGIN
        Write(' ');
        longitud := Pred(longitud);
      END;
    IF longitud < 1 THEN                        { Verifica que la longitud especificada sea al menos de uno. }
      BEGIN
        longitud := 1;
        MensajeErr('Longitud menor que 1. Se fijará en 1.');
      END;
    des := (longitud - (decimales + 1) * Ord(decimales > 0)) DIV 4;          { Calcula el número de espacios a reservar. }
    IF longitud < decimales + 2 THEN                                         { Verifica que la longitud especificada }
      BEGIN                                                                  { pueda contener al menos una posición }
        longitud := decimales + 2;                                           { entera, el punto decimal y el número de }
        MensajeErr('Longitud menor que el núm. de decimales. Se fijará en ' +  { decimales especificado. }
                   JusEnte(longitud,2) + '.');
      END;
    IF ent_real = 0
      THEN cadena := '0'
      ELSE Str(ent_real:0:decimales,cadena);
    IF Length(cadena) > longitud THEN                                  { Si la cadena obtenida del valor }
      BEGIN                                                            { por defecto sobrepasa la }
        cadena := Copy(cadena,Length(cadena),Length(cadena) - longitud);  { longitud máxima será acotada }
        MensajeErr('Valor inicial mayor que la longitud especificada. Se truncará.');  { por la derecha. }
      END;
    Write(Copy(espacios,1,longitud));                                  { Borra los espacios necesarios. }
    GotoXY(WhereX - longitud,WhereY);
    Editar(cadena,T_REAL,longitud - des,decimales,negativo);           { Se hace la entrada por teclado. }
    VAL(cadena,ent_real,num_err);
    IF negativo THEN                                                   { Si se aceptan negativos ajusta }
      BEGIN                                                            { la posición del cursor un lugar }
        Write(^H);                                                     { a la izquierda y restaura la }
        longitud := Succ(longitud);                                    { longitud original. }
      END;
    Write(JusReal(ent_real,longitud,decimales));                       { Presenta el número en pantalla. }
  END;
```

```
PROCEDURE PregCar(VAR ent_car: Char);          { Edición de entrada de una variable tipo carácter. El parámetro var }
                                               { 'ent_car' recibe la entrada por defecto y envía la entrada hecha. }
  VAR
    cadena: Cad78;                             { Cadena para poder utilizar el procedimiento 'Editar'. }

  BEGIN
    cadena := ent_car;
    IF ent_car = #0                            { En caso de que sea el carácter #0 se substituye por una cadena vacía. }
      THEN cadena := '';
    Editar(cadena,T_CAR,1,0,FALSE);            { Se hace la entrada por teclado. }
    IF Length(cadena) = 1                      { Coloca la cadena en la variable de entrada. }
      THEN ent_car := cadena[1]
      ELSE ent_car := #32;
    Write(ent_car);                            { Presenta el carácter en la pantalla. }
  END;
```

```
PROCEDURE PregCadMay(VAR ent_cad: Cad78;       { Edición de entrada de una variable tipo cadena para mayúsculas. }
                     longitud   : Byte  );     { El parámetro var 'ent_car' recibe la entrada por defecto y envía }
                                               { la entrada hecha de longitud máxima 'longitud'. }
  BEGIN
    IF longitud > MAX_LONG THEN                { Si la longitud propuesta excede el máximo tolerable será acotada. }
      BEGIN
        longitud := MAX_LONG;
        MensajeErr('Longitud excedida. Se fijará en '+ JusEnte(MAX_LONG,3) + '.');
      END;
    Editar(ent_cad,T_CAD_MAY,longitud,0,FALSE);          { Se hace la entrada por teclado. }
    Write(ent_cad);                                      { Presenta la cadena en la pantalla }
  END;
```

```
PROCEDURE PregCad(VAR ent_cad: Cad78;          { Edición de entrada de una variable tipo cadena. El parámetro var }
                  longitud   : Byte  );        { 'ent_cad' recibe la entrada por defecto y envía la entrada hecha }
                                               { de longitud máxima 'longitud'. }
  BEGIN
    IF longitud > MAX_LONG THEN                { Si la longitud propuesta excede el máximo tolerable será acotada. }
      BEGIN
        longitud := MAX_LONG;
        MensajeErr('Longitud excedida. Se fijará en '+ JusEnte(MAX_LONG,3) + '.');
      END;
    Editar(ent_cad,T_CAD,longitud,0,FALSE);    { Se hace la entrada por teclado. }
    Write(ent_cad);                            { Presenta la cadena en la pantalla. }
  END;
```

```
PROCEDURE PregOpcion(VAR    opcion: TTecla;      { Entrada selectiva de una sola tecla. El parámetro var 'opcion' }
                     posib_car,                  { envia la tecla seleccionada, los posibles caracteres estarán }
                     posib_fun: Cad78 );         { contenidos en la cadena 'posib_car' y los códigos de las teclas de }
                                                 { función en la cadena 'posib_fun'. }
  VAR
    x, y: Byte;                                  { Coordenadas del cursor. }

  BEGIN
    REPEAT
      Write(' ',^H);                             { Borra un espacio en la posición del cursor y se regresa. }
      opcion[2] := #0;                           { Indica por defecto que no se introduce una tecla de función. }
      Read(Kbd, opcion[1]);                      { Lee directamente del teclado el primer carácter de la tecla utilizada. }
      IF (KeyPressed AND (opcion[1] = ESC)) THEN         { Si se ha introducido un segundo carácter y el primero es el }
        BEGIN                                            { carácter de escape: }
          Read(Kbd,opcion[2]);                    { Se lee el código de la función utilizada. }
          opcion[1] := #0;                        { El primer carácter se hace cero para indicar que es una función. }
        END;
      opcion[1] := UpCase(opcion[1]);            { Pasa el primer carácter a su respectiva mayúscula. }
      IF opcion[1] = #12 THEN                    { '^L' Coloca/Quita la llave. }
        BEGIN
          x := WhereX; y := WhereY;              { Memoriza la posición del cursor. }
          IF con_llave
            THEN                                 { Si estaba con llave, se tendrán dos oportunidades para introducirla }
              BEGIN                              { correctamente y así quitarla. De lo contrario no se quitará la llave, }
                un_ente := 0;                    { sonará una alarma y aparecerá un aviso de amenaza. }
                REPEAT
                  una_cad := '';
                  GotoXY(1,24); Write('Introduzca la clave llave ->');
                  un_ente := Succ(un_ente);
                  FOR un_byte := 1 TO 6 DO
                    BEGIN
                      Read(Kbd,un_car);
                      una_cad := una_cad + un_car;
                    END;
                  IF una_cad = LLAVE
                    THEN
                      BEGIN
                        con_llave := FALSE;
                        MensajeErr('Se quitó la llave.');
                      END
                    ELSE
                      IF un_ente = 2
                        THEN
                          BEGIN
                            Alarma(100);
                            MensajeErr('Está registrado el intento de violación.');
                          END
                        ELSE Timbre;
                UNTIL (un_ente = 2) OR NOT(con_llave);
              END
            ELSE                                             { Si no estaba con llave, se pone de nuevo y aparece }
```

```
            BEGIN                                          ( un mensaje para confirmar la operación. )
              con_llave := TRUE;
              MensajeErr('Está puesta la llave.');
            END;
          GotoXY(x,y);                                     ( Regresa el cursor a su posición original. )
        END;
    UNTIL (Pos(opcion[1],posib_car) <> 0) OR (Pos(opcion[2],posib_fun) <> 0);      ( Se repite hasta presionar una )
                                                                                   ( tecla correcta. )
    IF opcion[1] IN [#32..#254] THEN                                ( Si la tecla presionada es un carácter )
      BEGIN                                                         ( válido para la pantalla, se presentará )
        TextColor(VERDE_CLARO);                                     ( en color brillante. )
        Write(opcion[1]);
        TextColor(VERDE);
      END;
  END;
```

Se tienen dos diferentes formas de permitir la entrada de datos; la edición de una entrada propuesta y la entrada controlada de una sola tecla (carácter o función) para seleccionar alguna opción.

EDICION DE UNA ENTRADA PROPUESTA:

En estos procedimientos se tiene la opción de dar una entrada por defecto. Para hacer la entrada por teclado por medio de éstos, la única condición requerida es que se utilice el procedimiento adecuado para cada tipo de variable. Estos procedimientos garantizan que la información que se introduce es correcta para el tipo de variable en cuestión.

Posibilidades para la edición de la entrada:

+ Las teclas de desplazamiento horizontal del cursor operan normalmente.

+ Las teclas "Home" y "End" hacen que el cursor se desplace a la primera y última posición de la entrada respectivamente.

+ Con la tecla "Ins" se puede alternar entre sobreponer caracteres o bien insertarlos. De estas dos posibles estados, el primero es el que actúa por defecto.

+ Con la tecla "Del" se puede borrar el carácter sobre el cual está el cursor.

+ La tecla "BackSpace" hace que se elimine el carácter a la izquierda del cursor.

+ Presionando simultáneamente las teclas "Ctrl" y "Del" se borran todos los caracteres de la entrada.

+ Utilizando la tecla "Esc", se restaura la entrada por defecto.

Implementación de los caracteres del español para cadenas:

Para poder utilizar con sencillez los caracteres del español, se utiliza la comilla, que indica que se introducirá un carácter del español. Después de la comilla se presiona la tecla correspondiente al carácter a convertir, y éste puede ser:

a -->	á	e -->	é	i -->	í	o -->	ó
u -->	ú	v -->	ü	U -->	Ü	n -->	ñ
N -->	Ñ	? -->	¿	! -->	¡		

Nota: No habrá transformación para los caracteres restantes.

También se puede utilizar cualquier tipo de carácter ASCII (*American Standard Code for Information Interchange*) a partir del número treinta y dos utilizando simultáneamente la tecla "Alt" y el número del código de cada carácter.

Procedimientos a utilizar:

PregByte; permite la entrada por teclado de una variable tipo *byte* con un número establecido de dígitos. Para su utilización se requiere del parámetro por variable tipo *byte* a ser editado y la longitud máxima de la entrada.

Ej: Edición de entrada a una variable tipo *byte* proponiendo un cero como entrada por defecto y soportando hasta dos dígitos;

```
variable_byte := 0;
PregByte(variable_byte,2);
```

PregEnte; para la entrada por teclado de una variable tipo entera con la posibilidad de ser restringida a enteros positivos y a cierta longitud máxima. Lo primero es indicar el parámetro por variable a editar, seguido de la longitud máxima para el número ya justificado y del parámetro *booleano* que permitirá la entrada de valores negativos.

Ej: Edición de entrada a una variable tipo entera, con una longitud máxima de cinco caracteres, proponiendo un valor por defecto de trescientos cuarenta y dos, y sólo para negativos;

```
variable_entera := 342;
```

```
PregEnte(variable_entera,5,FALSE);
Nota: el valor máximo es el 9,999.
```

PregReal; hace la edición de entrada de una variable tipo real de longitud máxima determinada, con cierto número de decimales y aceptando o no negativos. Se coloca el parámetro por variable real a editar, seguido de la longitud máxima permisible, el número de decimales y la autorización a valores negativos.

Ej: Edición de entrada a una variable tipo real, con valor por defecto de cero, con una longitud máxima de diez caracteres, con dos decimales y aceptando negativos.

```
variable_real := 0;
PregReal(variable_real,10,2,TRUE);
```

Nota: los valores extremos serán el ±99,999.99

PregCar; hace la edición de entrada de una variable tipo carácter. El parámetro inicial requerido es la variable (de este tipo) a ser editada. Si se desea que aparezca como vacía, se puede usar el carácter nulo (código ASCII número cero).

Ej: Edición de entrada de una variable tipo carácter con valor por defecto nulo.

```
variable_caracter := #0;
PregCar(variable_caracter);
```

PregCadMay; para hacer la edición de entrada de una variable tipo cadena de longitud máxima determinada y convirtiendo toda letra minúscula a mayúscula en el momento de introducirlas por el teclado.

Ej: Pedir una cadena alfanumérica sin letras minúsculas de longitud treinta, proponiendo la cadena "EVENT" como entrada por defecto.

```
variable_cadena := 'EVENT';
PregCadMay(variable_cadena,30);
```

PregCad; utilizada para la entrada de cadenas alfanuméricas totalmente libres. Se coloca el parámetro por variable tipo cadena a editar, seguido de la longitud máxima de la cadena.

Ej: Editar la entrada a una variable tipo cadena, vacía y con una longitud máxima de cuarenta caracteres.

```
variable_cadena := '';
PregCad(variable_cadena,40);
```

El procedimiento Editar, no se utiliza directamente ya que éste es utilizado por los otros procedimientos ya explicados.

Selección de una opción:

El procedimiento que cumple con este propósito es el denominado PregOpcion, el cual permite una sola entrada y está restringido a las teclas que se seleccionen. Para su uso, es necesario indicar los siguientes parámetros:

+ La variable que almacenará la tecla seleccionada (que es de dos caracteres) que será un parámetro por variable.

+ La cadena de caracteres del teclado que se tomarán en cuenta.

+ La cadena que contenga los códigos de las teclas de función o de propósito especial que se puedan aceptar.

Ej: Registrar en la variable opcion la tecla que se seleccione de entre:
- Teclas correspondientes a las letras: "A", "P", "I", "F" y "C".
- Las teclas: "Esc", "F1", "F2" y "F3".

```
PregOpcion(opcion,'APIFC',#27#59#60#61);
```

Con esto se tiene la solución a la entrada de datos por teclado para los fines del programa en cuestión.

SUBPROGRAMA DE ACCESO AL dBase III.

Para crear el subprograma que desde Pascal opere los archivos del dBase III, es necesario saber en qué formato se almacenan las bases de datos. Estudiando una base de datos ya creada, se pudo conocer dicho formato.

El formato está almacenado en un solo archivo junto con los registros de la base de datos; es decir, la primera sección es la que comprende la descripción de la estructura de la base de datos (como es el número de campos que la forman, el nombre de los campos, su tipo, etc.) y en la segunda sección es donde se almacenan todos los registros de la base de datos.

DESCRIPCION DE LA ESTRUCTURA DE LA BASE DE DATOS:

a) DATOS GENERALES:

Byte Nº	Descripción
0	3 (Se refiere a que fue creado por el dBase III).
1	Año de la última modificación.
2	Mes " " " " ".
3	Día " " " " ".
4	
5	* 256
6	* 256^2
7	* 256^3
	Σ = Número de registros existentes.
8	
9	* 256
	Σ = Tamaño de la descripción (TD).
	Número de campos = TD/32 - 1
10	
11	* 256
	Σ = Longitud del registro (LR).

b) DATOS DE LOS CAMPOS:

Nomenclatura

NC -> Número de campo que se va a analizar (1 ≤ NC ≤ número de campos).
PADC(NC) -> Posición absoluta de la descripción del campo NC.

PADC(NC) = NC * 32

A partir de la posición PADC(NC) (inclusive) se encuentra la descripción del campo NC haciendo desplazamientos como lo indica la siguiente tabla.

Desplazamiento	Descripción
00-10	Nombre del campo (Como tiene una sintaxis definida, el nombre del campo termina con un espacio en blanco).
11	Tipo de campo (N, F, C, L, T).
12	*Byte* inicial (No utilizado).
16	Longitud del campo (LC).
17	Número de decimales.

Ej: ¿En qué posición absoluta del archivo se encuentra indicado el tipo de campo del campo número 4?

Respuesta: 4 * 32 + 11 = 139 ← Posición absoluta del Desplazamiento (tipo de campo) ← Número de campo ← Posición absoluta de la desc. del campo

FORMA DE ALMACENAR LOS DATOS EN LOS REGISTROS:

El dBase III almacena todos los tipos de campo en código ASCII, es decir:

+ Los números se almacenan utilizando todas sus posiciones decimales, justificados a la derecha y los espacios libres se dejan en blanco. Por ejemplo:

```
   1542          ->   Campo de long. 7 y sin decimales.
^^^^^^^
       143.20    ->   Campo de longitud 14 y 2 decimales.
^^^^^^^^^^^^^^
  0.0000         ->   Campo de longitud 8 y 4 decimales.
^^^^^^^^
```

+ Las fechas se registran en formato ANSI (*American National Standards Institute*) y ocupan 8 caracteres; los primeros cuatro números son el año, los dos siguientes, el mes, y los dos últimos, son el día. Por ejemplo:

9 de Octubre de 1967 se registra como 19671009.

+ Los campos lógicos se almacenan con una sola letra representativa; T para verdadero y F para falso.

+ Los campos de caracteres se almacenan tal y como se introducen por el usuario; la cadena de caracteres se justifica por la izquierda y los caracteres no utilizados se dejan con espacios en blanco. Por ejemplo:

Angel Urraza #32-1 -> Campo de longitud 25.

+ Los campos de tipo texto no se estudiarán en esta tesis debido a que no se utilizan para el proyecto en cuestión.

El dBase III reserva un carácter al principio de cada registro para marcar los registros que están borrados. Ese carácter es un espacio en blanco cuando no está borrado y un asterisco cuando sí lo está. Por simplicidad y eficiencia en el programa, se hará caso omiso de este carácter; por lo tanto, siempre que se borren registros desde el dBase III se debe de ejecutar la orden PACK para que elimine realmente todos los registros borrados.

A continuación se muestra cómo se pueden localizar las componentes de una base de datos, es decir, la forma de encontrar la información contenida en un campo de un registro deseado.

Nomenclatura

NR ->	Número de registro (1 ≤ NR ≤ Número de registros).
NC ->	Número de campo (1 ≤ NC ≤ Número de campos).
PRC(NC) ->	Posición relativa del campo NC.
LC(NC) ->	Longitud del campo NC (ver inciso 'b)').
PAD(NR,NC) ->	Posición absoluta del dato localizado en NR y NC.
TD ->	Tamaño de la descripción (ver inciso'a)').
LR ->	Longitud del registro (ver inciso 'a)').

Sabiendo que PRC(1) = 0
=> para 2 ≤ NC ≤ número de campos

PRC(NC) = PRC(NC-1) + LC(NC-1)

PAD(NC,NR) = TD + (NR - 1) * LR + PRC(NC) + 1

Con todo este análisis, se puede crear el subprograma de acceso a toda la información contenida en archivos del dBase III. A continuación se muestran los procedimentos y funciones que forman este subprograma.

ACCDBASE.INC (Nombre del fichero que contiene los siguientes procedimientos.)

```
PROCEDURE Iniciacion;                    ( Inicializa la variable general bas. )

  BEGIN                                  ( Coloca ceros en toda la variable 'bas'. )
    FillChar(bas,SizeOf(bas),0);
  END;
```

```
FUNCTION naValido: Boolean;              ( Verifica que el número de base de datos 'na' sea válido. )

  BEGIN                                  ( La función se hace verdadera si 'na' está en el intervalo cerrado )
    naValido := FALSE;                   ( de uno hasta el número máximo de archivos. )
    IF na IN [1..NUM_MAX_ARCH]
      THEN naValido := TRUE
      ELSE MensajeErr('Número de archivo fuera de rango. ** NO CONTINUAR **');
  END;
```

```
FUNCTION LecturaCar: Char;               ( Lee un carácter en el archivo de base de datos. )

  BEGIN
    Read(bas[na].arch, car1);
    LecturaCar := car1;
  END;
```

```
PROCEDURE AbrirArchivo;                    { Abre el archivo número 'na'. }

  VAR
    i : Integer;                           { Usado como contador. }

  PROCEDURE Saltar(num_cars: Byte);        { Salta 'num_cars' caracteres en el archivo de base de datos. }

    VAR
      i: Byte;                             { Usado como contador. }

    BEGIN                                  { Lee del archivo el número de caracteres a saltar. }
      WITH bas[na] DO
        FOR i := 1 TO num_cars DO Read(arch, car1);
    END;

  PROCEDURE Lectura1(VAR entero: Integer); { Lee dos bytes del archivo de base de datos y los coloca en el parámetro }
                                           { var 'entero'. }
    BEGIN
      Read(bas[na].arch, car1, car2);      { Lee dos caracteres del archivo. }
      entero := Ord(car1) + Ord(car2) * 256; { Interpreta dichos caracteres como un entero. }
    END;

  BEGIN
    IF naValido THEN                       { Si el número de archivo indicado es válido: }
      WITH bas[na] DO
        IF abierto = FALSE
          THEN                             { Si dicho archivo no está abierto: }
            BEGIN
              Assign(arch, nom_arch);
              {$I-} Reset(arch); {$I+}     { Verifica que se pueda utilizar el archivo deseado. }
              ok := (IOresult = 0);
              IF ok
                THEN                       { En caso de que sí se haya abierto con éxito: }
                  BEGIN                    { Se lee la estructura de la base de datos. }
                    abierto := TRUE;
                    Seek(arch,4); Lectura1(num_regs);
                    Saltar(2); Lectura1(tam_desc);
                    Lectura1(long_reg);
                    num_camps := Trunc(tam_desc / 32) - 1;
                    pos_camp[1] := 0;
                    FOR i := 1 TO num_camps DO          { Lectura de la descripción de cada campo. }
                      BEGIN
                        Seek(arch, i * 32 + 11); tipo_camp[i] := LecturaCar;
                        Saltar(4); long_camp[i] := Ord(LecturaCar);
                        IF i > 1 THEN pos_camp[i] := pos_camp[i-1] + long_camp[i-1];
                        num_dec[i] := Ord(LecturaCar);
                      END;
                  END
```

```
        ELSE MensajeErr('No se encuentra el archivo ' + nom_arch + '. ** NO CONTINUAR **');
      END
    ELSE MensajeErr('Se intenta abrir un archivo abierto. ** NO CONTINUAR **');
  END;
```

```
FUNCTION YaAbierto: Boolean;                    { Verifica que la base de datos que se pretende utilizar ya esté abierta. }

  BEGIN                                         { La función 'YaAbierto' se hace verdadera sólo cuando el número de }
    YaAbierto := FALSE;                         { archivo en uso sea válido y cuando dicho archivo ya esté abierto. }
    IF naValido THEN
      IF bas[na].abierto
        THEN YaAbierto := TRUE
        ELSE MensajeErr('Se quiere usar el archivo cerrado #'+ Chr(48 + na) + '. ** NO CONTINUAR **');
  END;
```

```
PROCEDURE Compactar(VAR campo : Cad255);        { Elimina los espacios en blanco (caracteres #32) por la izquierda y }
                                                { por la derecha del parámetro var 'campo'. }
  BEGIN
    WHILE (Length(campo) > 0) AND (campo[1] = espacio) DO Delete(campo,1,1);
    WHILE (Length(campo) > 0) AND (campo[Length(campo)] = espacio) DO Delete(campo,Length(campo),1);
  END;
```

```
FUNCTION PosRegCamp: Boolean;                   { Coloca el puntero del archivo en el campo del registro deseado. }

  FUNCTION VerifRegCamp: Boolean;               { Verifica que el número de registro y el número de campo sean válidos. }

    PROCEDURE AgRegArch;                        { Agrega un registro a la base de datos. }

    VAR
      posicion: Real;                           { Posición absoluta dentro del archivo del carácter a leer. }
      i: Integer;                               { Usado como contador. }

    BEGIN
      WITH bas[na] DO
        BEGIN
          un_real := long_reg;                              { Se pasa a un real para la multiplicación. }
          posicion := un_real * num_regs + tam_desc;        { Localiza la posición del registro a agregar. }
          LongSeek(arch, posicion);                         { Se coloca en la posición del registro. }
          FOR i := 0 TO long_reg DO Write(arch, espacio);   { Agrega espacios en blanco al registro. }
          num_regs := num_regs + 1;                         { Actualiza el número de registros de la base de }
          car1 := Chr(Lo(num_regs));                        { datos. }
          car2 := Chr(Hi(num_regs));
          Seek(arch, 4); Write(arch, car1, car2);
        END;
```

```
  END;
```

```
BEGIN                                   { La función 'VerifRegCampo' se hace verdadera sólo en el caso }
  VerifRegCamp := FALSE;                { de que el número del campo exista en la base de datos y que el }
  WITH bas[na] DO                       { número de registro sea como máximo uno después del último, en }
    IF (reg > 0) AND (reg <= num_regs + 1)   { cuyo caso lo agregará. }
      THEN
        IF (camp > 0) AND (camp <= num_camps)
          THEN
            BEGIN
              VerifRegCamp := TRUE;
              IF reg = num_regs + 1 THEN AgRegArch;
            END
          ELSE MensajeErr('Campo fuera de rango en archivo #' + Chr(48 + na) + '. ** NO CONTINUAR **')
      ELSE MensajeErr('Registro fuera de rango en archivo #' + Chr(48 + na) + '. ** NO CONTINUAR **');
END;
```

```
BEGIN                                   { La función 'PosRegCamp' será verdadera si la función 'VerifRegCamp' }
  PosRegCamp := FALSE;                  { también lo es. Si esto sucede, se calcula la posición del dato }
  WITH bas[na] DO                       { buscado y se posiciona el puntero de fichero en dicho dato en el }
    IF VerifRegCamp THEN                { archivo. }
      BEGIN
        PosRegCamp := TRUE;
        un_real := long_reg;
        posicion := un_real * (reg - 1) + tam_desc + pos_camp[camp] + 1;
        LongSeek(arch, posicion);
      END;
END;
```

```
PROCEDURE Leer(VAR campo: Cad255);      { Lee el campo encontrado y lo coloca en el parámetro var 'campo'. }

BEGIN
  WITH bas[na] DO
    IF PosRegCamp THEN                  { Si fue posible localizar la posición del dato: }
      BEGIN                             { Incorpora a la variable 'campo' todo el campo de la base de datos y }
        campo := '';                    { después compacta dicha variable. }
        FOR un_byte := 1 TO long_camp[camp] DO
          campo := campo + LecturaCar;
        Compactar(campo);
      END;
END;
```

```
PROCEDURE Escribir(campo : Cad255);     { Escribe la cadena 'campo' en el campo encontrado. }
```

```
PROCEDURE FormatoC;                     { Acota la cadena a la longitud del campo tipo carácter. }
```

```
  BEGIN                                  { Si la longitud de la cadena 'campo' es mayor que la longitud del }
    WITH bas[na] DO                      { campo, se acotará por la derecha. De lo contrario, se agregarán espacios }
      BEGIN                              { en blanco mientras su longitud sea menor que la del campo. }
        IF Length(campo) > long_camp[camp]
          THEN Delete(campo,long_camp[camp] + 1,Length(campo) - long_camp[camp])
          ELSE WHILE (Length(campo) < long_camp[camp]) DO campo := campo + espacio;
      END;
  END;
```

```
BEGIN
  WITH bas[na] DO
    BEGIN
      CASE tipo_camp[camp] OF            { Verifica que todo esté listo para la escritura en la base de datos. }
        'N' : ;
        'L' : IF (campo <> 'Y') OR (campo <> 'N') THEN
                BEGIN
                  MensajeErr('Se quiere escribir ' + campo + ' en un campo lógico arch. #' + Chr(48 + na) +
                             ', ** NO CONTINUAR **');
                  Exit;
                END;
        'D',
        'C' : FormatoC;
        ELSE
          BEGIN
            MensajeErr('Tipo de campo erróneo en el archivo ' + nom_arch + '. ** NO CONTINUAR **');
            Exit;
          END;
      END;
      IF PosRegCamp THEN                          { Si fue posible encontrar la posición del dato, se escribe }
        FOR un_byte := 1 TO long_camp[camp] DO    { carácter a carácter la cadena 'campo' en la base de datos. }
          Write(arch,campo[un_byte]);
    END;
END;
```

```
PROCEDURE LeerByte(r: Integer;           { Lee del registro 'r' y campo 'c', un campo tipo numérico del archivo }
                   c: Byte;              { de base de datos, la información para el parámetro var 'ent_byte'. }
                   VAR ent_byte: Byte);

BEGIN
  reg := r; camp := c;                   { Pasa las variables locales 'r' y 'c' a las generales 'reg' y 'camp'. }
  un_ente := 0;
  ent_byte := 0;
  IF YaAbierto THEN                      { Si el archivo ya está abierto: }
    WITH bas[na] DO
      IF num_dec[camp] <> 0              { Continuará si el campo a leer no contiene decimales. }
        THEN MensajeErr('Se quiere leer un byte con decimales. ** NO CONTINUAR **')
        ELSE
          IF tipo_camp[camp] <> 'N'      { Continuará si el campo a leer es numérico. }
            THEN MensajeErr('Se quiere leer un byte de un campo no numérico. ** NO CONTINUAR **')
```

```
            ELSE
              BEGIN
                Leer(una_cad);                          { Lee el campo de la base de datos. }
                Val(una_cad,un_ente,num_err);           { Convierte la cadena de lectura a un entero. }
                IF num_err <> 0                         { Continuará si no hubo error en la conversión anterior. }
                  THEN MensajeErr('Se quiere leer un byte mal grabado. ** NO CONTINUAR **')
                  ELSE
                    IF (un_ente < 0) OR (un_ente > 255)
                      THEN MensajeErr('Se quiere leer un byte fuera de rango. ** NO CONTINUAR **')
                      ELSE ent_byte := un_ente;         { Asigna el valor encontrado si está dentro de los límites. }
              END;
  END;
```

```
PROCEDURE LeerEnte(r: Integer;                  { Lee del registro 'r' y campo 'c', un campo tipo numérico del archivo }
                   c: Byte;                     { de base de datos, la información para el parámetro var 'ent_ente'. }
                   VAR ent_ente: Integer);

  BEGIN
    reg := r; camp := c;                        { Pasa las variables locales 'r' y 'c' a las generales 'reg' y 'camp'. }
    ent_ente := 0;
    IF YaAbierto THEN                           { Si el archivo ya está abierto: }
      WITH bas[na] DO
        IF num_dec[camp] <> 0                   { Continuará si el campo a leer no contiene decimales. }
          THEN MensajeErr('Se quiere leer un entero con decimales. ** NO CONTINUAR **')
          ELSE
            IF tipo_camp[camp] <> 'N'           { Continuará si el campo a leer es numérico. }
              THEN MensajeErr('Se quiere leer un entero de un campo no numérico. ** NO CONTINUAR **')
              ELSE
                BEGIN
                  Leer(una_cad);                { Lee el campo de la base de datos. }
                  Val(una_cad,ent_ente,num_err);{ Convierte la cadena de lectura a un entero. }
                  IF num_err <> 0               { Presenta un mensaje si hubo error en la conversión anterior. }
                    THEN MensajeErr('Se quiere leer un entero mal grabado. ** NO CONTINUAR **');
                END;
  END;
```

```
PROCEDURE LeerReal(r: Integer;                  { Lee del registro 'r' y campo 'c', un campo tipo numérico del archivo }
                   c: Byte;                     { de base de datos, la información para el parámetro var 'ent_real'. }
                   VAR ent_real: Real);
  BEGIN
    reg := r; camp := c;                        { Pasa las variables locales 'r' y 'c' a las generales 'reg' y 'camp'. }
    ent_real := 0;
    IF YaAbierto THEN                           { Si el archivo ya está abierto: }
      WITH bas[na] DO
        IF tipo_camp[camp] <> 'N'               { Continuará si el campo a leer es numérico. }
          THEN MensajeErr('Se quiere leer un real de un campo no numérico. ** NO CONTINUAR **')
          ELSE
            BEGIN
              Leer(una_cad);                    { Lee el campo de la base de datos. }
              Val(una_cad,ent_real,num_err);    { Convierte la cadena de lectura a un real. }
              IF num_err <> 0                   { Presenta un mensaje si hubo error en la conversión anterior. }
                THEN MensajeErr('Se quiere leer un real mal grabado. ** NO CONTINUAR **');
            END;
  END;
```

```
PROCEDURE LeerCar(r: Integer;                   { Lee del registro 'r' y campo 'c', un campo tipo carácter del archivo }
                  c: Byte;                      { de base de datos, la información para el parámetro var 'ent_car'. }
                  VAR ent_car: Char);
  BEGIN
    reg := r; camp := c;                        { Pasa las variables locales 'r' y 'c' a las generales 'reg' y 'camp'. }
    IF YaAbierto THEN                           { Si el archivo ya está abierto: }
      WITH bas[na] DO
        BEGIN
          Leer(una_cad);                        { Lee el campo de la base de datos. }
          ent_car := una_cad[1];                { Coloca en la variable carácter el primer componente de la cadena leída. }
        END;
  END;
```

```
PROCEDURE LeerCad(r: Integer;                   { Lee del registro 'r' y campo 'c', un campo tipo carácter del archivo }
                  c: Byte;                      { de base de datos, la información para el parámetro var 'ent_cad'. }
                  VAR ent_cad: Cad255);
  BEGIN
    reg := r; camp := c;                        { Pasa las variables locales 'r' y 'c' a las generales 'reg' y 'camp'. }
    IF YaAbierto THEN                           { Si el archivo ya está abierto: }
      Leer(ent_cad);                            { Lee el campo de la base de datos para colocarlo en 'ent_cad'. }
  END;
```

```
PROCEDURE LeerFecha(r: Integer;                 { Lee del registro 'r' y campo 'c', un campo tipo fecha del archivo }
                    c: Byte;                    { de base de datos, la información para el parámetro var 'ent_fech'. }
                    VAR ent_fech: TFecha );
  BEGIN
    reg := r; camp := c;                        { Pasa las variables locales 'r' y 'c' a las generales 'reg' y 'camp'. }
    IF YaAbierto THEN                           { Si el archivo ya está abierto: }
      WITH bas[na] DO
        IF tipo_camp[camp] <> 'D'               { Continuará si el campo a leer es de fechas. }
          THEN MensajeErr('Se quiere leer una fecha de un campo de otro tipo. ** NO CONTINUAR **')
          ELSE
            BEGIN
              Leer(una_cad);                    { Lee el campo de la base de datos. }
              IF Length(una_cad) = 8
                THEN                            { Si es de ocho caracteres la longitud de la cadena lida, la interpreta }
                  ent_fech := una_cad[7] + una_cad[8] + '/' +       { como una fecha de la forma "dia/mes/año". }
                              una_cad[5] + una_cad[6] + '/' +
                              una_cad[3] + una_cad[4]
                ELSE ent_fech := '';
            END;
  END;
```

```
PROCEDURE EscribirByte(r: Integer;              { Escribe en el registro 'r' y campo 'c', un campo tipo numérico del }
                       c: Byte;                 { archivo de base, de datos la información del parámetro 'ent_byte'. }
                       ent_byte: Byte);
  BEGIN
    reg := r; camp := c;                        { Pasa las variables locales 'r' y 'c' a las generales 'reg' y 'camp'. }
    IF YaAbierto THEN                           { Si el archivo ya está abierto: }
      WITH bas[na] DO
        IF num_dec[camp] <> 0                   { Continuará si el campo a escribir no contiene decimales. }
          THEN MensajeErr('Se quiere escribir un byte con decimales. ** NO CONTINUAR **')
          ELSE
            IF tipo_camp[camp] <> 'N'           { Continuará si el campo a escribir es numérico. }
              THEN MensajeErr('Se quiere escribir un byte en campo no numérico. ** NO CONTINUAR **')
              ELSE
                BEGIN                           { Pasa la variable byte a una variable cadena y la escribe en la b. d. }
                  Str(ent_byte: long_camp[camp], una_cad);
                  Escribir(una_cad);
                END;
  END;
```

```
PROCEDURE EscribirEnte(r: Integer;              { Escribe en el registro 'r' y campo 'c', un campo tipo numérico del }
                       c: Byte;                 { archivo de base, de datos la información del parámetro 'ent_ente'. }
                       ent_ente: Integer);
  BEGIN
    reg := r; camp := c;                        { Pasa las variables locales 'r' y 'c' a las generales 'reg' y 'camp'. }
    IF YaAbierto THEN                           { Si el archivo ya está abierto: }
      WITH bas[na] DO
        IF num_dec[camp] <> 0                   { Continuará si el campo a escribir no contiene decimales. }
          THEN MensajeErr('Se quiere escribir un entero con decimales. ** NO CONTINUAR **')
          ELSE
            IF tipo_camp[camp] <> 'N'           { Continuará si el campo a escribir es numérico. }
              THEN MensajeErr('Se quiere escribir un entero en campo no numérico. ** NO CONTINUAR **')
              ELSE
                BEGIN                           { Pasa la variable entera a una variable cadena y la escribe en la b. d. }
                  Str(ent_ente: long_camp[camp], una_cad);
                  Escribir(una_cad);
                END;
  END;
```

```
PROCEDURE EscribirReal(r: Integer;              { Escribe en el registro 'r' y campo 'c', un campo tipo numérico del }
                       c: Byte;                 { archivo de base de datos, la información del parámetro 'ent_real'. }
                       ent_real: Real);
  BEGIN
    reg := r; camp := c;                        { Pasa las variables locales 'r' y 'c' a las generales 'reg' y 'camp'. }
    IF YaAbierto THEN                           { Si el archivo ya está abierto: }
      WITH bas[na] DO
        IF tipo_camp[camp] <> 'N'               { Continuará si el campo a escribir es numérico. }
          THEN MensajeErr('Se quiere escribir un real en un campo no numérico. ** NO CONTINUAR **')
          ELSE
            BEGIN                               { Pasa la variable real a una variable cadena y la escribe en la b. d. }
              Str(ent_real: long_camp[camp]: num_dec[camp], una_cad);
              Escribir(una_cad);
            END;
  END;
```

```
PROCEDURE EscribirCar(r: Integer;               { Escribe en el registro 'r' y campo 'c', un campo tipo carácter del }
                      c: Byte;                  { archivo de base de datos, la información del parámetro 'ent_car'. }
                      ent_car: Char);
  BEGIN
    reg := r; camp := c;                        { Pasa las variables locales 'r' y 'c' a las generales 'reg' y 'camp'. }
    IF YaAbierto THEN                           { Si el archivo ya está abierto: }
      WITH bas[na] DO
        IF (tipo_camp[camp] <> 'C') AND         { Continuará si el campo a escribir es tipo carácter o lógico. }
           (tipo_camp[camp] <> 'L')
          THEN MensajeErr('Se quiere escribir un carácter en campo inválido. ** NO CONTINUAR **')
          ELSE
            BEGIN
```

```
            una_cad := ent_car;              { Pasa la variable carácter a una variable cadena y la escribe en la b. d. }
            Escribir(una_cad);
          END;
  END;
```

```
PROCEDURE EscribirCad(r: Integer;            { Escribe en el registro 'r' y campo 'c', un campo tipo carácter del }
                      c: Byte;               { archivo de base de datos, la información del parámetro 'ent_cad'. }
                      ent_cad: Cad255);
  BEGIN
    reg := r; camp := c;                     { Pasa las variables locales 'r' y 'c' a las generales 'reg' y 'camp'. }
    IF YaAbierto THEN                        { Si el archivo ya está abierto: }
      WITH bas[na] DO
        IF (tipo_camp[camp] <> 'C') AND      { Continuará si el campo a escribir es tipo carácter o lógico. }
           (tipo_camp[camp] <> 'L')
          THEN MensajeErr('Se quiere escribir una cadena en campo inválido. ** NO CONTINUAR **')
          ELSE Escribir(ent_cad);            { Escribe la cadena en la base de datos. }
  END;
```

```
PROCEDURE EscribirFecha(r: Integer;          { Escribe en el registro 'r' y campo 'c', un campo tipo fecha del }
                        c: Byte;             { archivo de base de datos, la información del parámetro 'ent_fech'. }
                        ent_fech: TFecha );
  BEGIN
    reg := r; camp := c;                     { Pasa las variables locales 'r' y 'c' a las generales 'reg' y 'camp'. }
    IF YaAbierto THEN                        { Si el archivo ya está abierto: }
      WITH bas[na] DO
        IF tipo_camp[camp] <> 'D'            { Continuará si el campo a escribir es de tipo fecha. }
          THEN MensajeErr('Se quiere escribir una fecha en campo inválido. ** NO CONTINUAR **')
          ELSE
            BEGIN
              una_cad := '';
              IF Length(ent_fech) = 8 THEN   { Si la fecha a registrar tiene longitud ocho, se coloca la fecha en otra }
                una_cad := '19' + ent_fech[7] + ent_fech[8] +          { cadena pero ya con el formato adecuado. }
                           ent_fech[4] + ent_fech[5] +
                           ent_fech[1] + ent_fech[2];
              Escribir(una_cad);             { Escribe la cadena en la base de datos. }
            END;
  END;
```

```
PROCEDURE CerrarArchivo;                     { Cierra el archivo 'na' (en uso). }

  BEGIN
    WITH bas[na] DO
      IF abierto = TRUE
        THEN                                 { Lo cierra si el archivo ya estaba abierto. }
          BEGIN
            Close(arch);
```

```
        abierto := FALSE;
      END
    ELSE MensajeErr('Se intenta cerrar un archivo no abierto.');
END;
```

Las variables generales utilizadas son: bas, na, reg y camp. En la variable bas, se almacena la descripción del formato de las bases de datos. La variable na indica el número de la base de datos en uso, y las variables reg y camp indican, respectivamente, el número de registro y el campo actual.

La utilización del subprograma de acceso a los archivos del dBase III es muy sencilla, y se resume en los siguientes pasos:

1. Ejecutar el procedimiento Iniciacion, antes de trabajar con las bases de datos. Al hacerlo inicializará la variable bas, es decir, dejará indicado en la variable bas que todas la bases de datos están cerradas (Ej: Iniciacion;).

2. Abrir la base de datos deseada, lo cual es esencial para poder utilizarla. Para hacer esto hay que cargar en la variable na el número de archivo que se desea utilizar (Ej: operar con el archivo #2: na := 2;), cerciorándose de que ese número de archivo no está ya utilizado. Después de esto, se debe de almacenar el nombre del archivo a abrir en la variable nom_arch, y por medio del procedimiento AbrirArchivo se abre el archivo indicado.

3. Leer los datos requeridos; para esto se necesita almacenar en la variable na el número de archivo que se desea usar, y pasar la información del disco a la memoria RAM, utilizando los procedimientos LeerByte, LeerEnte, LeerReal, LeerCar, LeerCad y LeerFecha. Estos procedimientos se usan colocando como primer argumento el número de registro a leer; como segundo, el número de campo a leer, y como último, la variable en la cual se almacenará la información. Hay un procedimiento específico para cada tipo de variable (Ej: leer una cantidad entera del registro #10 y campo #2 de la base de datos en uso: LeerEnte(10,2,variable_entera);).

4. Escribir los datos requeridos; en forma similar al inciso anterior, hay que indicar el número de base de datos a utilizar y los procedimientos EscribirByte, EscribirEnte, EscribirReal, EscribirCar, EscribirCad y EscribirFecha para pasar la información de la memoria RAM al disco. Estos procedimientos se utilizan en forma análoga a la lectura del disco (Ej: escribir una fecha en el registro #50 y campo #7

de la base de datos en uso: EscribirFecha(50,7,variable_cadena_de_fecha);).

5. Cerrar una base de datos, al terminar de utilizarla, para que se actualice el directorio del disco. Para cerrar una base de datos se debe de indicar en na el número de base de datos que se desea cerrar y después ejecutar el procedimiento CerrarArchivo.

Cada vez que se utilizan los procedimientos de lectura y escritura para la base de datos, se hacen los cálculos para localizar cada campo en el archivo. Esto hace que el acceso a disco sea relativamente lento, por lo que será de relevante importancia hacer el mínimo de accesos al disco para lograr una mayor velocidad en el programa. La sección clave para lograrlo será el procedimiento denominado CargaInicial que se comenta más adelante.

ULTIMOS PROCEDIMIENTOS DE PROPOSITO GENERAL.

Estos últimos subprogramas se localizan al final por necesitar de algún otro que se encuentra antes.

VARIOS2.INC (Fichero que contiene el código fuente que se muestra.)

```
FUNCTION OkPrepararImpresora: Boolean;          ( Función booleana que se hace verdadera cuando se ha preparado la imp. )
;
;  VAR
;    opcion: TTecla;                            ( Para registrar la tecla presionada. )
;
;  BEGIN
;    OkPrepararImpresora := TRUE;
;    GotoXY(1,23);ClrEol; Write('Coloque el papel en la impresora y enciéndala <Enter> <Esc>');
;    PregOpcion(opcion,CR + ESC, '');           ( Se pide que se presione la tecla "ENTER" o "ESC". )
;    WriteLn;
;    IF opcion[1] = CR                          ( Si se presionó la tecla "ENTER" se inicializa la impresora y la función )
;      THEN Write(Lst, ESC,'@')                 ( 'OkPrepararImpresora' permanecerá como verdadera. Si por lo contrario )
;      ELSE OkPrepararImpresora := FALSE;       ( se presionó la tecla de escape, la función se cambia a falso. )
;  END;
;
```

```
PROCEDURE Terminar;                             ( Cierra los archivos para salir del programa. )
;
;  BEGIN
;    FOR na := 1 TO 3 DO
;      CerrarArchivo;
;  END;
```

OkPrepararImpresora; función *booleana* que verifica que la impresora esté preparada. Esta función no requiere de argumentos y lo que hace es pedir al usuario que encienda la impresora, si no lo está, y enviar un código de inicialización a la impresora. Este procedimiento se utiliza cada vez que se quiera imprimir algo.

Terminar; procedimiento que cierra los archivos de base de datos abiertos. No requiere de argumentos y se utiliza al final de la ejecución del programa.

4.3.4 SUBPROGRAMAS UTILIZADOS REPETIDAMENTE.

Estos son los procedimientos y funciones que no fue posible colocar permanentemente en memoria RAM de la computadora, por limitaciones en el tamaño del código objeto del Turbo Pascal. Estos subprogramas se colocarán sólo en aquellas secciones donde se les necesite.

FUNCIONES DE VERIFICACION.

Estas funciones se hacen verdaderas en el caso de que la tarea realizada se haya llevado a cabo con éxito. A continuación se muestra el código fuente de dichas funciones y después se pasará a la explicación de cada una de ellas.

OK.BBL (Fichero que contiene el código fuente que se muestra.)

```
FUNCTION OkMaterial(     artic: TArtic;                ( Función que verifica la existencia del material de clave )
                     VAR numero: Integer ): Boolean;   ( 'artic' y pone su número de registro en el parámetro var )
                                                       ( 'numero'. )
  PROCEDURE Busqueda(inicio,fin: Integer);             ( Procedimiento recursivo que localiza el artículo buscado. )

  BEGIN                                                ( Este procedimiento inicia la búsqueda en un intervalo cerrado )
    IF inicio > fin                                    ( de registros. Localiza el número de reg. central del rango, )
      THEN numero := 0                                 ( en caso de que no sea el buscado y que aún se pueda continuar )
      ELSE                                             ( la búsqueda, hace la llamada recursiva al procedimiento con )
        BEGIN                                          ( el nuevo rango a analizar. )
          numero := (inicio + fin) DIV 2;
          IF artic < pnt_ap[numero]^.artic
            THEN Busqueda(inicio,numero - 1)
            ELSE
              IF artic > pnt_ap[numero]^.artic THEN Busqueda(numero + 1,fin);
        END;
  END;

  BEGIN
    IF (artic = '') OR (nua_ap = 0)
      THEN numero := 0
      ELSE Busqueda(1,nua_ap);
    OkMaterial := (numero > 0);
  END;
```

```
FUNCTION OkProducto(      artic: TArtic;                  { Función que verifica la existencia del producto terminado de }
                    VAR numero: Integer ): Boolean;       { clave 'artic' y pone su número de registro en el parámetro var }
                                                          { 'numero'. }

  PROCEDURE Busqueda(inicio,fin: Integer);                { Procedimiento recursivo que localiza el articulo buscado. }

  BEGIN                                                   { Este procedimiento inicia la búsqueda en un intervalo cerrado }
    IF inicio > fin                                       { de registros. Localiza el número de reg. central del rango, }
      THEN numero := 0                                    { en caso de que no sea el buscado y que aún se pueda continuar }
      ELSE                                                { la búsqueda, hace la llamada recursiva al procedimiento con }
        BEGIN                                             { el nuevo rango a analizar. }
          numero := (inicio + fin) DIV 2;
          IF artic < pnt_pt[numero]^.artic
            THEN Busqueda(inicio,numero - 1)
            ELSE
              IF artic > pnt_pt[numero]^.artic THEN Busqueda(numero + 1,fin);
        END;
  END;

BEGIN
  IF (artic = '') OR (num_pt = 0)
    THEN numero := 0
    ELSE Busqueda(1,num_pt);
  OkProducto := (numero > 0);
END;
```

```
FUNCTION OkSubensaeble(      artic: TArtic;               { Función que verifica la existencia de un subensamble de clave }
                         VAR numero: Integer ): Boolean;  { 'artic' y pone su número de registro en el parámetro var }
BEGIN                                                     { 'numero'. }
  OkSubensamble := FALSE;
  IF artic <> '' THEN                                     { Si la cadena no es nula: }
    IF artic[1] = CLA_SUBE
      THEN                                                { Se iniciará la búsqueda si contiene la clave de maquila. }
        IF OkMaterial(artic,numero)
          THEN OkSubensamble := TRUE                      { Se hace verdadero si fue encontrada. }
          ELSE MensajeErr('No está registrado el subensamble ' + artic)
      ELSE MensajeErr('No contiene el signo (' + CLA_SUBE + ').');
END;
```

```
FUNCTION OkMaquila(      artic: TArtic;                   { Función que verifica la existencia de un producto de maquila }
                     VAR numero: Integer ): Boolean;      { de clave 'artic' y pone su número de registro en el parámetro }
                                                          { var 'numero'. }
BEGIN
  OkMaquila := FALSE;
  IF artic <> '' THEN                                     { Si la cadena no es nula: }
    IF artic[1] = CLA_MAQ
```

```
      THEN                                    { Se iniciará la búsqueda si contiene la clave de maquila. }
        IF OkMaterial(artic,numero)
          THEN OkMaquila := TRUE              { Se hace verdadero si fue encontrada. }
          ELSE MensajeErr('No está registrado el producto de maquila ' + artic)
      ELSE MensajeErr('No contiene el signo (' + CLA_MAQ + ').');
  END;
```

```
FUNCTION OkDesProduccion(     artic: TArtic;          { Función que verifica la existencia de la descripción de }
                          VAR numero: Integer ): Boolean; { producción del elemento de clave 'artic' y pone su número }
                                                      { de registro en el parámetro var 'numero'. }
```

```
  PROCEDURE Busqueda(inicio,fin: Integer);            { Procedimiento recursivo que localiza el artículo buscado. }

  BEGIN                                               { Este procedimiento inicia la búsqueda en un intervalo cerrado }
    IF inicio > fin                                   { de registros. Localiza el número de reg. central del rango, }
      THEN numero := 0                                { en caso de que no sea el buscado y que aún se pueda continuar }
      ELSE                                            { la búsqueda, hace la llamada recursiva al procedimiento con }
        BEGIN                                         { el nuevo rango a analizar. }
          numero := (inicio + fin) DIV 2;
          IF artic < pnt_prod[numero]^.artic
            THEN Busqueda(inicio,numero - 1)
            ELSE
              IF artic > pnt_prod[numero]^.artic THEN Busqueda(numero + 1,fin);
        END;
  END;
```

```
  BEGIN
    IF (artic = '') OR (num_prod = 0)
      THEN numero := 0
      ELSE Busqueda(1,num_prod);
    OkDesProduccion := (numero > 0);
  END;
```

```
FUNCTION OkDesMaquila(     artic: TArtic;          { Función que verifica la existencia de la descripción de }
                       VAR numero: Integer ): Boolean;  { maquila del elemento de clave 'artic' y pone su número }
                                                      { de registro en el parámetro var 'numero'. }
```

```
  PROCEDURE Busqueda(inicio,fin: Integer);            { Procedimiento recursivo que localiza el artículo buscado. }

  BEGIN                                               { Este procedimiento inicia la búsqueda en un intervalo cerrado }
    IF inicio > fin                                   { de registros. Localiza el número de reg. central del rango, }
      THEN numero := 0                                { en caso de que no sea el buscado y que aún se pueda continuar }
      ELSE                                            { la búsqueda, hace la llamada recursiva al procedimiento con }
        BEGIN                                         { el nuevo rango a analizar. }
          numero := (inicio + fin) DIV 2;
          IF artic < pnt_maq[numero]^.artic
            THEN Busqueda(inicio,numero - 1)
```

```
        ELSE
          IF artic > pnt_aaq[numero]^.artic THEN Busqueda(numero + 1,fin);
      END;
  END;
```

```
  BEGIN
    IF (artic = '') OR (num_aaq = 0)
      THEN numero := 0
      ELSE Busqueda(1,num_aaq);
    OkDesMaquila := (numero > 0);
  END;
```

```
FUNCTION OkPartesProduccion(       artic: TArtic;              { Función que verifica la descripción de partes para }
                           VAR    numero: Integer;            { producción del elemento 'artic', pone su número de }
                           VAR part_usad: TPartUsad): Boolean; { registro en el parámetro var 'número' y coloca las }
  VAR                                                          { partes en el parámetro var 'part_usad'. }
    salir: Boolean;                                 { Permite la salida. }
    pos_guion,                                      { Posición del guión. }
    pos_espacio,                                    { Posición del espacio en blanco. }
    cont_partes: Byte;                              { Contador de componentes necesarios. }
    numero_ins,                                     { Número de registro de un componente. }
    cant: Integer;                                  { Cantidad necesaria del componente. }
    partes: TPartes;                                { Cadena que contiene la descripción de partes (componentes). }
    artic_ins: TArtic;                              { Un componente. }
    cad_cant: String[5];                            { Cantidad necesaria del componente en variable cadena. }

  BEGIN
    OkPartesProduccion := FALSE;
    cont_partes := 0;
    salir := FALSE;
    IF OkDesProduccion(artic,numero) THEN           { Continuará sólo en el caso de que se encuentre la descripción }
      BEGIN                                         { de partes y localizará el número de registro. }
        partes := pnt_prod[numero]^.partes;         { Recupera de la memoria dinámica la descripción de partes. }
        REPEAT
          WHILE partes[1] = ' ' DO Delete(partes,1,1);  { Borra los espacios en blanco del principio de la cadena. }
          IF partes[1] = '*'
            THEN salir := TRUE                      { Termina cuando encuentra el asterisco. }
            ELSE
              BEGIN
                pos_guion := Pos('-',partes);       { Localiza la posición del próximo guión y espacio en blanco. }
                pos_espacio := Pos(' ',partes);
                IF (pos_guion > 1) AND (pos_espacio > 3)
                  THEN                              { Si se encontró el guión y el espacio dentro de valores válidos }
                    BEGIN                           { se procederá a obtener la clave del componente y la cantidad. }
                      artic_ins := Copy(partes,1,pos_guion - 1);
                      cad_cant := Copy(partes,pos_guion + 1, pos_espacio - pos_guion - 1);
                      Val(cad_cant,cant,num_err);
                      IF OkMaterial(artic_ins,numero_ins)
                        THEN                                { Si el componente es un material registrado y }
```

```
                    IF num_err = 0                          { la cantidad está correctamente indicada, se }
                      THEN                                  { almacenan en 'part_usad' los datos encontrados. }
                        BEGIN
                          cont_partes := Succ(cont_partes);
                          part_usad[cont_partes].numero := numero_ins;
                          part_usad[cont_partes].cant  := cant;
                        END
                      ELSE MensajeErr('Está mal la cantidad de ' + artic_ins + ' para el ' + artic)
                    ELSE MensajeErr('No está registrado el componente ' + artic_ins + ' para el ' + artic);
                  Delete(partes,1,pos_espacio);             { Elimina los caracteres ya analizados. }
                END
              ELSE salir := TRUE;
          END;
      UNTIL salir;
      part_usad[0].cant := cont_partes;
      IF cont_partes = 0                          { Si no se encontraron componentes, no se acepta la descripción. }
        THEN MensajeErr('Están mal definidas las partes para producción del ' + artic)
        ELSE OkPartesProduccion := TRUE;
    END;
  END;
```

```
FUNCTION OkPartesMaquila(        artic: TArtic;             { Función que verifica la descripción de partes para }
                         VAR    numero: Integer;            { maquila del elemento 'artic', pone su número de }
                         VAR part_usad: TPartUsad): Boolean; { registro en el parámetro var 'número' y coloca las }
  VAR                                                        { partes en el parámetro var 'part_usad'. }
    salir: Boolean;                                 { Permite la salida. }
    pos_guion,                                      { Posición del guión. }
    pos_espacio,                                    { Posición del espacio en blanco. }
    cont_partes: Byte;                              { Contador de componentes necesarios. }
    numero_ins,                                     { Número de registro de un componente. }
    cant: Integer;                                  { Cantidad necesaria del componente. }
    partes: TPartes;                                { Cadena que contiene la descripción de partes (componentes). }
    artic_ins: TArtic;                              { Un componente. }
    cad_cant: String[5];                            { Cantidad necesaria del componente en variable cadena. }

  BEGIN
    OkPartesMaquila := FALSE;
    cont_partes := 0;
    salir := FALSE;
    IF OkDesMaquila(artic,numero) THEN              { Continuará sólo en el caso de que se encuentre la descripción }
      BEGIN                                         { de partes y localizará el número de registro. }
        partes := pnt_aaq[numero]^.partes;          { Recupera de la memoria dinámica la descripción de partes. }
        REPEAT
          WHILE partes[1] = ' ' DO Delete(partes,1,1);  { Borra los espacios en blanco del principio de la cadena. }
          IF partes[1] = '*'
            THEN salir := TRUE                      { Termina cuando encuentra el asterisco. }
            ELSE
              BEGIN
                pos_guion := Pos('-',partes);       { Localiza la posición del próximo guión y espacio en blanco. }
                pos_espacio := Pos(' ',partes);
```

```
            IF (pos_guion > 1) AND (pos_espacio > 3)
              THEN                                  { Si se encontró el guión y el espacio dentro de valores válidos }
                BEGIN                               { se procederá a obtener la clave del componente y la cantidad. }
                  artic_ins := Copy(partes,1,pos_guion - 1);
                  cad_cant := Copy(partes,pos_guion + 1, pos_espacio - pos_guion - 1);
                  Val(cad_cant,cant,num_err);
                  IF OkMaterial(artic_ins,numero_ins)
                    THEN                                    { Si el componente es un material registrado y }
                      IF num_err = 0                        { la cantidad está correctamente indicada, se }
                        THEN                                { almacenan en 'part_usad' los datos encontrados. }
                          BEGIN
                            cont_partes := Succ(cont_partes);
                            part_usad[cont_partes].numero := numero_ins;
                            part_usad[cont_partes].cant  := cant;
                          END
                        ELSE MensajeErr('Está mal la cantidad de ' + artic_ins + ' para el ' + artic)
                    ELSE MensajeErr('No está registrado el componente ' + artic_ins + ' para el ' + artic);
                  Delete(partes,1,pos_espacio);         { Elimina los caracteres ya analizados. }
                END
              ELSE salir := TRUE;
            END;
        UNTIL salir;
        part_usad[0].cant := cont_partes;
        IF cont_partes = 0                          { Si no se encontraron componentes, no se acepta la descripción. }
          THEN MensajeErr('Están mal definidas las partes para maquila del ' + artic)
          ELSE OkPartesMaquila := TRUE;
      END;
  END;
```

```
FUNCTION OkProveedor(     clave: TClave;            { Función que verifica la existencia del proveedor 'clave' y }
                     VAR numero: Integer ): Boolean;  { pone su número de registro en el parámetro var 'numero'. }
```

```
  PROCEDURE Busqueda(inicio,fin: Integer);          { Procedimiento recursivo que localiza la clave buscada. }

  BEGIN                                             { Este procedimiento inicia la búsqueda en un intervalo cerrado }
    IF inicio > fin                                 { de registros. Localiza el número de reg. central del rango, }
      THEN numero := 0                              { en caso de que no sea el buscado y que aún se pueda continuar }
      ELSE                                          { la búsqueda, hace la llamada recursiva al procedimiento con }
        BEGIN                                       { el nuevo rango a analizar. }
          numero := (inicio + fin) DIV 2;
          IF clave < pro[numero]
            THEN Busqueda(inicio,numero - 1)
            ELSE
              IF clave > pro[numero] THEN Busqueda(numero + 1,fin);
        END;
  END;
```

```
  BEGIN
```

```
  IF (clave = '') OR (num_pro = 0)
    THEN numero := 0
    ELSE Busqueda(1,num_pro);
  OkProveedor := (numero > 0);
END;
```

```
FUNCTION OkDistribuidor(     clave: TClave;              { Función que verifica la existencia del distribuidor 'clave' y }
                         VAR numero: Integer ): Boolean; { pone su número de registro en el parámetro var 'numero'. }

  PROCEDURE Busqueda(inicio,fin: Integer);               { Procedimiento recursivo que localiza la clave buscada. }

  BEGIN                                                  { Este procedimiento inicia la búsqueda en un intervalo cerrado }
    IF inicio > fin                                      { de registros. Localiza el número de reg. central del rango, }
      THEN numero := 0                                   { en caso de que no sea el buscado y que aún se pueda continuar }
      ELSE                                               { la búsqueda, hace la llamada recursiva al procedimiento con }
        BEGIN                                            { el nuevo rango a analizar. }
          numero := (inicio + fin) DIV 2;
          IF clave < dis[numero]
            THEN Busqueda(inicio,numero - 1)
            ELSE
              IF clave > dis[numero] THEN Busqueda(numero + 1,fin);
        END;
  END;

BEGIN
  IF (clave = '') OR (num_dis = 0)
    THEN numero := 0
    ELSE Busqueda(1,num_dis);
  OkDistribuidor := (numero > 0);
END;
```

```
FUNCTION OkCambioPrecios: Boolean;              { Función que verifica que los precios unitarios de las partidas de la }
                                                { variable 'pedido' sean los correctos. }
  VAR
    cont,                                       { Contador de partida. }
    loc_artic: Integer;                         { Localización del artículo. }

  BEGIN
    WITH pedido DO
      BEGIN
        OkCambioPrecios := FALSE;
        FOR cont := 1 TO n_p DO                 { Se hará el análisis para todas las partidas. }
          WITH part[cont] DO
            IF OkProducto(artic,loc_artic)
              THEN                              { Si se ha localizado el artículo como un producto }
                IF precio <> pnt_pt[loc_artic]^.precio_v   { terminado y se encuentra que el precio ha cambiado, }
                  THEN OkCambioPrecios := TRUE  { el indicador de cambio de precio se hace verdadero. }
                  ELSE
              ELSE
                IF OkMaterial(artic,loc_artic)
                  THEN                          { Si se ha localizado el artículo como un material y }
                    IF precio <> pnt_mp[loc_artic]^.precio_v   { se encuentra que el precio ha cambiado, el indicador }
                      THEN OkCambioPrecios := TRUE              { de cambio de precio se hace verdadero. }
                      ELSE
                  ELSE MensajeErr( 'El artículo ' + artic + ' en la partida ' + JusEnte(cont,2) +
                                   ' ya no está registrado' );
      END;
  END;
```

```
FUNCTION OkImprimirForma(  nombre: Cad78;                   { Función para la impresión del formato 'nombre' con }
                           num_cop: Integer  ): Boolean;    { 'num_cop' número de copias. El valor que tome la }
                                                            { función dependerá de que el(los) impreso(s) se }
    VAR                                                     { concluyan con éxito. }
      cont_imp,                                 { Contador de impresión. }
      cont_ren,                                 { Contador de renglón de la hoja de papel. }
      cont,                                     { Número de carácter de la línea en estudio. }
      posi,                                     { Posición relativa a 'cont' para obtener la clave de acceso 'acceso'. }
      tam,                                      { Tamaño de la inserción a realizar. }
      r_e,                                      { Renglón de }
      c_e,                                      { Columna de }
      tam_e: Integer;                           { Tamaño de }
      opcion: TTecla;                           { Tecla de opción. }
      arch: Text;                               { Archivo a utilizar. }
      acceso: Cad4;                             { Clave para el acceso de un campo al formato. }
      linea,                                    { Línea que se leerá del archivo de formato. }
      externa: String[132];                     { Cadena externa al formato. }
      hoja: Array[1..60] OF String[132];        { Variable que memorizará la hoja a imprimir. }
      pantalla_mem: TPant;                      { Variable que memorizará la pantalla de visualización. }

    BEGIN
```

```
    OkImprimirForma := FALSE;
    Assign(arch,nombre);
    pantalla_aux := pantalla;              { Memoriza la pantalla de visualización actual. }
    cont_imp := 1;
    REPEAT
      ClrScr;
      Reset(arch);                         { Abre el archivo de texto que contiene el formato. }
      cont_ren := 0;
      tam_e := 0;
      WHILE NOT(Eof(arch)) DO
        BEGIN
          cont_ren := Succ(cont_ren);
          ReadLn(arch,linea);              { Lee una línea del formato. }
          cont := 0;
          WHILE cont < Length(linea) DO    { Mientras el contador no haya alcanzado la longitud de la línea }
            BEGIN                          { se repetirá la operación. }
              cont := Succ(cont);
              CASE linea[cont] OF
                '@' : BEGIN                { Si se ha encontrado el mando de inserción, se calcula la longitud de la }
                      tam := 1;            { misma y se procede a su análisis. }
                      WHILE (linea[cont + tam] <> ' ')
                          AND (cont + tam <= Length(linea)) DO
                        tam := Succ(tam);
                      CASE linea[cont + 1] OF
                        'C' : BEGIN                                  { Si se encuentra la letra "C" leerá una }
                              externa := ''; PregCad(externa,tam);   { cadena de longitud 'tam' desde el teclado. }
                              GotoXY(cont,WhereY);
                              Solapar(JusCad(externa,tam),linea,cont);
                            END;
                        'E' : BEGIN                                  { Si se encuentra la letra "E" localizará el }
                              c_e := cont;                           { lugar en donde se debe presentar el }
                              r_e := cont_ren;
                              tam_e := tam;
                              externa := JusCad('ORIGINAL',tam);
                              Solapar(externa,linea,cont);
                            END;
                        ELSE BEGIN                                   { En cualquier otro caso se tomará por clave }
                              acceso := '';                          { de acceso a todo carácter distinto de }
                              posi := cont + 1;                      { espacio y subrayado. }
                              WHILE (linea[posi] <> '_') AND (linea[posi] <> ' ')
                                  AND (posi - cont < 5) AND (posi < Length(linea)) DO
                                BEGIN
                                  acceso := acceso + linea[posi];
                                  posi := Succ(posi);
                                END;
                              externa := JusCad(CadenaExterna(acceso),tam);      { Evalúa la cadena externa. }
                              Solapar(externa,linea,cont);                       { Solapa la cadena externa. }
                            END;
                      END;
                    END;
              END;
              Write(linea[cont]);                                        { Presenta la línea en pantalla. }
```

```
            END;
          WriteLn;
          hoja[cont_ren] := linea;                    { Memoriza la línea analizada. }
        END;
      WriteLn;
      WriteLn;
      GotoXY(26,23); Write('¿Está todo correcto? (S/N) (Esc) ');
      PregOpcion(opcion,'SN' + ESC,'');
      WriteLn;
    UNTIL opcion[1] <> 'N';                           { Repetirá la operación en caso negativo. }
    Close(arch);
    IF opcion[1] <> ESC THEN                          { Si no se canceló la operación prepara la }
      IF OkPrepararImpresora THEN                     { impresora. }
        BEGIN
          GotoXY(1,23); ClrEol;
          GotoXY(24,23); WriteLn('E S P E R E   U N   M O M E N T O');
          OkImprimirForma := TRUE;
          FOR cont_imp := 1 TO num_cop + 1 DO         { Imprimirá el formato el número de veces }
            BEGIN                                     { requerido. En caso de que se haya }
              IF (cont_imp = 2) AND (tam_e > 0) THEN  { utilizado el mando "@E" y que sea la }
                BEGIN                                 { segunda impresión, colocará el letrero }
                  externa := JusCad(' COPIA',tam_e);  { "COPIA" en el lugar indicado. }
                  Solapar(externa,hoja[r_e],c_e);
                END;
              FOR cont := 1 TO cont_ren DO            { Imprime todos los renglones del formato y }
                WriteLn(Lst,hoja[cont]);              { pasa a la siguiente hoja. }
              Write(Lst,FF);
            END;
        END;
    pantalla := pantalla_ocu;                         { Restaura la pantalla original. }
  END;
```

La localización en memoria RAM de las claves, que realizan algunas de estas funciones, es bastante rápida si se sabe que estas claves están registradas en orden alfabético ascendente.

OkMaterial; función que se utiliza para verificar que esté registrada la clave de un material (materia prima o subensamble) y para localizar su número de registro. Para utilizarlo, el primer parámetro es la cadena que contiene la clave del artículo y el segundo es el parámetro por variable tipo entero en donde se memorizará el número de registro de dicha clave.

Ej: Si está registrado el material de clave "TOR1", mostrar en pantalla su número de registro.

```
IF OkMaterial('TOR1',numero_reg) THEN
  Write(numero_reg);
```

OkProducto; función que tiene como fin el de verificar que la clave de un producto terminado esté registrada y el de

localizar el número de registro de dicho producto. Su primer parámetro es la cadena que representa la clave del artículo y el segundo será el parámetro por variable tipo entero que almacenará el número de registro del producto terminado.

Ej: Si el producto terminado de clave "SDA1" existiera, presentar en pantalla su número de registro y, si no, mostrar un mensaje de error.

```
IF OkProducto('SDA1',numero_reg)
  THEN Write(numero_reg)
  ELSE MensajeErr('Clave no registrada.');
```

OkSubensamble; función utilizada para verificar que la clave de un subensamble esté registrada y para localizar el número de registro de dicha clave utilizando, a su vez, la función **OkMaterial**. Los parámetros utilizados son: la cadena que contiene la clave del subensamble y la variable depositaria del número de registro del mismo.

Ej: Mostrar el número de registro del subensamble de clave "[ARON".

```
IF OkSubensamble('[ARON',numero_reg)
  THEN Write(numero_reg);
```

OkMaquila; función que verifica que la clave del producto de maquila se encuentre registrada y que localiza su número de registro, utilizando la función **OkMaterial**. Se requieren de dos parámetros para usarla y éstos son: la cadena que contiene la clave del producto de maquila y el entero que recibirá el número de registro.

Ej: Si la clave del producto de maquila "^BAFLE1" no está registrada, salir del procedimiento actual.

```
IF NOT OkMaquila('^BAFLE1',numero_reg) THEN Exit;
```

OkDesProduccion; función para verificar que la descripción de las partes para producción de un subensamble o producto terminado estén registradas y para localizar su número de registro. Al igual que las funciones anteriores, requiere de la cadena que contiene la clave del artículo y el parámetro por variable que recibirá el número de registro.

Ej: Presentar el número de registro de la descripción de producción del subensamble de clave "[PUENTE" si es que está registrado.

```
IF OkDesProduccion('[PUENTE',numero_reg_des)
  THEN Write(numero_reg_des);
```

OkDesMaquila; función de utilidad para verificar y localizar la clave de un producto de maquila (subensamble). Los parámetros necesarios son la cadena que contiene la clave del artículo y el parámetro por variable que recibirá el número de registro.

Ej: Localizar el número de registro de la descripción de maquila del subensamble de clave "^BAFLE1".

```
IF OkDesMaquila('^BAFLE1',numero_reg_des)
  THEN Write(numero_reg_des);
```

OkPartesProduccion; función que, después de verificar que esté registrada la clave del artículo de producción (utilizando la función OkDesProduccion), encuentra los números de registro y cantidades requeridas de todas las partes que lo componen. Utiliza tres parámetros: la clave del artículo de producción, el parámetro por variable entero para el número de registro de la descripción de producción y, por último, el parámetro por valor tipo registro que almacenará la localización y cantidad requerida de cada material que lo forma.

Ej: Encontrar las partes de producción del artículo de clave "SDA1".

```
var_bool := OkPartesProduccion('SDA1',
                    numero_reg_des,partes_usadas);
```

OkPartesMaquila; función que, en forma análoga a la anterior, verifica que esté registrada la descripción del producto de maquila (usando la función OkDesMaquila) y después, localiza las partes integrantes del mismo. Necesita la clave del producto de maquila y dos parámetros por variable: el número de registro de la descripción y el tipo registro para las partes componentes.

Ej: Colocar en la variable tipo registro partes_usadas los componentes del subensamble o producto de maquila de clave "^TAPAN", si es que está registrado.

```
var bool := OkPartesMaquila('^TAPAN',
                    numero_reg_des,partes_usadas);
```

OkProveedor; función que verifica que la clave del proveedor esté registrada y localiza dicha clave. Requiere la clave a buscar y el parámetro por variable para el número de registro de la misma.

Ej: Localizar el número de registro del proveedor de clave "IMP_TOR".

```
var_bool := OkProveedor('IMP_TOR',numero_reg);
```

OkDistribuidor; función utilizada para verificar que la clave del distribuidor esté registrada, y también para localizar el número de registro de dicha clave. Los dos parámetros necesarios son: la clave del distribuidor y el parámetro por variable que recibirá la localización de dicha clave.

Ej: Indicar si la clave de distribuidor "MEX_LUZ" está registrada.

```
IF OkDistribuidor('MEX_LUZ',numero_reg)
  THEN Write('Clave de distribuidor registrada.')
  ELSE Write('Clave de distribuidor no reg.');
```

OkCambioPrecios; función que verifica que los precios de los artículos, del pedido en uso, no hayan cambiado. Para su utilización se requiere de un pedido a analizar en la variable externa a la función pedido.

Ej: Dar un aviso si ha cambiado al menos un precio de algún artículo del pedido en uso.

```
IF NOT OkCambioPrecios
  THEN MensajeErr('Hay cambio en los precios.');
```

OkImprimirForma; función de utilidad para imprimir formatos el número de veces deseado. Para permitir una gran flexibilidad en los formatos de salida por impresora, se desarrolló esta función que, en base a claves localizadas en un fichero de texto ASCII, permite la inserción de campos de la base de datos, comentarios, etc. en dichos formatos. A continuación se explica la forma de crear estos formatos:

+ Para crear el formato, sólo se necesita de un editor o procesador de textos que trabaje en código ASCII.

+ Todos los caracteres que se escriban en el texto (a excepción del carácter "@") serán enviados a la impresora tal y como están escritos. Esto permite la utilización de mandos para la impresora para obtener atributos especiales como lo son: subrayados, impresión en negrita, caracteres a doble ancho o reducidos, etc.

+ Para poner comentarios desde el teclado antes de la impresión del formato, se coloca en el lugar deseado el mando "@C", seguido de líneas de subrayado para indicar el tamaño del comentario.

+ Si se desea que aparezca el letrero "ORIGINAL" en la primera impresión y "COPIA" en las siguientes, el mando a utilizar será "@E", seguido de líneas de subrayado para indicar el tamaño máximo del letrero.

+ Para utilizar y dar una localización a la información almacenada en la base de datos y otros datos, existe una clave para cada tipo de reporte (razón por la cual existe una función CadenaExterna para cada función OkImprimirForma). El procedimiento será colocar en el lugar deseado el mando "@" seguido de la clave asignada y de las líneas de subrayado que indicarán el tamaño máximo para la información a colocar.

Los parámetros requeridos para esta función son: el nombre completo del fichero de texto que contiene el formato y el número de copias a imprimir.

PROCEDIMIENTOS PARA LECTURAS ESPECIFICAS DEL DISCO.

Estos procedimientos se utilizan para leer del disco registros completos y pasarlos a la memoria de la computadora. A continuación se muestran estos procedimientos.

CARGA.BBL (Fichero que contiene el código fuente que se muestra.)

```
PROCEDURE CargaProveedor(      numero: Integer;                    { Lee del disco para depositar en el parámetro var }
                         VAR des_pro: TDesPro );                   { 'des_pro' la descripción del proveedor 'numero'. }
  BEGIN
    WITH des_pro DO
      BEGIN
        LeerCad(numero,1,clave);     LeerCad(numero,2,nombre);
        LeerCad(numero,3,direc);     LeerCad(numero,4,telef);
        LeerCad(numero,5,ciudad);    LeerCad(numero,6,estado);
        LeerCad(numero,7,c_p);       LeerByte(numero,8,cdp);
        LeerReal(numero,9,saldo);
      END;
  END;
```

```
PROCEDURE CargaDistribuidor(      numero: Integer;                 { Lee del disco para depositar en el parámetro var }
                            VAR des_dis: TDesDis  );               { 'des_dis' la descripción del distribuidor 'numero'. }
  BEGIN
    WITH des_dis DO
      BEGIN
        LeerCad(numero,1,clave);     LeerCad(numero,2,nombre);
        LeerCad(numero,3,direc);     LeerCad(numero,4,telef);
        LeerCad(numero,5,ciudad);    LeerCad(numero,6,estado);
        LeerCad(numero,7,c_p);       LeerCad(numero,8,eob_a);
        LeerByte(numero,9,cdp);      LeerByte(numero,10,d);
        LeerByte(numero,11,dp);      LeerReal(numero,12,saldo);
        LeerReal(numero,13,l_cred); LeerReal(numero,14,cred);
      END;
  END;
```

```
PROCEDURE CargaPedido(      numero: Integer;                       { Lee del disco para depositar en el parámetro var }
                       VAR loc_ped: Integer;                       { 'pedido' y 'loc_ped' los datos y la localización }
                       VAR  pedido: TPed     );                    { del pedido con folio 'numero'. }
  VAR
    cont,                                                          { Contador para las partidas del pedido. }
    loc_part,                                                      { Localización de las partidas del pedido. }
    ver_ped: Integer;                                              { Utilizado para verificar que el folio del pedido }
                                                                   { coincida con el folio especificado en cada partida. }
  BEGIN
    WITH pedido DO
```

```
    BEGIN
      loc_ped := numero - num_ped_i + 1 + 9999 * Ord(numero < num_ped_i);          ( Localiza el pedido. )
      na := 4;
      LeerEnte(loc_ped,1,num);          LeerCar(loc_ped,2,cerr);                  ( Lectura de los datos generales )
      LeerFecha(loc_ped,3,fech);        LeerByte(loc_ped,4,can);                  ( del pedido. )
      LeerFecha(loc_ped,5,fech_can);    LeerFecha(loc_ped,6,fech_he);
      LeerFecha(loc_ped,7,fech_ent);    LeerCad(loc_ped,8,clave);
      LeerCad(loc_ped,9,nombre);        LeerCad(loc_ped,10,direc);
      LeerCad(loc_ped,11,telef);        LeerCad(loc_ped,12,ciudad);
      LeerCad(loc_ped,13,estado);       LeerCad(loc_ped,14,c_p);
      LeerCad(loc_ped,15,eab_a);        LeerByte(loc_ped,16,cdp);
      LeerByte(loc_ped,17,d);           LeerByte(loc_ped,18,dp);
      LeerFecha(loc_ped,19,fech_cob);   LeerCar(loc_ped,20,ap);
      LeerReal(loc_ped,21,importe);     LeerReal(loc_ped,22,pag_ant);
      LeerFecha(loc_ped,23,fech_pa);    LeerReal(loc_ped,24,pag_tot);
      LeerFecha(loc_ped,25,fech_pt);    LeerEnte(loc_ped,26,factura);
      LeerFecha(loc_ped,27,fech_fac);   LeerEnte(loc_ped,28,n_p);
      na := 5;
      FOR cont := 1 TO n_p DO                                                   ( Lee las partidas del pedido. )
        WITH part[cont] DO
          BEGIN
            loc_part := ped_part[loc_ped] + cont - 1;
            LeerEnte(loc_part,1,ver_ped);
            IF ver_ped <> num THEN MensajeErr('Error en el registro de pedidos. No corresponde la partida.');
            LeerCad(loc_part,2,artic);
            LeerEnte(loc_part,3,cant);
            LeerCad(loc_part,4,desc);
            LeerReal(loc_part,5,precio);
          END;
    END;
  END;
```

```
PROCEDURE CargaEntrada(     numero: Integer;                    ( Lee del disco para depositar en el parámetro var )
                       VAR loc_ent: Integer;                    ( 'entrada' y 'loc_ent' los datos y la localización )
                       VAR entrada: TEnt     );                 ( de la entrada con folio 'numero'. )
  VAR
    cont,                                                       ( Contador para las partidas de la entrada. )
    loc_part,                                                   ( Localización de las partidas de la entrada. )
    ver_ent: Integer;                                           ( Utilizado para verificar que el folio de la entrada )
                                                                ( coincida con el folio especificado en cada partida. )
  BEGIN
    WITH entrada DO
      BEGIN
        loc_ent := numero - num_ent_i + 1 + 9999 * Ord(numero < num_ent_i);     ( Localiza la entrada. )
        na := 4;
        LeerEnte(loc_ent,1,num);                                                ( Lectura de los datos generales )
        LeerCar(loc_ent,2,origen);                                              ( de la entrada. )
        LeerByte(loc_ent,3,razon);
        LeerCad(loc_ent,4,clave);
        LeerEnte(loc_ent,5,pedido);
```

```
        LeerFecha(loc_ent,6,fech);
                LeerFecha(loc_ent,7,fech_pag);
        LeerFecha(loc_ent,8,fech_pt);
        LeerReal(loc_ent,9,importe);
        LeerEnte(loc_ent,10,n_p);
        na := 5;
        FOR cont := 1 TO n_p DO                                    { Lee las partidas de la entrada. }
          WITH part[cont] DO
            BEGIN
              loc_part := ent_part[loc_ent] + cont - 1;
              LeerEnte(loc_part,1,ver_ent);
              IF ver_ent <> numero THEN MensajeErr('Error en el registro de entradas. No corresponde la partida.');
              LeerCad(loc_part,2,artic);
              LeerEnte(loc_part,3,cant);
              LeerReal(loc_part,4,precio);
            END;
      END;
  END;
```

```
PROCEDURE CargaSalida(     numero: Integer;                { Lee del disco para depositar en el parámetro var }
                      VAR loc_sal: Integer;                { 'salida' y 'loc_sal' los datos y la localización }
                      VAR  salida: TSal     );             { de la salida con folio 'numero'. }
  VAR
    cont,                                                  { Contador para las partidas de la salida. }
    loc_part,                                              { Localización de las partidas de la salida. }
    ver_sal: Integer;                                      { Utilizado para verificar que el folio de la salida }
                                                           { coincida con el folio especificado en cada partida. }
  BEGIN
    WITH salida DO
      BEGIN
        loc_sal := numero - num_sal_i + 1 + 9999 * Ord(numero < num_sal_i);      { Localiza la salida. }
        na := 4;
        LeerEnte(loc_sal,1,nua);                                                 { Lectura de los datos generales }
        LeerCar(loc_sal,2,origen);                                               { de la salida. }
        LeerByte(loc_sal,3,razon);
        LeerCad(loc_sal,4,clave);
        LeerEnte(loc_sal,5,pedido);
        LeerFecha(loc_sal,6,fech);
        LeerReal(loc_sal,7,importe);
        LeerEnte(loc_sal,8,n_p);
        na := 5;
        FOR cont := 1 TO n_p DO                                                  { Lee las partidas de la salida. }
          WITH part[cont] DO
            BEGIN
              loc_part := sal_part[loc_sal] + cont - 1;
              LeerEnte(loc_part,1,ver_sal);
              IF ver_sal <> numero THEN MensajeErr('Error en el registro de salidas. No corresponde la partida.');
              LeerCad(loc_part,2,artic);
              LeerEnte(loc_part,3,cant);
              LeerReal(loc_part,4,precio);
```

```
        END;
    END;
END;
```

```
PROCEDURE CargaAjuste(      numero: Integer;                    { Lee del disco para depositar en el parámetro var }
                      VAR ajuste: TAju     );                   { 'ajuste' los datos del ajuste con número de registro }
BEGIN                                                           { 'numero'. }
  WITH ajuste DO
    BEGIN
      na := 4;
      LeerEnte(numero,1,num);
      LeerCar(numero,2,origen);
      LeerByte(numero,3,razon);
      LeerCad(numero,4,artic);
      LeerEnte(numero,5,cant);
      LeerReal(numero,6,precio);
      LeerFecha(numero,7,fech);
    END;
END;
```

```
PROCEDURE CargaCobro(     numero: Integer;                    { Lee del disco para depositar en el parámetro var }
                      VAR cobro: TCob      );                 { 'cobro' los datos del cobro del registro 'numero'. }
  BEGIN
    WITH cobro DO
      BEGIN
        na := 4;
        LeerCad(numero,1,clave);
        LeerFecha(numero,2,fech);
        LeerCad(numero,3,forma_pag);
        LeerReal(numero,4,importe);
      END;
  END;
```

```
PROCEDURE CargaPago(    numero: Integer;                     { Lee del disco para depositar en el parámetro var }
                     VAR pago: TPag      );                  { 'pago' los datos del pago del registro 'numero'. }
  BEGIN
    WITH pago DO
      BEGIN
        na := 4;
        LeerCad(numero,1,clave);
        LeerFecha(numero,2,fech);
        LeerCad(numero,3,forma_pag);
        LeerReal(numero,4,importe);
      END;
  END;
```

Estos procedimientos pueden localizar en forma muy rápida ualquier dato solicitado de la siguiente forma:

+ Para el caso de registros foliados; sabiendo que deberán estar registrados todos los folios en forma consecutiva desde el primero hasta el último en la base de datos, mediante una sencilla operación se puede saber en donde se localiza un registro de folio conocido.

+ Para la lectura de alguna descripción en base a una clave; antes de utilizar estos procedimientos, se debe localizar el número de registro en la memoria RAM de dicha clave y después utilizar el procedimiento de lectura enviando el número encontrado.

CargaPedido; procedimiento que lee del disco todos los datos del pedido del folio dado y su número de registro del archivo. El primer parámetro es el número de folio del pedido y los siguientes son los parámetros por variable que almacenarán el número de registro y los datos del pedido.

CargaEntrada; utilizado para leer del disco los datos de la entrada de folio conocido y para localizar su número de registro del archivo. El primer parámetro es el folio de la entrada buscada, seguido de dos parámetros por variable en donde se almacenará el número de registro y los datos de dicha entrada.

CargaSalida; lee del disco los datos de la salida de folio conocido y localiza su número de registro del archivo. Como primer parámetro se envía el número del folio de la entrada y los dos parámetros por variable: número de registro y datos de la salida.

CargaCobro; se utiliza para leer del disco los datos del cobro cuyo número de registro del archivo es conocido. Se utiliza el número de registro del archivo y el parámetro por variable que almacenará los datos del cobro.

CargaPago; utilizado para leer del disco los datos del pago con número de registro del archivo conocido. Para usarlo se envía como primer parámetro el número de registro del archivo y el parámetro por variable para almacenar los datos del pago.

CargaAjuste; usado para leer del disco los datos del ajuste con número de registro conocido. Se requiere del número de registro del archivo y del parámetro por variable para los datos del ajuste.

CargaProveedor; lee del disco la descripción del proveedor con número de registro conocido. El primer parámetro es el número de registro de dicho proveedor y el segundo será el parámetro por variable depositario de la información.

CargaDistribuidor; procedimiento que lee del disco la descripción del distribuidor con número de registro conocido. El primer parámetro será el número de registro, seguido del parámetro por variable que recibirá dicha descripción.

PROCEDIMIENTOS QUE MUESTRAN PANTALLAS DE DATOS.

Estos procedimientos muestran en pantalla la información contenida en una variable tipo registro. Para ello utilizan desde el renglón tres al veinte de la pantalla. A continuación se muestran los listados de dichos procedimientos.

MOSTRAR.BBL (Fichero que contiene el código fuente que se muestra.)

```
PROCEDURE MostrarCabezaPedido;                    ( Muestra la cabecera del pedido que esté en la )
                                                  ( variable 'pedido'. )
  BEGIN
    WITH pedido DO
      BEGIN
        GotoXY(1,3);
        ClrEol; WriteLn('                                                                          ');
        ClrEol; WriteLn(' Clave:          Nombre:                                                  ');
        ClrEol; WriteLn('                                                                          ');
        ClrEol; WriteLn('    Dirección:                                        Tel:                ');
        ClrEol; WriteLn('       Ciudad:                                                            ');
        ClrEol; WriteLn('       Estado:                                                            ');
        ClrEol; WriteLn(' Código postal:                                                           ');
        ClrEol; WriteLn('  Embárquese a:                                                           ');
        ClrEol; WriteLn('                                                                          ');
        ClrEol; WriteLn(' Descuento:   %                        Descuento por pronto pago:   %     ');
        ClrEol; WriteLn('                                                                          ');
        ClrEol; WriteLn(' Condiciones de pago:     Días                                            ');
        ClrEol; WriteLn('                                                                          ');
        ClrEol; WriteLn;
        ClrEol; WriteLn;
        ClrEol; WriteLn;
        ClrEol; WriteLn;
        ClrEol; WriteLn;
        GotoXY(10,4); Write(clave);
        GotoXY(32,4); Write(JusCad(nombre,TAM_NOMBRE));
        GotoXY(18,6); Write(JusCad(direc,TAM_DIREC));
        GotoXY(68,6); Write(JusCad(telef,TAM_TELEF));
        GotoXY(18,7); Write(JusCad(ciudad,TAM_CIUDAD));
        GotoXY(18,8); Write(JusCad(estado,TAM_ESTADO));
        GotoXY(18,9); Write(JusCad(c_p,TAM_C_P));
        GotoXY(18,10); Write(JusCad(emb_a,TAM_EMB_A));
        GotoXY(13,12); Write(d:2);
        GotoXY(74,12); Write(dp:2);
        GotoXY(24,14); Write(cdp:3);
      END;
  END;
```

```
PROCEDURE MostrarCabezaPedidoExtra;                    { Muestra la cabecera del pedido que esté en la }
                                                       { variable 'pedido' y algunos datos que estén en la }
  BEGIN                                                { variable 'des_dis'. }
    WITH pedido DO
      BEGIN
        GotoXY(1,3);
        ClrEol; WriteLn('                                                                              ');
        ClrEol; WriteLn('  Clave:              Nombre:                                                 ');
        ClrEol; WriteLn('                                                                              ');
        ClrEol; WriteLn('     Dirección:                                          Tel:                 ');
        ClrEol; WriteLn('        Ciudad:                                                               ');
        ClrEol; WriteLn('        Estado:                                                               ');
        ClrEol; WriteLn('  Código postal:                                                              ');
        ClrEol; WriteLn('   Embárquese a:                                                              ');
        ClrEol; WriteLn('                                                                              ');
        ClrEol; WriteLn('  Descuento:   %                                  Descuento por pronto pago:  %');
        ClrEol; WriteLn('                                                                              ');
        ClrEol; WriteLn('  Condiciones de pago:     Dias          Saldo (acree.): $                MN ');
        ClrEol; WriteLn('                                                                              ');
        ClrEol; WriteLn('  Crédito actual: $                MN   Crédito límite: $                MN ');
        ClrEol; WriteLn('                                                                              ');
        ClrEol;
        WriteLn; ClrEol;
        WriteLn; ClrEol;
        GotoXY(10,4); Write(clave);
        GotoXY(32,4); Write(JusCad(nombre,TAM_NOMBRE));
        GotoXY(18,6); Write(JusCad(direc,TAM_DIREC));
        GotoXY(68,6); Write(JusCad(telef,TAM_TELEF));
        GotoXY(18,7); Write(JusCad(ciudad,TAM_CIUDAD));
        GotoXY(18,8); Write(JusCad(estado,TAM_ESTADO));
        GotoXY(18,9); Write(JusCad(c_p,TAM_C_P));
        GotoXY(18,10); Write(JusCad(emb_a,TAM_EMB_A));
        GotoXY(13,12); Write(d:2);
        GotoXY(74,12); Write(dp:2);
        GotoXY(24,14); Write(cdp:3);
        GotoXY(58,14); Write(JusReal(des_dis.saldo,14,2));
        GotoXY(21,16); Write(JusReal(des_dis.cred,14,2));
        GotoXY(58,16); Write(JusReal(des_dis.l_cred,14,2));
      END;
  END;
```

```
PROCEDURE MostrarPartidas;                             { Muestra las partidas del pedido que estén en la }
                                                       { variable 'pedido'. }
  VAR
    cont: Integer;                                     { Contador de partida. }
    tot_part,                                          { Total de las partida (importe de la partida). }
    suma,                                              { Suma de los importes de las partidas. }
    descuento,                                         { Importe del descuento. }
    total,                                             { Total antes de I.V.A. }
```

```
  a_pagar: Real;                                    { Cantidad a pagar. }

BEGIN
  WITH pedido DO
    BEGIN
      GotoXY(1,3);
      ClrEol; WriteLn('
      ClrEol; WriteLn('  #P  | ARTIC | CANT | DESCRIPCION | PRECIO | IMPORTE  ');
      ClrEol; WriteLn('
      ClrEol; WriteLn('   1
      ClrEol; WriteLn('   2
      ClrEol; WriteLn('   3
      ClrEol; WriteLn('   4
      ClrEol; WriteLn('   5
      ClrEol; WriteLn('   6
      ClrEol; WriteLn('   7
      ClrEol; WriteLn('   8
      ClrEol; WriteLn('   9
      ClrEol; WriteLn('  10
      ClrEol; WriteLn('
      ClrEol; WriteLn('                                          SUMA
      ClrEol; WriteLn('                          I.V.A.          % DESC
      ClrEol; WriteLn('                       TOT/I.V.A.         TOTAL
      ClrEol; WriteLn('
      GotoXY(53,18); Write(JusEnte(d,2));
      IF n_p > 0 THEN
        BEGIN
          suma := 0;
          FOR cont := 1 TO n_p DO                   { Presenta las partidas y va acumulando en 'suma' }
            WITH part[cont] DO                      { los totales de las partidas. }
              BEGIN
                GotoXY(7,5 + cont); Write(artic);
                GotoXY(16,5 + cont); Write(JusEnte(cant,6));
                GotoXY(24,5 + cont); Write(desc);
                GotoXY(47,5 + cont); Write(JusReal(precio,14,2));
                tot_part := precio * cant;
                GotoXY(64,5 + cont); Write(JusReal(tot_part,14,2));
                suma := suma + tot_part;
              END;
          GotoXY(64,17); Write(JusReal(suma,14,2));
          descuento := suma * d/100;                { Calcula el importe del descuento. }
          total := suma - descuento;
          GotoXY(64,18); Write(JusReal(descuento,14,2));
          a_pagar := total * (1 + IVA);             { Calcula la cantidad a pagar. }
          IF Abs(a_pagar - importe) >= 1 THEN       { En caso de que 'importe' registrado no coincida con }
            MensajeErr('El importe presentado no es correcto');  { lo calculado, se presenta un mensaje de error. }
          GotoXY(64,19); Write(JusReal(importe/(1 + IVA),14,2));
          GotoXY(33,18); Write(JusReal(importe/(1 + 1/IVA),14,2));
```

```
        GotoXY(33,19); Write(JusReal(importe,14,2));
      END;
    END;
  END;
```

```
PROCEDURE MostrarOtros;                                  { Muestra otros datos del pedido que estén en la }
                                                         { variable 'pedido'. }
  BEGIN
    WITH pedido DO
      BEGIN
        GotoXY(1,3);
        ClrEol; WriteLn('                                                                              ');
        ClrEol; WriteLn('   Pedido:                          Fecha de pedido:                          ');
        ClrEol; WriteLn('                                                                              ');
        ClrEol; WriteLn('   Factura:                         Fecha de factura:                         ');
        ClrEol; WriteLn('  ----------------------------------------------------------------------------');
        ClrEol; WriteLn('   Cerrado:                                                                   ');
        ClrEol; WriteLn('  Cancelado:                        Fecha de cancelación:                     ');
        ClrEol; WriteLn('  ----------------------------------------------------------------------------');
        ClrEol; WriteLn('                                  Fecha en que se debe cobrar:                ');
        ClrEol; WriteLn('   Anticipo:                                Fecha anticipo:                   ');
        ClrEol; WriteLn('      Pago:                                    Fecha pago:                   ');
        ClrEol; WriteLn('                Se aplicó el descuento por pronto pago:                       ');
        ClrEol; WriteLn('  ----------------------------------------------------------------------------');
        ClrEol; WriteLn('     Fecha en que se entregó:                                                 ');
        ClrEol; WriteLn('  Fecha prometida de entrega:                                                 ');
        ClrEol; WriteLn('                                                                              ');
        ClrEol; WriteLn;
        ClrEol; WriteLn;
        GotoXY(14,4); Write(JusFolio(num));
        GotoXY(57,4); Write(fech);
        GotoXY(14,6); IF factura > 0 THEN Write(JusFolio(factura));
        GotoXY(57,6); Write(fech_fac);
        GotoXY(14,8); Write(cerr);
        GotoXY(14,9); Write(can);
        GotoXY(57,9); Write(fech_can);
        GotoXY(57,11); Write(fech_cob);
        GotoXY(14,12); Write(JusReal(pag_ant,14,2));
        GotoXY(57,12); Write(fech_pa);
        GotoXY(14,13); Write(JusReal(pag_tot,14,2));
        GotoXY(57,13); Write(fech_pt);
        GotoXY(57,14); Write(ap);
        GotoXY(32,16); Write(fech_ent);
        GotoXY(32,17); Write(fech_he);
      END;
  END;
```

```
PROCEDURE MostrarEntrada;                                  { Muestra los datos de la entrada que estén en la }
                                                           { variable 'entrada'. }
  VAR
    cont: Integer;                                         { Contador de partida. }
    tot_part,                                              { Total de partida. }
    suma:     Real;                                        { Suma de los totales de las partidas. }

  BEGIN
    WITH entrada DO
      BEGIN
        GotoXY(1,3);
        ClrEol; WriteLn('                                                                    ');
        ClrEol; WriteLn('  Entrada:        Clave del origen:          Pedidos:        Fecha:  ');
        ClrEol; WriteLn('   Origen:   Razón:      Fecha para pago:              Fecha de pago: ');
        ClrEol; WriteLn('                                                                    ');
        ClrEol; WriteLn('  #P  ARTIC   CANT      PRECIO        IMPORTE  ');
        ClrEol; WriteLn('  ');
        ClrEol; WriteLn('   1');
        ClrEol; WriteLn('   2');
        ClrEol; WriteLn('   3');
        ClrEol; WriteLn('   4');
        ClrEol; WriteLn('   5');
        ClrEol; WriteLn('   6');
        ClrEol; WriteLn('   7');
        ClrEol; WriteLn('   8');
        ClrEol; WriteLn('   9');
        ClrEol; WriteLn('  10');
        ClrEol; WriteLn('  ');
        ClrEol; WriteLn('                                  SUMA');
        ClrEol; WriteLn('                            TOTAL A PAGAR');
        ClrEol; WriteLn('  ');
        GotoXY(12,4); Write(JusFolio(num));
        GotoXY(36,4); Write(JusCad(clave,TAM_CLAVE));
        GotoXY(53,4); Write(JusFolio(pedido));
        GotoXY(70,4); Write(JusCad(fech,8));
        GotoXY(12,5); Write(origen);
        GotoXY(22,5); Write(JusEnte(razon,3));
        GotoXY(44,5); Write(JusCad(fech_pag,8));
        GotoXY(70,5); Write(JusCad(fech_pt,8));
        IF n_p > 0 THEN
          BEGIN
            suma := 0;
            FOR cont := 1 TO n_p DO                        { Presenta las partidas y va acumulando en 'suma' }
              WITH part[cont] DO                           { los totales de las partidas. }
                BEGIN
                  GotoXY(7,8 + cont); Write(artic);
                  GotoXY(16,8 + cont); Write(JusEnte(cant,6));
                  GotoXY(25,8 + cont); Write(JusReal(precio,14,2));
                  tot_part := precio * cant;
                  GotoXY(42,8 + cont); Write(JusReal(tot_part,14,2));
                  suma := suma + tot_part;
```

```
            END;
          GotoXY(42,20); Write(JusReal(suma,14,2));
          GotoXY(42,21); Write(JusReal(importe,14,2));
        END;
    END;
END;
```

```
PROCEDURE MostrarSalida;                          { Muestra los datos de la salida que estén en la }
                                                  { variable 'salida'. }
  VAR
    cont: Integer;                                { Contador de partida. }
    tot_part,                                     { Total de partida. }
    suma:   Real;                                 { Suma de los totales de las partidas. }

  BEGIN
    WITH salida DO
      BEGIN
        GotoXY(1,3);
        ClrEol; WriteLn('                                                                ');
        ClrEol; WriteLn('  Salida:      Clave del origen:        Pedido:          Fecha:  ');
        ClrEol; WriteLn('  Origen:    Razón:                                             ');
        ClrEol; WriteLn('                                                                ');
        ClrEol; WriteLn('  #P  | ARTIC | CANT |    PRECIO    |    IMPORTE    |           ');
        ClrEol; WriteLn('      |        |      |              |               |           ');
        ClrEol; WriteLn('   1  |        |      |              |               |           ');
        ClrEol; WriteLn('   2  |        |      |              |               |           ');
        ClrEol; WriteLn('   3  |        |      |              |               |           ');
        ClrEol; WriteLn('   4  |        |      |              |               |           ');
        ClrEol; WriteLn('   5  |        |      |              |               |           ');
        ClrEol; WriteLn('   6  |        |      |              |               |           ');
        ClrEol; WriteLn('   7  |        |      |              |               |           ');
        ClrEol; WriteLn('   8  |        |      |              |               |           ');
        ClrEol; WriteLn('   9  |        |      |              |               |           ');
        ClrEol; WriteLn('  10  |        |      |              |               |           ');
        ClrEol; WriteLn('      |        |      |              |               |           ');
        ClrEol; WriteLn('                                  SUMA                          ');
        ClrEol; WriteLn('                          TOTAL A PAGAR                         ');
        ClrEol; WriteLn('                                                                ');
        GotoXY(12,4); Write(JusFolio(nua));
        GotoXY(36,4); Write(JusCad(clave,TAM_CLAVE));
        GotoXY(53,4); Write(JusFolio(pedido));
        GotoXY(70,4); Write(JusCad(fech,8));
        GotoXY(12,5); Write(origen);
        GotoXY(22,5); Write(JusEnte(razon,3));
        IF n_p > 0 THEN
          BEGIN
            suma := 0;
            FOR cont := 1 TO n_p DO               { Presenta las partidas y va acumulando en 'suma' }
              WITH part[cont] DO                  { los totales de las partidas. }
                BEGIN
```

```
                GotoXY(7,8 + cont); Write(artic);
                GotoXY(16,8 + cont); Write(JusEnte(cant,6));
                GotoXY(25,8 + cont); Write(JusReal(precio,14,2));
                tot_part := precio * cant;
                GotoXY(42,8 + cont); Write(JusReal(tot_part,14,2));
                suma := suma + tot_part;
              END;
          GotoXY(42,20); Write(JusReal(suma,14,2));
          GotoXY(42,21); Write(JusReal(importe,14,2));
        END;
    END;
END;
```

Estos procedimientos hacen uso de variables externas a ellos que deben contener la información a mostrar. Para su utilización no requieren de parámetros.

MostrarCabezaPedido; muestra en pantalla los datos de la cabecera (datos generales) del pedido en uso. Estos datos se refieren principalmente a información referente al cliente y a las condiciones específicas de cada pedido. Muestra los datos contenidos en la variable pedido.

MostrarCabezaPedidoExtra; cumple la tarea del procedimiento anterior y además muestra algunos datos extras del distribuidor como, por ejemplo, su saldo actual. Muestra los datos contenidos en las variables pedido y des_dis.

MostrarPartidas; muestra las partidas y los totales del pedido. Muestra los datos de la variable pedido.

MostrarOtros; presenta en pantalla los datos del estado actual del pedido. los datos que se muestran, por ejemplo, son: la fecha en que se entregó el pedido, el número de factura asignada, etc. Muestra los datos contenidos en la variable pedido.

MostrarEntrada; presenta en pantalla todos los datos de la entrada, incluyendo sus partidas y sus totales. Usa la variable de nombre entrada.

MostrarSalida; muestra en pantalla todos los datos de una salida contenidos en la variable salida.

OTRO PROCEDIMIENTO.

Este es un procedimiento que, por no haber otro similar, se presenta en una sección por separado. A continuación se muestra el código fuente del mismo.

CAMBIAR.BBL (Fichero que contiene el código fuente que se muestra.)

```
{PROCEDURE CambiarPrecios;                          ( Actualiza los precios de un pedido que está contenido en )
                                                    ( la variable 'pedido'. )
  VAR
    algun_cambio: Boolean;                          ( Indica que se ha encontrado algún cambio de precio. )
    cont: Integer;                                  ( Contador de partida. )
    suma: Real;                                     ( Suma de los importes de las partidas. )
    producto,                                       ( Indica que es un producto terminado. )
    camb_precio: Array[1..MAX_N_P] OF Boolean;      ( Indica que dicho producto presenta un cambio de precio. )
    loc_artic: Array[1..MAX_N_P] OF Integer;        ( Memoriza la localización del artículo. )

  BEGIN
    FillChar(producto,SizeOf(producto),0);                  ( Por defecto se indicará que no son productos term. )
    FillChar(camb_precio,SizeOf(camb_precio),0);            ( Por def. se indicará que no hay cambio de precios. )
    WITH pedido DO
      BEGIN
        algun_cambio := FALSE;
        FOR cont := 1 TO n_p DO                             ( Se hará el análisis para todas las partidas. )
          WITH part[cont] DO
            IF OkProducto(artic,loc_artic[cont])
              THEN                                          ( Si se ha localizado el artículo como un producto )
                BEGIN                                       ( terminado, memoriza que es un producto terminado. )
                  producto[cont] := TRUE;
                  IF precio <> pnt_pt[loc_artic[cont]]^.precio_v THEN     ( Si se encuentra que hay algún cambio en el )
                    BEGIN                                                 ( precio, el indicador de cambio de precio )
                      algun_cambio := TRUE;                               ( general y el de dicha partida se hacen )
                      camb_precio[cont] := TRUE;                          ( verdaderos. )
                      MensajeErr( 'El producto ' + artic + ' cambia de ' +
                                  JusReal(precio,14,2) + ' a ' + JusReal(pnt_pt[loc_artic[cont]]^.precio_v,14,2) );
                    END;
                END
              ELSE
                IF OkMaterial(artic,loc_artic[cont]) THEN               ( Si se ha localizado el artículo como un )
                  IF precio <> pnt_mp[loc_artic[cont]]^.precio_v THEN   ( material y si se encuentra que hay algún )
                    BEGIN                                               ( cambio en el precio, el indicador de )
                      algun_cambio := TRUE;                             ( cambio de precio general y el de dicha )
                      camb_precio[cont] := TRUE;                        ( partida se hacen verdaderos. )
                      MensajeErr( 'El material ' + artic + ' cambia de ' +
                                  JusReal(precio,14,2) + ' a ' + JusReal(pnt_mp[loc_artic[cont]]^.precio_v,14,2) );
                    END;
      IF algun_cambio
        THEN                                    ( Si se ha encontrado algún cambio de precios, se )
```

```
        BEGIN                                   { procede a la actualización de precios. }
          na := 5;
          bas[na].nom_arch := nom_arch_ped_p;
          AbrirArchivo;                         { Se abre la base de datos de pedidos por partida. }
          suma := 0;
          FOR cont := 1 TO n_p DO               { Se repite la operación para todas las partidas. }
            WITH part[cont] DO
              BEGIN
                IF camb_precio[cont] THEN       { Si en la partida en cuestión se registró cambio de precio, }
                  BEGIN                         { se actualiza el precio unitario ya sea un p.t. o material }
                    IF producto[cont]           { tanto en la memoria RAM como en la base de datos. }
                      THEN precio := pnt_pt[loc_artic[cont]]^.precio_v
                      ELSE precio := pnt_mp[loc_artic[cont]]^.precio_v;
                    EscribirReal(ped_part[loc_ped] + cont - 1,5,precio);
                  END;
                suma := suma + precio * cant;   { Acumula los importes de todas las partidas. }
              END;
          CerrarArchivo;
          importe := suma * (1 - d/100);        { Reasigna el descuento otorgado en RAM. }
          importe := importe * (1 + IVA);       { Agrega el I.V.A. en RAM. }
          na := 4;
          EscribirReal(loc_ped,21,importe);     { Actualiza el importe en la base de datos. }
        END
      ELSE MensajeErr('No hay cambio en los precios.');
    END;
  END;
```

El procedimiento CambiarPrecios se utiliza para hacer la actualización (en memoria RAM y disco) de los precios unitarios de los artículos contenidos en un pedido. Este procedimiento no requiere de parámetros, ya que hace uso de la variable externa a él, de nombre pedido, que contiene los datos necesarios.

4.3.5 SUBPROGRAMA DE CARGA INICIAL.

Este procedimiento está constituido por un solo procedimiento de solapamiento, cuya función es leer del disco toda la información que se va a conservar en forma permanente en la memoria. El propósito de esto es principalmente el de agilizar, en la medida de lo posible, la operación del programa. El código fuente de este procedimiento es el que se muestra a continuación.

CARGA_IN.OVL (Fichero que contiene el código fuente que se muestra.)

```
Overlay PROCEDURE CargaInicial;                 ( Procedimiento de solapamiento de primer nivel que lee del disco los )
                                                ( datos que permanecerán en memoria RAM. )
  VAR
    num_regs,                                   ( Número de registros del archivo. )
    n_p,                                        ( Número de partidas. )
    loc: Integer;                               ( Localización del registro. )
    mp:  TMP;                                   ( Variable tipo materia prima. )
    pt:  TPT;                                   ( Variable tipo producto terminado. )
    prod: TProd;                                ( Variable tipo descripción de producción. )
    maq:  TMaq;                                 ( Variable tipo descripción de maquila. )

  BEGIN
    ClrScr;                                                  ( Aparece la pantalla de presentación. )
    WriteLn;
    WriteLn('                                                ');
    WriteLn('                                                ');
    WriteLn('                                                ');
    WriteLn('                                                ');
    WriteLn('                                                ');
    WriteLn('                                                ');
    WriteLn('                                                ');
    WriteLn('                                                ');
    WriteLn;
    WriteLn('          SISTEMA INTEGRADO PARA LA ADMINISTRACION POR COMPUTADORA');
    WriteLn;
    WriteLn('                                                ');
    WriteLn('                    VERSION:  R I M A, S. A.    ');
    WriteLn('                    (fabricante de iluminación) ');
    WriteLn('                                                ');
    WriteLn('                    POR: Juan Pablo Ruiz Velasco Márquez ');
    WriteLn('                                                ');
    WriteLn;
    WriteLn('                          ** ESPERE UN MOMENTO **');
    WriteLn;
    WriteLn('    ( En proceso de lectura desde el disco a la memoria de la computadora. )');

    Iniciacion;                                  ( Prepara el manejo de archivos. )
    nom_arch_des_dis := 'BASES\DES_DIS.DBF';     ( Establece el nombre de los archivos de base de datos a )
```

```
    nom_arch_des_pro := 'BASES\DES_PRO.DBF';          { utilizar en el programa. }
    nom_arch_ped     := 'BASES\R_PED.DBF';
    nom_arch_ped_p   := 'BASES\R_PED_P.DBF';
    nom_arch_ent     := 'BASES\R_ENT.DBF';
    nom_arch_ent_p   := 'BASES\R_ENT_P.DBF';
    nom_arch_sal     := 'BASES\R_SAL.DBF';
    nom_arch_sal_p   := 'BASES\R_SAL_P.DBF';
    nom_arch_aju     := 'BASES\R_AJU.DBF';
    nom_arch_pag     := 'BASES\R_PAG.DBF';
    nom_arch_cob     := 'BASES\R_COB.DBF';
    nom_arch_inv_mp  := 'BASES\INV_MP.DBF';
    nom_arch_inv_pt  := 'BASES\INV_PT.DBF';
    nom_arch_prod    := 'BASES\PROD.DBF';
    nom_arch_maq     := 'BASES\MAQ.DBF';
    nom_arch_cont    := 'BASES\CONT.DBF';
    na := 1;                                      { Opera con el archivo #1. }
    bas[na].nom_arch := nom_arch_inv_mp;          { Indica que se utilizará el archivo inventarios de m.p. }
    AbrirArchivo;                                 { Abre el archivo #1 (inventarios de m.p.) }
    num_mp := bas[na].num_regs;       { Almacena en 'num_mp' el número de materias primas registradas. }
    IF num_mp > MAX_M_P THEN          { Si se tienen más materias primas registradas que el máximo especificado 'MAX_MP' }
      BEGIN                           { se trabajará sólo con las primeras 'MAX_MP' materias primas. }
        num_mp := MAX_M_P;
        MensajeErr('Solamente se podrán cargar las primeras ' + JusEnte(num_mp,6) + ' materias primas.');
      END;
    GotoXY(1,23); ClrEol; WriteLn('--) Se están cargando las ',JusEnte(num_mp,5),' materias primas.');
    IF num_mp > 0
      THEN                            { Si se tienen registradas materias primas: }
        FOR loc := 1 TO num_mp DO     { Inicia un conteo con 'loc' desde uno hasta el número de m. p. }
          BEGIN
            WITH mp DO                { Trabajando con la variable mp. }
              BEGIN                   { Lectura del archivo #1 (inventario de m.p.) para el registro número 'loc'. }
                New(pnt_mp[loc]);             { Asigna un nuevo puntero para la materia prima siguiente. }
                LeerCad(loc,1,artic);         { Lee la clave del artículo, del campo #1. }
                LeerEnte(loc,2,exist);        { Lee las existencias del artículo, del campo #2. }
                LeerEnte(loc,3,ca_ped);       { Lee la cantidad pedida del artículo, del campo #3. }
                LeerEnte(loc,4,min);          { Lee el mínimo del artículo, del campo #4. }
                LeerEnte(loc,5,max);          { Lee el máximo del artículo, del campo #5. }
                LeerByte(loc,6,red);          { Lee las cifras de redondeo del artículo, del campo #6. }
                LeerCad(loc,7,desc);          { Lee la descripción del artículo, del campo #7. }
                LeerCad(loc,8,clave);         { Lee la clave del proveedor del artículo, del campo #8. }
                LeerReal(loc,9,precio_v);     { Lee el precio de venta del artículo, del campo #9. }
                LeerReal(loc,10,precio_ult);  { Lee el último costo del artículo, del campo #10. }
                LeerFecha(loc,11,fech);       { Lee la fecha del último costeo del artículo, del campo #11. }
              END;
            pnt_mp[loc]^ := mp;               { Coloca los datos en la memoria dinámica de la computadora. }
          END
      ELSE MensajeErr('Nota: no se tienen materias primas registradas.');
    na := 2;                          { Opera con el archivo #2. }
    bas[na].nom_arch := nom_arch_inv_pt;      { Indica que se utilizará el archivo de inventarios de p.t. }
    AbrirArchivo;                     { Abre el archivo #2 (inventarios de p.t.). }
    num_pt := bas[na].num_regs;       { Almacena en 'num_pt' el número de productos terminados registrados. }
    IF num_pt > MAX_P_T THEN          { Si se tienen más productos terminados registrados que 'MAX_PT' }
```

```
    BEGIN                                  { se trabajará sólo con los primeros 'MAX_PT' productos terminados. }
      num_pt := MAX_P_T;
      MensajeErr('Solamente se podrán cargar los primeros ' + JusEnte(num_pt,6) + ' productos terminados.');
    END;
  GotoXY(1,23); ClrEol; WriteLn('--> Se están cargando los ',JusEnte(num_pt,5),' productos terminados.');
  IF num_pt > 0
    THEN                                   { Si se tienen registrados productos terminados: }
      FOR loc := 1 TO num_pt DO            { Iniciar un conteo con 'loc' desde uno hasta el número de p. t. }
        BEGIN
          WITH pt DO                       { Trabajando con la variable pt. }
            BEGIN                          { Lectura del archivo #2 (inventario de p.t.) para el registro número loc. }
              New(pnt_pt[loc]);            { Asigna un nuevo puntero para el producto terminado siguiente. }
              LeerCad(loc,1,artic);        { Lee la clave del artículo, del campo #1. }
              LeerEnte(loc,2,exist);       { Lee las existencias del artículo, del campo #2. }
              LeerEnte(loc,3,ca_dis);      { Lee la cantidad disponible del artículo, del campo #3. }
              LeerEnte(loc,4,ca_ped);      { Lee la cantidad pedida del artículo, del campo #4. }
              LeerEnte(loc,5,min);         { Lee el mínimo del artículo, del campo #5. }
              LeerEnte(loc,6,max);         { Lee el máximo del artículo, del campo #6. }
              LeerByte(loc,7,red);         { Lee las cifras de redondeo del artículo, del campo #7. }
              LeerCad(loc,8,desc);         { Lee la descripción del artículo, del campo #8. }
              LeerReal(loc,9,precio_v);    { Lee el precio de venta del artículo, del campo #9. }
              LeerReal(loc,10,precio_ult); { Lee el último costo del artículo, del campo #10. }
              LeerFecha(loc,11,fech);      { Lee la fecha del último costeo del artículo, del campo #11. }
            END;
          pnt_pt[loc]^ := pt;              { Coloca los datos en la memoria dinámica de la computadora. }
        END
    ELSE MensajeErr('Nota: no se tienen productos terminados registrados.');
  GotoXY(1,23); ClrEol; WriteLn('--> Lectura de los contadores.');
  na := 3;                                 { Opera con el archivo #3. }
  bas[na].nom_arch := nom_arch_cont;       { Indica que se utilizará el archivo de contadores. }
  AbrirArchivo;                            { Abre el archivo #3 (contadores). }
  LeerEnte(1,2,num_devol);                 { Lee el folio de la devolución siguiente. }
  LeerEnte(2,2,num_fact);                  { Lee el folio de la factura siguiente. }
  LeerEnte(3,2,num_not_cred);              { Lee el folio de la nota de crédito siguiente. }
  LeerEnte(4,2,num_ord_prod);              { Lee el folio de la orden de producción siguiente. }
  LeerEnte(5,2,num_ord_maq);               { Lee el folio de la orden de maquila siguiente. }
  na := 4;                                 { Opera con el archivo #4. }
  bas[na].nom_arch := nom_arch_prod;       { Indica que se utilizará el archivo de partes de producción. }
  AbrirArchivo;                            { Abre el archivo #4 (partes de producción). }
  num_prod := bas[na].num_regs;            { Almacena en 'num_prod' el número de descr. de producción registradas. }
  IF num_prod > MAX_PROD THEN              { Si se tienen más desc. de prod. registradas que 'MAX_PROD' }
    BEGIN                                  { se trabajará sólo con las primeras 'MAX_PROD' desc. de producción. }
      num_prod := MAX_PROD;
      MensajeErr('Solamente se podrán cargar las primeras ' + JusEnte(num_prod,6) + ' descripciones de producción.');
    END;
  GotoXY(1,23); ClrEol; WriteLn('--> Se están cargando las ',JusEnte(num_prod,5),' descripciones de producción.');
  IF num_prod > 0
    THEN                                   { Si se tienen registradas descripciones de producción: }
      FOR loc := 1 TO num_prod DO          { Inicia un conteo con 'loc' desde uno hasta el número de desc. de prod. }
        BEGIN
          WITH prod DO                     { Trabajando con la variable 'prod'. }
            BEGIN                          { Lectura del archivo #4 (desc. producción) para el registro número 'loc'. }
```

```
        New(pnt_prod[loc]);              { Asigna un nuevo puntero para la descripción de producción siguiente. }
        LeerCad(loc,1,artic);            { Lee la clave del artículo, del campo #1. }
        LeerCad(loc,2,partes);           { Lee las partes componentes del artículo, del campo #2. }
      END;
    pnt_prod[loc]^ := prod;              { Coloca los datos en la memoria dinámica de la computadora. }
  END
ELSE MensajeErr('Nota: no se tienen descripciones registradas para producción.');
CerrarArchivo;                           { Cierra el archivo #3. }
bas[na].nom_arch := nom_arch_maq;        { Indica que se utilizará el archivo de partes de maquila. }
AbrirArchivo;                            { Abre el archivo #4 (partes de maquila). }
num_maq := bas[na].num_regs;             { Almacena en 'num_maq' el número de descripciones de maquila registradas. }
IF num_maq > MAX_MAQ THEN                { Si se tienen más desc. de maquila registradas que 'MAX_MAQ' }
  BEGIN                                  { se trabajará sólo con las primeras 'MAX_MAQ' desc. de maquila. }
    num_maq := MAX_MAQ;
    MensajeErr('Solamente se podrán cargar las primeras ' + JusEnte(num_maq,6) + ' descripciones de maquila.');
  END;
GotoXY(1,23); ClrEol; WriteLn('--> Se están cargando las ',JusEnte(num_maq,5),' descripciones de maquila.');
IF num_maq > 0
  THEN                                   { Si se tienen registradas descripciones de maquila: }
    FOR loc := 1 TO num_maq DO           { Inicia un conteo con 'loc' desde uno hasta 'num_maq'. }
      BEGIN
        WITH maq DO                      { Trabajando con la variable 'maq'. }
          BEGIN                          { Lectura del archivo #4 (desc. maquila) para el registro número 'loc'. }
            New(pnt_maq[loc]);           { Asigna un nuevo puntero para la descripción de maquila siguiente. }
            LeerCad(loc,1,artic);        { Lee la clave del artículo, del campo #1. }
            LeerCad(loc,2,partes);       { Lee las partes componentes del artículo, del campo #2. }
            LeerReal(loc,3,precio_ult);  { Lee el último costo de la maquila, del campo #3. }
            LeerFecha(loc,4,fech);       { Lee la fecha del último costeo de la maquila, del campo #4. }
          END;
        pnt_maq[loc]^ := maq;            { Coloca los datos en la memoria dinámica de la computadora. }
      END
  ELSE MensajeErr('Nota: no se tienen descripciones registradas para maquila.');
CerrarArchivo;                           { Cierra el archivo #4 }
bas[na].nom_arch := nom_arch_des_pro;    { Indica que se utilizará el archivo de descripción de proveedores. }
AbrirArchivo;                            { Abre el archivo #4 (descripción de proveedores). }
num_pro := bas[na].num_regs;             { Almacena en 'num_pro' el número de descripciones de proveedores. }
IF num_pro > MAX_PRO THEN                { Si se tienen más desc. de prov. registradas que 'MAX_PRO' }
  BEGIN                                  { se trabajará sólo con las primeras 'MAX_PRO' desc. de proveedores. }
    num_pro := MAX_PRO;
    MensajeErr('Solamente se podrán cargar las primeras ' + JusEnte(num_pro,6) + ' claves de proveedores.');
  END;
GotoXY(1,23); ClrEol; WriteLn('--> Se están cargando las ',JusEnte(num_pro,5),' claves de proveedores.');
IF num_pro > 0
  THEN                                   { Si se tienen registradas descripciones de proveedores: }
    BEGIN
      FOR loc := 1 TO num_pro DO         { Inicia un conteo con 'loc' desde uno hasta 'num_pro'. }
        LeerCad(loc,1,pro[loc])          { Lee la clave del proveedor, del campo #1. }
    END
  ELSE MensajeErr('Nota: no se tienen proveedores registrados.');
CerrarArchivo;                           { Cierra el archivo #4. }
bas[na].nom_arch := nom_arch_des_dis;    { Indica que se utilizará el archivo de descripción de distribuidores. }
AbrirArchivo;                            { Abre el archivo #4 (descripción de distribuidores). }
```

```
    num_dis := bas[na].num_regs;                { Almacena en 'num_dis' el número de descripciones de distribuidores. }
    IF num_dis > MAX_DIS THEN                   { Si se tienen más desc. de dist. registradas que 'MAX_DIS' }
      BEGIN                                     { se trabajará sólo con las primeras 'MAX_DIS' desc. de distribuidores. }
        num_dis := MAX_DIS;
        MensajeErr('Solamente se podrán cargar las primeras ' + JusEnte(num_dis,6) + ' claves de distribuidores.');
      END;
    GotoXY(1,23); ClrEol; WriteLn('--> Se están cargando las ',JusEnte(num_dis,5),' claves de distribuidores.');
    IF num_dis > 0
      THEN                                      { Si se tienen registradas descripciones de distribuidores: }
        BEGIN
          FOR loc := 1 TO num_dis DO            { Inicia un conteo con 'loc' desde uno hasta 'num_dis'. }
            LeerCad(loc,1,dis[loc])             { Lee la clave del distribuidor, del campo #1. }
        END
      ELSE MensajeErr('Nota: no se tienen distribuidores registrados.');
    CerrarArchivo;                              { Cierra el archivo #4. }
    bas[na].nom_arch := nom_arch_ped;           { Indica que se utilizará el archivo de registro de pedidos. }
    AbrirArchivo;                               { Abre el archivo #4 (registro de pedidos). }
    num_ped_i := 0;
    num_ped_f := 0;
    num_ped := bas[na].num_regs;                { Almacena en 'num_ped' el número de registros de pedidos. }
    IF num_ped > MAX_PED THEN                   { Si se tienen más pedidos registrados que 'MAX_PED' }
      BEGIN                                     { se trabajará sólo con los primeros 'MAX_PED' registros de pedidos. }
        num_ped := MAX_PED;
        MensajeErr('Solamente se podrá operar con los primeros ' + JusEnte(num_ped,6) + ' pedidos.');
      END;
    GotoXY(1,23); ClrEol; WriteLn('--> Se están localizando los ',JusEnte(num_ped,5),' pedidos.');
    IF  num_ped > 0 THEN                        { Si se tienen registrados pedidos: }
      BEGIN
        LeerEnte(1,1,num_ped_i);                { Almacena en 'num_ped_i' el folio del primer pedido. }
        LeerEnte(num_ped,1,num_ped_f);          { Almacena en 'num_ped_f' el folio del último pedido. }
        ped_part[1] := 1;                       { Especifica que la primera partida del primer pedido es el reg. #1. }
        IF num_ped > 1 THEN                     { Si hay más de un pedido registrado: }
          FOR loc := 1 TO num_ped - 1 DO        { Inicia un contador desde uno hasta el penúltimo pedido registrado. }
            BEGIN
              LeerEnte(loc,28,n_p);                         { Lee el número de partidas que utilizó el pedido. }
              ped_part[loc + 1] := ped_part[loc] + n_p;     { Localiza la 1ª partida del pedido siguiente. }
            END;
        IF NOT OkFolio(num_ped,num_ped_i,num_ped_f) THEN
          MensajeErr('Error en la numeración de pedidos. ** NO CONTINUAR **');
      END;
    CerrarArchivo;                              { Cierra el archivo #4. }
    bas[na].nom_arch := nom_arch_ent;           { Indica que se utilizará el archivo de registro de entradas. }
    AbrirArchivo;                               { Abre el archivo #4 (registro de entradas). }
    num_ent_i := 0;
    num_ent_f := 0;
    num_ent := bas[na].num_regs;                { Almacena en 'num_ent' el número de registros de entradas. }
    IF num_ent > MAX_ENT THEN                   { Si se tienen más entradas registradas que 'MAX_ENT' }
      BEGIN                                     { se trabajará sólo con los primeros 'MAX_ENT' registros de entradas. }
        num_ent := MAX_ENT;
        MensajeErr('Solamente se podrá operar con las primeras ' + JusEnte(num_ent,6) + ' entradas.');
      END;
    GotoXY(1,23); ClrEol; WriteLn('--> Se están localizando las ',JusEnte(num_ent,5),' entradas.');
```

```
  IF  num_ent > 0 THEN                    { Si se tienen registradas entradas: }
    BEGIN
      LeerEnte(1,1,nus_ent_i);            { Almacena en 'num_ent_i' el folio de la primera entrada. }
      LeerEnte(num_ent,1,num_ent_f);      { Almacena en 'num_ent_f' el folio de la última entrada. }
      ent_part[1] := 1;                   { Especifica que la primera partida de la primera entrada es el reg. #1. }
      IF num_ent > 1 THEN                 { Si hay más de una entrada registrada: }
        FOR loc := 1 TO num_ent - 1 DO    { Inicia un contador desde uno hasta la penúltima entrada registrada. }
          BEGIN
            LeerEnte(loc,10,n_p);                          { Lee el número de partidas que utilizó la entrada. }
            ent_part[loc + 1] := ent_part[loc] + n_p;      { Localiza la 1ª partida de la entrada siguiente. }
          END;
      IF NOT OkFolio(num_ent,num_ent_i,num_ent_f) THEN
        MensajeErr('Error en la numeración de entradas. ** NO CONTINUAR **');
    END;
  CerrarArchivo;                          { Cierra el archivo #4. }
  bas[na].nom_arch := nom_arch_sal;       { Indica que se utilizará el archivo de registro de salidas. }
  AbrirArchivo;                           { Abre el archivo #4 (registro de salidas). }
  num_sal_i := 0;
  num_sal_f := 0;
  num_sal := bas[na].num_regs;            { Almacena en 'num_sal' el número de registros de salidas. }
  IF num_sal > MAX_SAL THEN               { Si se tienen más salidas registradas que 'MAX_SAL' }
    BEGIN                                 { se trabajará sólo con los primeros 'MAX_SAL' registros de salidas. }
      num_sal := MAX_SAL;
      MensajeErr('Solamente se podrá operar con las primeras ' + JusEnte(num_sal,6) + ' salidas.');
    END;
  GotoXY(1,23); ClrEol; WriteLn('--> Se están localizando las ',JusEnte(num_sal,5),' salidas.');
  IF  num_sal > 0 THEN                    { Si se tienen registradas salidas: }
    BEGIN
      LeerEnte(1,1,num_sal_i);            { Almacena en 'num_sal_i' el folio de la primera salida. }
      LeerEnte(num_sal,1,num_sal_f);      { Almacena en 'num_sal_f' el folio de la última salida. }
      sal_part[1] := 1;                   { Especifica que la primera partida de la primera salida es el reg. #1. }
      IF num_sal > 1 THEN                 { Si hay más de una salida registrada: }
        FOR loc := 1 TO num_sal - 1 DO    { Inicia un contador desde uno hasta la penúltima salida registrada. }
          BEGIN
            LeerEnte(loc,8,n_p);                           { Lee el número de partidas que utilizó la salida. }
            sal_part[loc + 1] := sal_part[loc] + n_p;      { Localiza la 1ª partida de la salida siguiente. }
          END;
      IF NOT OkFolio(num_sal,num_sal_i,num_sal_f) THEN
        MensajeErr('Error en la numeración de salidas. ** NO CONTINUAR **');
    END;
  CerrarArchivo;                          { Cierra el archivo #4. }
  bas[na].nom_arch := nom_arch_aju;       { Indica que se utilizará el archivo de registro de ajustes. }
  AbrirArchivo;                           { Abre el archivo #4 (registro de ajustes). }
  num_aju := bas[na].num_regs;            { Almacena en 'num_aju' el número de registros de ajustes. }
  GotoXY(1,23); ClrEol; WriteLn('--> Se están localizando las ',JusEnte(num_aju,5),' ajustes.');
  IF  num_aju > 0 THEN                    { Si se tienen registrados ajustes: }
    BEGIN
      LeerEnte(1,1,num_aju_i);            { Almacena en 'num_aju_i' el folio del primer ajuste. }
```

```
        LeerEnte(num_aju,1,num_aju_f);        { Almacena en 'num_aju_f' el folio del último ajuste. }
      END;
    CerrarArchivo;                            { Cierra el archivo #4. }
  END;
```

Este procedimiento hace posible que se pueda conocer la localización, en el disco, de cualquier dato buscado sin tener que recurrir al él; de esta manera, se evitan totalmente las "lentas" búsquedas en los registros de las bases de datos del disco. También se utiliza para mantener en memoria la información de las bases de datos de inventarios (materiales y productos terminados) y las descripciones para producción y maquila.

Las bases de datos INV_MP, INV_PT, PROD y MAQ, se almacenarán totalmente en la memoria dinámica de la computadora. Para hacerlo se hace uso de variables puntero, para lo cual se cuenta con un sistema de 640 *KBytes*, con una capacidad de almacenamiento aproximadamente de 500 *KBytes*. Dicha cantidad de memoria resulta más que suficiente para poder operar en forma muy eficiente con grandes bases de datos. En la operación normal del programa, sólo será necesario recurrir a estas bases de datos para su actualización.

Para una localización rápida de la descripción de proveedores y de distribuidores, se almacenarán en la memoria estructurada de la computadora las claves que identifican a cada uno de ellos. Como estas claves estarán en orden alfabético ascendente, la búsqueda en memoria será muy eficiente. Una vez que se localiza una clave y su número de registro en memoria, éste será el mismo que el número de registro en disco, de tal manera que el acceso al dato deseado se puede hacer en forma directa.

Para la localización de datos de los pedidos, entradas, salidas, y ajustes, lo que se hace es leer del disco el primer y último folio registrado para cada uno de estos conceptos. Al estar registrados en cada base de datos todos los folios consecutivos desde el primero hasta el último, la localización del número de registro en disco, para algún folio intermedio, se hace mediante una sencilla operación.

Para la localización de las partidas de los pedidos, entradas, y salidas, se almacena en una variable tipo registro (para cada uno de estos conceptos) el número de registro en disco correspondiente a la primera partida de cada pedido, entrada o salida.

De esta forma se logra superar eficazmente uno de los inconvenientes que presentaba el utilizar el lenguaje Pascal para leer archivos del dBase III.

4.3.6 SUBPROGRAMA DE CONSULTAS.

Es un procedimiento de solapamiento que permite las consultas que se indican en los diagramas de flujo (sección 4.2). El código fuente del procedimiento es tan extenso que el editor de textos del Turbo Pascal no permite su manejo en un solo fichero, por lo que se localiza en varios ficheros incluidos. Este caso se presenta también en los procedimentos de solapamiento de pedidos, entradas y salidas.

En seguida se muestra el listado de cada uno de los ficheros incluidos que contienen dicho procedimiento en su totalidad.

CO.OVL (Fichero que contiene el código fuente que se muestra.)

```
Overlay PROCEDURE Consultas;                    ( Procedimiento de solapamiento de primer nivel )
                                                ( utilizado para hacer consultas diversas. )
  VAR
    opcion: TTecla;                             ( Opción seleccionada. )

  PROCEDURE CargaProveedor                      ( Insertar dicho procedimiento. )

  PROCEDURE CargaDistribuidor                   ( Insertar dicho procedimiento. )
```

CI.INC (Fichero que contiene el código fuente que se muestra.)

```
  PROCEDURE Inventarios;                ( Procedimiento para consulta a inventarios. )

    VAR
      opcion: TTecla;                   ( Opción seleccionada. )

    PROCEDURE Materiales;               ( Procedimiento para consulta a materiales. )

      CONST
        REN_VIS = 18;                   ( Número de renglones visibles en pantalla. )

      VAR
        terminar: Boolean;              ( Para terminar con la sección de elección. )
        ren: Byte;                      ( Renglón en curso. )
        cont,                           ( Contador de registros. )
        primero,                        ( Primer registro a presentar. )
        ultimo: Integer;                ( Ultimo registro a presentar. )
        opcion: TTecla;                 ( Opción seleccionada. )
```

```
    crit: TArtic;                       { Criterio de selección utilizado. }

  BEGIN
    ClrScr;                             { Borra la pantalla y presenta la cabecera. }
    GotoXY(9,1); Write('CONSULTAS/INVENTARIOS-> M A T E R I A L E S');
    GotoXY(70,1); WriteLn(Fecha);
    IF num_ap < 1 THEN                  { Si no hay materiales registrados se sale del procedimiento. }
      BEGIN
        MensajeErr('No hay materias primas registradas.');
        Exit;
      END;
    Write('Criterio de selección: ');   { Solicita un criterio de selección. }
    crit := ''; PregCadMay(crit,TAM_ARTIC); WriteLn;
    WriteLn('┌────────┬──────────────────────┬──────┬──────────────┬─────┬─────┬──────┐');
    WriteLn('│ ARTIC  │ DESCRIPCION          │ EXIST│ PRECIO VENTA │ MIN │ MAX │ CA PED│');
    WriteLn('├────────┼──────────────────────┼──────┼──────────────┼─────┼─────┼──────┤');
    cont := 1;
    primero := 1;
    ultimo := num_ap;
    IF crit <> '' THEN                  { En caso de que se pida un criterio de selección: }
      BEGIN
        WHILE (cont <= num_ap) AND      { Avanza el contador hasta que la clave cumpla con el criterio. }
              (Pos(crit,pnt_ap[cont]^.artic) <> 1) DO
          cont := Succ(cont);
        IF cont > num_ap THEN           { Si el contador se encuentra más allá del último material }
          BEGIN                         { se sale del procedimiento. }
            MensajeErr('Ningún material cuaple con el criterio de selección.');
            Exit;
          END;
        primero := cont;                { Se ha encontrado el primer material a mostrar. }
        IF primero = num_ap
          THEN ultimo := primero        { Si el primero está en el límite, éste será también el último. }
          ELSE                          { Si hay más materiales después del primero, localizará }
            BEGIN                       { el último que cumpla con el criterio de selección. }
              WHILE (cont <= num_ap) AND
                    (Pos(crit,pnt_ap[cont]^.artic) = 1) DO
                cont := Succ(cont);
              ultimo := cont - 1;
            END;
        cont := primero;
      END;
    REPEAT
      GotoXY(1,6);
      ren := 1;
      REPEAT                            { Presenta en pantalla la cantidad de 'REN_VIS' materiales iniciando en }
        ClrEol;                         { el registro 'cont' y terminando en 'terminar' de ser posible. }
        IF cont <= ultimo THEN
          WITH pnt_ap[cont]^ DO
            BEGIN
              Write('│',JusCad(artic,TAM_ARTIC),
                    '│',JusCad(desc,TAM_DESC),
                    '│',JusEnte(exist,7),
```

```
                '|',JusReal(precio_v,15,2),
                '|',JusEnte(min,6),
                '|',JusEnte(max,6),
                '|',JusEnte(ca_ped,6),'|'     );
            IF cont = ultimo THEN
              BEGIN
                WriteLn;
                Write('└──────┴──────────────────────────┴──────┴──────┴──────┴──────┴──────┘');
              END;
            cont := Succ(cont);
          END;
        ren := Succ(ren);
        WriteLn;
      UNTIL ren > REN_VIS;              { Se repite la operación hasta que se sobrepasan los renglones visibles. }
      REPEAT                            { Esta sección permite avanzar o retroceder las pantallas de consulta }
        terminar := TRUE;               { o terminar la consulta. }
        GotoXY(30,24);
        Write('(PgUp) (PgDn) (Esc) ');
        PregOpcion(opcion,ESC,#73#81);                  { Pide la elección de una de las opciones. }
        CASE opcion[2] OF
          #73 : IF cont > primero + REN_VIS             { PgUp; Pasa a la pantalla anterior, de ser posible. }
                THEN cont := cont - REN_VIS -
                             (cont - 1) Mod REN_VIS - REN_VIS + Ord(cont - 1) Mod REN_VIS = 0)
                ELSE terminar := FALSE;
          #81 : IF cont > ultimo THEN terminar := FALSE;    { PgDn; Pasa a la pantalla siguiente, de ser posible. }
        END;
      UNTIL terminar;                   { Se repite hasta que se acepte la opción tomada. }
    UNTIL opcion[1] = ESC;              { Se repite hasta que se apriete la tecla de escape. }
  END;

{-------------------------------------------------------------------------------------------------}
PROCEDURE Productos;                    { Procedimiento de consulta a productos terminados. }

  CONST
    REN_VIS = 18;                       { Opción seleccionada. }

  VAR
    terminar: Boolean;                  { Para terminar con la sección de elección. }
    ren: Byte;                          { Renglón en curso. }
    cont,                               { Contador de registros. }
    primero,                            { Primer registro a presentar. }
    ultimo: Integer;                    { Ultimo registro a presentar. }
    opcion: TTecla;                     { Opción seleccionada. }
    crit: TArtic;                       { Criterio de selección utilizado. }

  BEGIN
    ClrScr;                             { Borra la pantalla y presenta la cabecera. }
    GotoXY(1,1); Write('CONSULTAS/INVENTARIOS-> P R O D U C T O S   T E R M I N A D O S');
    GotoXY(70,1); WriteLn(fecha);
    IF num_pt < 1 THEN
      BEGIN
        MensajeErr('No se tienen registrados productos terminados.');
```

```
      Exit;
    END;
  Write('Criterio de selección: ');
  crit := ''; PregCadMay(crit,TAM_ARTIC); WriteLn;
  WriteLn('┌──────┬──────────────────────┬──────┬──────┬──────┬──────────────┐');
  WriteLn('│ ARTIC │      DESCRIPCION      │CA DIS│ EXIST │CA PED│ PRECIO VENTA │');
  WriteLn('├──────┼──────────────────────┼──────┼──────┼──────┼──────────────┤');
  cont := 1;
  primero := 1;
  ultimo := num_pt;
  IF crit <> '' THEN                          { En caso de que se pida un criterio de selección: }
    BEGIN
      WHILE (cont <= num_pt) AND              { Avanza el contador hasta que la clave cumpla con el criterio. }
            (Pos(crit,pnt_pt[cont]^.artic) <> 1) DO
        cont := Succ(cont);
      IF cont > num_pt THEN                   { Si el contador se encuentra más allá del último producto }
        BEGIN                                 { terminado se sale del procedimiento. }
          MensajeErr('Ningún producto terminado cumple con el criterio de selección.');
          Exit;
        END;
      primero := cont;                        { Se ha encontrado el primer producto terminado a mostrar. }
      IF primero = num_pt
        THEN ultimo := primero                { Si el primero está en el límite, éste será también el último. }
        ELSE                                  { Si hay más materiales después del primero, localizará }
          BEGIN                               { el último que cumpla con el criterio de selección. }
            WHILE (cont <= num_pt) AND
                  (Pos(crit,pnt_pt[cont]^.artic) = 1) DO
              cont := Succ(cont);
            ultimo := cont - 1;
          END;
      cont := primero;
    END;
  REPEAT
    GotoXY(1,6);
    ren := 1;
    REPEAT                                { Presenta en pantalla la cantidad de 'REN_VIS' materiales iniciando en }
      ClrEol;                             { el registro 'cont' y terminando en terminar de ser posible. }
      IF cont <= ultimo THEN
        WITH pnt_pt[cont]^ DO
          BEGIN
            Write('│',JusCad(artic,TAM_ARTIC),
                  '│',JusCad(desc,TAM_DESC),
                  '│',JusEnte(ca_dis,6),
                  '│',JusEnte(exist,7),
                  '│',JusEnte(ca_ped,6),
                  '│',JusReal(precio_v,15,2),'│' );
            IF cont = ultimo THEN
              BEGIN
                WriteLn;
                Write('└──────┴──────────────────────┴──────┴──────┴──────┴──────────────┘');
              END;
            cont := Succ(cont);
```

```
        END;
      ren := Succ(ren);
      WriteLn;
    UNTIL ren > REN_VIS;              { Se repite la operación hasta que se sobrepasan los renglones visibles. }
    REPEAT                            { Esta sección permite avanzar o retroceder las pantallas de consulta }
      terminar := TRUE;               { o terminar la consulta. }
      GotoXY(30,24); Write('<PgUp> <PgDn> <Esc> ');
      PregOpcion(opcion,ESC,#73#81);                  { Pide la elección de una de las opciones. }
      CASE opcion[2] OF
        #73 : IF cont > primero + REN_VIS             { PgUp; Pasa a la pantalla anterior, de ser posible. }
              THEN cont := cont - REN_VIS -
                           (cont - 1) Mod REN_VIS - REN_VIS + Ord((cont - 1) Mod REN_VIS = 0)
              ELSE terminar := FALSE;
        #81 : IF cont > ultimo THEN terminar := FALSE;  { PgDn; Pasa a la pantalla siguiente, de ser posible. }
      END;
    UNTIL terminar;                   { Se repite hasta que se acepte la opción tomada. }
  UNTIL opcion[1] = ESC;              { Se repite hasta que se apriete la tecla de escape. }
END;
```

```
BEGIN
  REPEAT
    ClrScr;                           { Presenta la cabecera del procedimiento de solapamiento. }
    GotoXY(17,1); Write('CONSULTAS-> I N V E N T A R I O S');
    GotoXY(70,1); Write(Fecha);
    Posibilidad(20, 5, 'P R O D U C T O S   T E R M I N A D O S');     { Presenta las posibilidades de elección. }
    Posibilidad(20, 7, 'M A T E R I A L E S');
    GotoXY(55,23); Write('¿Qué elección? <Esc> ');
    PregOpcion(opcion,'PM' + ESC,'');                 { Pide la elección de una de las opciones. }
    CASE opcion[1] OF                                 { Ejecuta el procedimiento seleccionado. }
      'P' : Productos;
      'M' : Materiales;
    END;
  UNTIL opcion[1] = ESC;              { Se repite hasta que se presione la tecla de escape. }
END;
```

CD.INC (Fichero que contiene el código fuente que se muestra.)

```
PROCEDURE Descripciones;                    ( Procedimiento para la consulta de descripciones de proveedores y )
                                            ( distribuidores. )
  VAR
    opcion: TTecla;                         ( Opción seleccionada. )

  PROCEDURE Proveedores;                    ( Procedimiento para la consulta de descripción de proveedores. )

    VAR
      cont,                                 ( Contador de registro. )
      primero,                              ( Primer registro a analizar. )
      ultimo: Integer;                      ( Ultimo registro a analizar. )
      terminar: Boolean;                    ( Permite terminar con la sección de opciones. )
      opcion: TTecla;                       ( Opción seleccionada. )
      crit: Cad78;                          ( Criterio de selección. )
      des_pro: TDesPro;                     ( Descripción del proveedor. )

    BEGIN
      ClrScr;                               ( Presenta el encabezado de la sección. )
      GotoXY(8,1); Write('CONSULTAS/DESCRIPCIONES-> P R O V E E D O R E S');
      GotoXY(70,1); WriteLn(Fecha);
      Write('Criterio de selección: ');     ( Pide un criterio de selección. )
      crit := ''; PregCadMay(crit,TAM_CLAVE); WriteLn;
      GotoXY(1,5);                          ( Muestra la plantilla de presentación. )
      WriteLn('
      WriteLn('  Clave:                    Nombre:                                          ');
      WriteLn('
      WriteLn('     Dirección:                                              Tel:          ');
      WriteLn('        Ciudad:                                                            ');
      WriteLn('        Estado:                                                            ');
      WriteLn('  Código postal:                                                           ');
      WriteLn('
      WriteLn('
      WriteLn('  Condiciones de pago:     Dias          Saldo (deudor): $            MN  ');
      WriteLn('
      IF num_pro > 0 THEN                   ( Si hay proveedores registrados: )
        BEGIN
          cont := 1;                        ( Localiza la primera clave que cumpla con el criterio de selección. )
          WHILE (cont <= num_pro) AND (Pos(crit,profcont]) <> 1) DO
            cont := Succ(cont);
          primero := cont;
          IF primero >= num_pro
            THEN ultimo := primero
            ELSE                            ( Si el primero está antes que el último, se hace la búsqueda del último. )
              BEGIN
                WHILE (cont <= num_pro) AND (Pos(crit,profcont]) = 1) DO
                  cont := Succ(cont);
                ultimo := cont - 1;
```

```
        END;
      cont := primero;                ( Se mostrarán a partir del primero. )
      IF primero <= num_pro
       THEN                           ( Si se localizó la clave: )
        BEGIN
          na := 5;                    ( Abre la base de datos de descripciones de proveedores. )
          bas[na].nom_arch := nom_arch_des_pro;
          AbrirArchivo;
          REPEAT
            CargaProveedor(cont,des_pro);       ( Lee de la base de datos la descripción en curso. )
            WITH des_pro DO
              BEGIN                           ( Despliega la descripción del proveedor. )
                GotoXY(10,6); Write(JusCad(clave,TAM_CLAVE));
                GotoXY(38,6); Write(JusCad(nombre,TAM_NOMBRE));
                GotoXY(18,8); Write(JusCad(direc,TAM_DIRECC));
                GotoXY(68,8); Write(JusCad(telef,TAM_TELEF));
                GotoXY(18,9); Write(JusCad(ciudad,TAM_CIUDAD));
                GotoXY(18,10); Write(JusCad(estado,TAM_ESTADO));
                GotoXY(18,11); Write(JusCad(c_p,TAM_C_P));
                GotoXY(24,14); Write(cdp:3);
                GotoXY(58,14); Write(JusReal(saldo,15,2));
              END;
            REPEAT
              terminar := TRUE;
              GotoXY(30,24); Write('<PgUp> <PgDn> <Esc> ');
              PregOpcion(opcion,ESC,#73#81);          ( Pide que se toque una de las opciones. )
              CASE opcion[2] OF
                #73: IF (cont > primero)              ( PgUp; pasa a la descripción anterior, si es posible. )
                       THEN cont := cont - 1
                       ELSE terminar := FALSE;
                #81: IF cont < ultimo                 ( PgDn; pasa a la descripción siguiente, si es )
                       THEN cont := cont + 1          ( posible. )
                       ELSE terminar := FALSE;
              END;
            UNTIL terminar;                           ( Pregunta hasta que la opción tomada sea la correcta. )
          UNTIL opcion[1] = ESC;                      ( Termina el procedimiento cuando se aprieta escape. )
          CerrarArchivo;                              ( Cierra el archivo de descripciones de proveedores. )
        END
       ELSE MensajeErr('Ningún proveedor cuaple con el criterio de selección.');
    END;
  END;
```

```
PROCEDURE Distribuidores;                 ( Procedimiento para la consulta de la descripción de dist. )

  VAR
    cont,                           ( Contador de registro. )
    primero,                        ( Primer registro a analizar. )
    ultimo: Integer;                ( Ultimo registro a analizar. )
    terminar: Boolean;              ( Permite terminar con la sección de opciones. )
    opcion: TTecla;                 ( Opción seleccionada. )
    crit: Cad78;                    ( Criterio de selección. )
```

```
  des_dis: TDesDis;                    { Descripción del distribuidor. }

BEGIN
  ClrScr;                              { Presenta el encabezado de la sección. }
  GotoXY(1,1); Write('CONSULTAS/DESCRIPCIONES-> D I S T R I B U I D O R E S');
  GotoXY(70,1); WriteLn(Fecha);
  Write('Criterio de selección: ');    { Pide un criterio de selección. }
  crit := ''; PregCadMay(crit,TAM_CLAVE); WriteLn;
  GotoXY(1,5);                         { Muestra la plantilla de presentación. }
  WriteLn('
  WriteLn('  Clave:                    Nombre:                                    ');
  WriteLn('
  WriteLn('      Dirección:                                        Tel:           ');
  WriteLn('         Ciudad:                                                       ');
  WriteLn('         Estado:                                                       ');
  WriteLn('  Código postal:                                                       ');
  WriteLn('   Esbárquese a:                                                       ');
  WriteLn('
  WriteLn('  Descuento:   %                          Descuento por pronto pago:   %');
  WriteLn('
  WriteLn('  Condiciones de pago:     Días          Saldo (acree.): $           MN');
  WriteLn('
  WriteLn('  Crédito actual: $                MN    Crédito límite: $           MN');
  WriteLn('
  IF num_dis > 0 THEN                  { Si hay distribuidores registrados: }
    BEGIN
      cont := 1;                       { Localiza la primera clave que cumpla con el criterio de selección. }
      WHILE (cont <= num_dis) AND (Pos(crit,dis[cont]) <> 1) DO
        cont := Succ(cont);
      primero := cont;
      IF primero >= num_dis
        THEN ultimo := primero
        ELSE                           { Si el primero está antes que el último, se hace la búsqueda del último. }
          BEGIN
            WHILE (cont <= num_dis) AND (Pos(crit,dis[cont]) = 1) DO
              cont := Succ(cont);
            ultimo := cont - 1;
          END;
      cont := primero;                 { Se mostrarán a partir del primero. }
      IF primero <= num_dis
        THEN                           { Si se localizó la clave: }
          BEGIN
            na := 5;                   { Abre la base de datos de descripciones de distribuidores. }
            bas[na].nom_arch := nom_arch_des_dis;
            AbrirArchivo;
            REPEAT
              CargaDistribuidor(cont,des_dis);    { Lee de la base de datos la descripción en curso. }
              WITH des_dis DO
                BEGIN                  { Despliega la descripción del distribuidor. }
                  GotoXY(10,6); Write(JusCad(clave,TAM_CLAVE));
                  GotoXY(38,6); Write(JusCad(nombre,TAM_NOMBRE));
                  GotoXY(18,8); Write(JusCad(direc,TAM_DIREC));
```

```
                GotoXY(68,8); Write(JusCad(telef,TAM_TELEF));
                GotoXY(18,9); Write(JusCad(ciudad,TAM_CIUDAD));
                GotoXY(18,10); Write(JusCad(estado,TAM_ESTADO));
                GotoXY(18,11); Write(JusCad(c_p,TAM_C_P));
                GotoXY(18,12); Write(JusCad(emb_a,TAM_EMB_A));
                GotoXY(14,14); Write(d:2);
                GotoXY(75,14); Write(dp:2);
                GotoXY(24,16); Write(cdp:3);
                GotoXY(58,16); Write(JusReal(saldo,14,2));
                GotoXY(21,18); Write(JusReal(cred,14,2));
                GotoXY(59,18); Write(JusReal(l_cred,14,2));
              END;
            REPEAT
              terminar := TRUE;
              GotoXY(30,24); Write('<PgUp> <PgDn> <Esc> ');
              PregOpcion(opcion,ESC,#73#81);          { Pide que se tome una de las opciones. }
              CASE opcion[2] OF
                #73: IF (cont > primero)              { PgUp; pasa a la descripción anterior, si es posible. }
                       THEN cont := cont - 1
                       ELSE terminar := FALSE;
                #81: IF cont < ultimo                 { PgDn; pasa a la descripción siguiente, si es }
                       THEN cont := cont + 1          { posible. }
                       ELSE terminar := FALSE;
              END;
            UNTIL terminar;                           { Pregunta hasta que la opción tomada sea la correcta. }
          UNTIL opcion[1] = ESC;                      { Termina el procedimiento cuando se aprieta escape. }
          CerrarArchivo;                              { Cierra el archivo de descripciones de dist. }
        END
      ELSE MensajeErr('Ningún distribuidor cumple con el criterio de selección.');
    END;
  END;
```

```
BEGIN
  REPEAT
    ClrScr;                                           { Presenta el encabezado de la sección. }
    GotoXY(16,1); Write('CONSULTAS-> D E S C R I P C I O N E S');
    GotoXY(70,1); Write(Fecha);
    Posibilidad(25, 5, 'P R O V E E D O R E S');      { Presenta las posibilidades de elección. }
    Posibilidad(25, 7, 'D I S T R I B U I D O R E S');
    GotoXY(55,23); Write('¿Qué elección? <Esc> ');
    PregOpcion(opcion,'PD' + ESC,'');                 { Pide que se tome una de las opciones. }
    CASE opcion[1] OF                                 { Ejecuta el procedimiento seleccionado. }
      'P' : Proveedores;
      'D' : Distribuidores;
    END;
  UNTIL opcion[1] = ESC;                              { Termina el procedimiento cuando se aprieta escape. }
END;
```

CR.INC { Fichero que contiene el código fuente que se muestra. }

```
PROCEDURE Registros;                    { Procedimiento para la consulta a los registros de bases de datos. }
|
|  VAR
|    opcion: TTecla;                    { Opción seleccionada. }
|
|-------------------------------------------------------------------------------
|  PROCEDURE Pedidos;                   { Procedimiento de consulta al registro de pedidos. }
|  |
|  |  VAR
|  |    salir: Boolean;                 { Permite salir de una sección. }
|  |    loc_ped: Integer;               { Número de registro del pedido en consulta. }
|  |    opcion: TTecla;                 { Opción seleccionada. }
|  |    pedido: TPed;                   { Variable que almacena los datos del pedido en consulta. }
|  |
|  |-----------------------------------------------------------------------------
|  |  PROCEDURE CargaPedido             { Insertar dicho procedimiento. }
|  |  |
|  |  |---------------------------------------------------------------------------
|  |  PROCEDURE MostrarCabezaPedido     { Insertar dicho procedimiento. }
|  |  |
|  |  |---------------------------------------------------------------------------
|  |  PROCEDURE MostrarPartidas         { Insertar dicho procedimiento. }
|  |  |
|  |  |---------------------------------------------------------------------------
|  |  PROCEDURE MostrarOtros            { Insertar dicho procedimiento. }
|  |  |
|  |-----------------------------------------------------------------------------
|  BEGIN
|    ClrScr;                                    { Presenta la cabecera de consulta al registro de pedidos. }
|    GotoXY(22,1); Write('CONSULTAS/REGISTROS-> P E D I D O S');
|    GotoXY(70,1); Write(Fecha);
|    IF num_ped = 0 THEN                        { Si no hay pedidos registrados, se sale del procedimiento. }
|      BEGIN
|        MensajeErr('No hay pedidos registrados.');
|        Exit;
|      END;
|    na := 4;                                   { Abre los archivos de base de datos de pedidos. }
|    bas[na].nom_arch := nom_arch_ped;
|    AbrirArchivo;
|    na := 5;
|    bas[na].nom_arch := nom_arch_ped_p;
|    AbrirArchivo;
|    REPEAT
|      GotoXY(1,2); Write('Pedido: ( ',JusFolio(num_ped_i),'..',JusFolio(num_ped_f),') ');
|      pedido.num := num_ped_i;
|      REPEAT                                           { Pide el folio del pedido a consultar. }
|        GotoXY(22,2); PregEnte(pedido.num,5,FALSE);
|        GotoXY(22,2); Write(JusFolio(pedido.num),' ');
```

```
    UNTIL OkFolio(pedido.num,num_ped_i,num_ped_f);          { Se repite hasta que el folio esté correcto. }
    REPEAT
      CargaPedido(pedido.num,loc_ped,pedido);               { Lee del disco el pedido requerido. }
      MostrarOtros;                                         { Muestra los datos del estado del pedido. }
      Salir := FALSE;
      REPEAT
        GotoXY(4,23); ClrEol; Write('<F1 Cabecera> <F2 Partidas> <F3 Otros> <PgUp> <PgDn> <Home Pedido> <Esc> ');
        PregOpcion(opcion,ESC,#59#60#61#71#73#81);          { Pide la elección de una de las opciones. }
        CASE opcion[2] OF
          #59 : MostrarCabezaPedido;                        { F1; Muestra la cabecera del pedido. }
          #60 : MostrarPartidas;                            { F2; Muestra las partidas del pedido. }
          #61 : MostrarOtros;                               { F3; Muestra los otros datos del pedido. }
          #73 : IF pedido.num <> num_ped_i THEN             { PgUp; Pasa al pedido anterior de ser posible. }
                  BEGIN
                    salir := TRUE;
                    pedido.num := AntFolio(pedido.num);
                    GotoXY(22,2); Write(JusFolio(pedido.num));
                  END;
          #81 : IF pedido.num <> num_ped_f THEN             { PgDn; Pasa al pedido siguiente de ser posible. }
                  BEGIN
                    salir := TRUE;
                    pedido.num := SigFolio(pedido.num);
                    GotoXY(22,2); Write(JusFolio(pedido.num));
                  END;
          ELSE salir := TRUE;                               { Esc o Home; Permite la salida. }
        END;
      UNTIL salir;                                          { Se repite si la opción seleccionada no es válida. }
    UNTIL (opcion[2] = #71) OR (opcion[1] = ESC);           { Se repite hasta que se apriete 'Esc' o 'Home' (para }
                                                            { seleccionar manualmente otro pedido). }
  UNTIL opcion[1] = ESC;                                    { Se repite hasta que se apriete la tecla de escape. }
  na := 4;                                                  { Se cierran los archivos utilizados. }
  CerrarArchivo;
  na := 5;
  CerrarArchivo;
END;
```

```
PROCEDURE Entradas;                     { Procedimiento de consulta al registro de entradas. }

  VAR
    salir: Boolean;                     { Permite salir de una sección. }
    loc_ent: Integer;                   { Número de registro de la entrada en consulta. }
    opcion: TTecla;                     { Opción seleccionada. }
    entrada: TEnt;                      { Variable que almacena los datos de la entrada en consulta. }

  PROCEDURE CargaEntrada                { Insertar dicho procedimiento. }

  PROCEDURE MostrarEntrada              { Insertar dicho procedimiento. }
```

```
BEGIN
  ClrScr;                                    { Presenta la cabecera de consulta al registro de entradas. }
  GotoXY(23,1); Write('CONSULTAS/REGISTROS-> E N T R A D A S');
  GotoXY(70,1); Write(Fecha);
  IF num_ent = 0 THEN                        { Si no hay entradas registradas, se sale del procedimiento. }
    BEGIN
      MensajeErr('No se tienen entradas registradas.');
      Exit;
    END;
  na := 4;                                   { Abre los archivos de base de datos de entradas. }
  bas[na].nom_arch := nom_arch_ent;
  AbrirArchivo;
  na := 5;
  bas[na].nom_arch := nom_arch_ent_p;
  AbrirArchivo;
  WITH entrada DO
    REPEAT
      GotoXY(1,2); Write('Entrada: (',JusFolio(num_ent_i),'..',JusFolio(num_ent_f),') ');
      num := num_ent_i;
      REPEAT                                 { Pide el folio de la entrada a consultar. }
        GotoXY(23,2); PregEnte(num,5,FALSE);
        GotoXY(23,2); Write(JusFolio(num),' ');
      UNTIL OkFolio(num,num_ent_i,num_ent_f); { Se repite hasta que el folio esté correcto. }
      REPEAT
        CargaEntrada(num,loc_ent,entrada);   { Lee del disco la entrada requerida. }
        MostrarEntrada;                      { Muestra los datos de la entrada en consulta. }
        Salir := FALSE;
        REPEAT
          GotoXY(23,23); ClrEol; Write('<PgUp> <PgDn> <Home Entrada> <Esc> ');
          PregOpcion(opcion,ESC,#71#73#81);  { Pide la elección de una de las opciones. }
          CASE opcion[2] OF
            #73 : IF num <> num_ent_i THEN   { PgUp; Pasa a la entrada anterior, de ser posible. }
                    BEGIN
                      salir := TRUE;
                      num := AntFolio(num);
                      GotoXY(23,2); Write(JusFolio(num));
                    END;
            #81 : IF num <> num_ent_f THEN   { PgDn; Pasa a la entrada siguiente, de ser posible. }
                    BEGIN
                      salir := TRUE;
                      num := SigFolio(num);
                      GotoXY(23,2); Write(JusFolio(num));
                    END;
            ELSE salir := TRUE;              { Esc o Home; Permite la salida. }
          END;
        UNTIL salir;                         { Se repite si la opción seleccionada no es válida. }
      UNTIL (opcion[2] = #71) OR (opcion[1] = ESC); { Se repite hasta que se apriete 'Esc' o 'Home' (para }
                                             { seleccionar manualmente otra entrada). }
    UNTIL opcion[1] = ESC;                   { Se repite hasta que se apriete la tecla de escape. }
  na := 4;                                   { Se cierran los archivos utilizados. }
  CerrarArchivo;
  na := 5;
```

```
    CerrarArchivo;
  END;

PROCEDURE Salidas;                    { Procedimiento de consulta al registro de salidas. }

  VAR
    salir: Boolean;                   { Permite salir de una sección. }
    loc_sal: Integer;                 { Número de registro de la salida en consulta. }
    opcion: TTecla;                   { Opción seleccionada. }
    salida: TSal;                     { Variable que almacena los datos de la salida en consulta. }

  PROCEDURE CargaSalida               { Insertar dicho procedimiento. }

  PROCEDURE MostrarSalida             { Insertar dicho procedimiento. }

  BEGIN                                          { Presenta la cabecera de consulta al registro de salidas. }
    ClrScr;
    GotoXY(22,1); Write('CONSULTAS/REGISTROS-> S A L I D A S');
    GotoXY(70,1); Write(Fecha);
    IF num_sal = 0 THEN                          { Si no hay salidas registradas, se sale del procedimiento. }
      BEGIN
        MensajeErr('No se tienen salidas registradas.');
        Exit;
      END;
    na := 4;                                     { Abre los archivos de base de datos de salidas. }
    bas[na].nom_arch := nom_arch_sal;
    AbrirArchivo;
    na := 5;
    bas[na].nom_arch := nom_arch_sal_p;
    AbrirArchivo;
    WITH salida DO
      REPEAT
        GotoXY(1,2); Write(' Salida: (',JusFolio(num_sal_i),'..',JusFolio(num_sal_f),') ');
        num := num_sal_i;
        REPEAT                                   { Pide el folio de la salida a consultar. }
          GotoXY(23,2); PregEnte(num,5,FALSE);
          GotoXY(23,2); Write(JusFolio(num),' ');
        UNTIL OkFolio(num,num_sal_i,num_sal_f);  { Se repite hasta que el folio esté correcto. }
        REPEAT
          CargaSalida(num,loc_sal,salida);       { Lee del disco la salida requerida. }
          MostrarSalida;                         { Muestra los datos de la salida en consulta. }
          Salir := FALSE;
          REPEAT
            GotoXY(23,23); ClrEol; Write('<PgUp> <PgDn> <Home Salida> <Esc> ');
            PregOpcion(opcion,ESC,#71#73#81);    { Pide la elección de una de las opciones. }
            CASE opcion[2] OF
              #73 : IF num <> num_sal_i THEN     { PgUp; Pasa a la salida anterior, de ser posible. }
                      BEGIN
```

```
                    salir := TRUE;
                    num := AntFolio(num);
                    GotoXY(23,2); Write(JusFolio(num));
                  END;
            #81 : IF num <> num_sal_f THEN          { PgDn; Pasa a la salida siguiente, de ser posible. }
                  BEGIN
                    salir := TRUE;
                    num := SigFolio(num);
                    GotoXY(23,2); Write(JusFolio(num));
                  END;
            ELSE salir := TRUE;                     { Esc o Home; Permite la salida. }
          END;
        UNTIL salir;                                { Se repite si la opción seleccionada no es válida. }
      UNTIL (opcion[2] = #71) OR (opcion[1] = ESC); { Se repite hasta que se apriete 'Esc' o 'Home' (para }
                                                    { seleccionar manualmente otra salida). }
    UNTIL opcion[1] = ESC;                          { Se repite hasta que se apriete la tecla de escape. }
  na := 4;                                          { Se cierran los archivos utilizados. }
  CerrarArchivo;
  na := 5;
  CerrarArchivo;
END;

PROCEDURE Ajustes;                  { Procedimiento de consulta al registro de ajustes. }

  CONST
    REN_VIS = 18;                   { Número de renglones visibles en pantalla. }

  VAR
    terminar: Boolean;              { Para terminar con la sección de elección. }
    ren: Byte;                      { Renglón en curso. }
    cont,                           { Contador de registros. }
    loc_aju: Integer;               { Localización del ajuste. }
    opcion: TTecla;                 { Opción seleccionada. }
    ajuste: TAju;                   { Variable que almacena los datos del ajuste en consulta. }

  PROCEDURE CargaAjuste             { Insertar dicho procedimiento. }

BEGIN
  ClrScr;                                   { Presenta la cabecera de consulta al registro de ajustes. }
  GotoXY(22,1); Write('CONSULTAS/REGISTROS-> A J U S T E S');
  GotoXY(70,1); Write(Fecha);
  IF num_aju = 0 THEN                       { Si no hay ajustes registrados, se sale del procedimiento. }
    BEGIN
      MensajeErr('No se tienen ajustes registrados.');
      Exit;
    END;
  na := 4;                                  { Abre el archivo de base de datos de ajustes. }
  bas[na].nom_arch := nom_arch_aju;
  AbrirArchivo;
```

```
  GotoXY(1,3);
  WriteLn('┌──────┬──────┬──────┬──────┬──────┬──────┬──────┐');
  WriteLn('│ NUMERO │ ORIGEN │ RAZON │ ARTICULO │ CANTIDAD │ PRECIO │ FECHA │');
  WriteLn('└──────┴──────┴──────┴──────┴──────┴──────┴──────┘');
  cont := 1;
  WITH ajuste DO
    REPEAT
      GotoXY(1,6);
      ren := 1;
      REPEAT                          { Presenta en pantalla la cantidad de 'REN_VIS' ajustes iniciando en }
        ClrEol;                       { el registro 'cont' y terminando en el último, de ser posible. }
        IF cont <= num_aju THEN
          BEGIN
            CargaAjuste(cont,ajuste);
            Write('│ ',JusFolio(nua),
                  ' │ ',origen,
                  ' │ ',JusEnte(razon,3),
                  ' │ ',JusCad(artic,9),
                  '│ ',JusEnte(cant,6),
                  ' │ ',JusReal(precio,14,2),
                  ' │ ',JusCad(fech,9),'│' );
            IF cont = num_aju THEN
              BEGIN
                WriteLn;
                Write('└──────┴──────┴──────┴──────┴──────┴──────┴──────┘');
              END;
            cont := Succ(cont);
          END;
        ren := Succ(ren);
        WriteLn;
      UNTIL ren > REN_VIS;            { Se repite la operación hasta que se sobrepasen los renglones visibles. }
      REPEAT                          { Esta sección permite avanzar o retroceder las pantallas de consulta }
        terminar := TRUE;             { o terminar la consulta. }
        GotoXY(30,24); Write('<PgUp> <PgDn> <Esc> ');
        PregOpcion(opcion,ESC,#73#81);                    { Pide la elección de una de las opciones. }
        CASE opcion[2] OF
          #73 : IF cont > 1 + REN_VIS                     { PgUp; Pasa a la pantalla anterior, de ser posible. }
                THEN cont := cont - REN_VIS -
                     (cont - 1) Mod REN_VIS - REN_VIS + Ord(cont - 1) Mod REN_VIS = 0)
                ELSE terminar := FALSE;
          #81 : IF cont > num_aju THEN terminar := FALSE;   { PgDn; Pasa a la pantalla siguiente, de ser posible. }
        END;
      UNTIL terminar;                 { Se repite hasta que se acepte la opción tomada. }
    UNTIL opcion[1] = ESC;            { Se repite hasta que se apriete la tecla de escape. }
  na := 4;                            { Se cierra el archivo utilizado. }
  CerrarArchivo;
END;
```

```
PROCEDURE Dinero;                    { Procedimiento de consulta a los registros de dinero. }

  CONST
    REN_VIS = 18;                    { Número de renglones visibles en pantalla. }

  VAR
    opcion: TTecla;                  { Opción seleccionada. }

  PROCEDURE Cobros;                  { Procedimiento de consulta al registro de cobros. }

    VAR
      terminar: Boolean;             { Para terminar con la sección de elección. }
      ren: Byte;                     { Renglón en curso. }
      cont,                          { Contador de registros. }
      loc_aju: Integer;              { Localización del ajuste. }
      opcion: TTecla;                { Opción seleccionada. }
      cobro: TCob;                   { Variable que almacena los datos del cobro en consulta. }

    PROCEDURE CargaCobro             { Insertar dicho procedimiento. }

    BEGIN
      ClrScr;                                  { Presenta la cabecera de consulta al registro de cobros. }
      GotoXY(20,1); Write('CONSULTAS/REGISTROS/DINERO-> C O B R O S');
      GotoXY(70,1); Write(Fecha);
      na := 4;                                 { Abre el archivo de base de datos de cobros. }
      basfnal.noa_arch := noa_arch_cob;
      AbrirArchivo;
      IF basfnal.nua_regs = 0 THEN             { Si no hay cobros registrados, se sale del procedimiento. }
        BEGIN
          MensajeErr('No se tienen cobros registrados.');
          CerrarArchivo;
          Exit;
        END;
      GotoXY(1,3);
      WriteLn('                                                                    ');
      WriteLn('  CLAVE  |  FECHA  |  FORMA PAGO  |  IMPORTE  ');
      WriteLn('                                                                    ');
      cont := 1;
      WITH cobro DO
        REPEAT
          GotoXY(1,6);
          ren := 1;
          REPEAT                           { Presenta en pantalla la cantidad de 'REN_VIS' cobros iniciando en }
            ClrEol;                        { el registro 'cont' y terminando en el último, de ser posible. }
            IF cont <= basfnal.nua_regs THEN
              BEGIN
                CargaCobro(cont,ccobro);
                Write('                  | ',JusCad(clave,8),
```

```
                '| ',JusCad(fech,10),
                '| ',JusCad(forma_pag,11),
                '| ',JusReal(importe,15,2),'|');
            IF cont = bas[na].num_regs THEN
              BEGIN
                WriteLn;
                Write('                └──────────┴──────────┴──────────┴──────────┴──────────┴──────────┘');
              END;
            cont := Succ(cont);
          END;
        ren := Succ(ren);
        WriteLn;
      UNTIL ren > REN_VIS;          { Se repite la operación hasta que se sobrepasen los renglones visibles. }
      REPEAT                        { Esta sección permite avanzar o retroceder las pantallas de consulta }
        terminar := TRUE;           { o terminar la consulta. }
        GotoXY(30,24); Write('<PgUp> <PgDn> <Esc> ');
        PregOpcion(opcion,ESC,#73#81);              { Pide la elección de una de las opciones. }
        CASE opcion[2] OF
          #73 : IF cont > 1 + REN_VIS               { PgUp; Pasa a la pantalla anterior, de ser posible. }
                THEN cont := cont - REN_VIS -
                    (cont - 1) Mod REN_VIS - REN_VIS + Ord((cont - 1) Mod REN_VIS = 0)
                ELSE terminar := FALSE;
          #81 : IF cont > bas[na].num_regs THEN     { PgDn; Pasa a la pantalla siguiente, de ser posible. }
                 terminar := FALSE;
        END;
      UNTIL terminar;               { Se repite hasta que se acepte la opción tomada. }
    UNTIL opcion[1] = ESC;          { Se repite hasta que se apriete la tecla de escape. }
    na := 4;                        { Se cierra el archivo utilizado. }
    CerrarArchivo;
  END;

{-------------------------------------------------------------------------------}
PROCEDURE Pagos;                    { Procedimiento de consulta al registro de cobros. }

  VAR
    terminar: Boolean;              { Para terminar con la sección de elección. }
    ren: Byte;                      { Renglón en curso. }
    cont,                           { Contador de registros. }
    loc_aju: Integer;               { Localización del ajuste. }
    opcion: TTecla;                 { Opción seleccionada. }
    pago: TPag;                     { Variable que almacena los datos del cobro en consulta. }

{-------------------------------------------------------------------------------}
  PROCEDURE CargaPago               { Insertar dicho procedimiento. }

{-------------------------------------------------------------------------------}
  BEGIN
    ClrScr;                                 { Presenta la cabecera de consulta al registro de pagos. }
    GotoXY(19,1); Write('CONSULTAS/REGISTROS/DINERO-> P A G O S');
    GotoXY(70,1); Write(Fecha);
    na := 4;                                { Abre el archivo de base de datos de pagos. }
    bas[na].nom_arch := nom_arch_pag;
```

```
  AbrirArchivo;
  IF bas[na].num_regs < 1 THEN              { Si no hay pagos registrados, se sale del procedimiento. }
    BEGIN
      MensajeErr('No se tienen pagos registrados.');
      CerrarArchivo;
      Exit;
    END;
  GotoXY(1,3);
  WriteLn('                                                                    ');
  WriteLn('               | CLAVE | FECHA | FORMA PAGO | IMPORTE |            ');
  WriteLn('               |-------|-------|------------|---------|            ');
  cont := 1;
  WITH pago DO
    REPEAT
      GotoXY(1,6);
      ren := 1;
      REPEAT                        { Presenta en pantalla la cantidad de 'REN_VIS' pagos iniciando en }
        ClrEol;                     { el registro 'cont' y terminando en el último, de ser posible. }
        IF cont <= bas[na].num_regs THEN
          BEGIN
            CargaPago(cont,pago);
            Write('               | ',JusCad(clave,8),
                  ' | ',JusCad(fech,10),
                  ' | ',JusCad(forma_pag,11),
                  ' | ',JusReal(importe,15,2),' |');
            IF cont = bas[na].num_regs THEN
              BEGIN
                WriteLn;
                Write('               |_______|_______|____________|_________|');
              END;
            cont := Succ(cont);
          END;
        ren := Succ(ren);
        WriteLn;
      UNTIL ren > REN_VIS;          { Se repite la operación hasta que se sobrepasen los renglones visibles. }
      REPEAT                        { Esta sección permite avanzar o retroceder las pantallas de consulta }
        terminar := TRUE;           { o terminar la consulta. }
        GotoXY(30,24); Write('<PgUp> <PgDn> <Esc> ');
        PregOpcion(opcion,ESC,#73#81);              { Pide la elección de una de las opciones. }
        CASE opcion[2] OF
          #73 : IF cont > 1 + REN_VIS               { PgUp; Pasa a la pantalla anterior, de ser posible. }
                THEN cont := cont - REN_VIS -
                     (cont - 1) Mod REN_VIS - REN_VIS + Ord((cont - 1) Mod REN_VIS = 0)
                ELSE terminar := FALSE;
          #81 : IF cont > bas[na].num_regs THEN     { PgDn; Pasa a la pantalla siguiente, de ser posible. }
                  terminar := FALSE;
        END;
      UNTIL terminar;               { Se repite hasta que se acepte la opción tomada. }
    UNTIL opcion[1] = ESC;          { Se repite hasta que se apriete la tecla de escape. }
  na := 4;                          { Se cierra el archivo utilizado. }
  CerrarArchivo;
END;
```

```
BEGIN
  REPEAT
    ClrScr;                                  { Presenta la cabecera de consulta a los registros de dinero. }
    GotoXY(23,1); Write('CONSULTAS/REGISTROS-> D I N E R O');
    GotoXY(70,1); Write(Fecha);
    Posibilidad(34,5,'C O B R O S');         { Presenta las posibilidades de elección. }
    Posibilidad(34,7,'P A G O S');
    GotoXY(55,23); Write('¿Qué elección? (Esc) ');   { Pide la elección de una de las opciones. }
    PregOpcion(opcion,'CP' + ESC,'');
    CASE opcion[1] OF                        { Ejecuta el procedimiento seleccionado. }
      'C' : Cobros;
      'P' : Pagos;
    END;
  UNTIL opcion[1] = ESC;                     { Se repite hasta que se apriete la tecla de escape. }
END;
```

```
BEGIN
  REPEAT
    ClrScr;                                  { Presenta la cabecera de consulta a los registros. }
    GotoXY(20,1); Write('CONSULTAS-> R E G I S T R O S');
    GotoXY(70,1); Write(Fecha);
    Posibilidad(30, 5, 'P E D I D O S');     { Presenta las posibilidades de elección. }
    Posibilidad(30, 7, 'E N T R A D A S');
    Posibilidad(30, 9, 'S A L I D A S');
    Posibilidad(30,11, 'D I N E R O');
    Posibilidad(30,13, 'A J U S T E S');
    GotoXY(55,23); Write('¿Qué elección? (Esc) ');
    PregOpcion(opcion,'PESDA' + ESC,'');     { Pide la elección de una de las opciones. }
    CASE opcion[1] OF                        { Ejecuta el procedimiento seleccionado. }
      'P' : Pedidos;
      'E' : Entradas;
      'S' : Salidas;
      'D' : Dinero;
      'A' : Ajustes;
    END;
  UNTIL opcion[1] = ESC;                     { Se repite hasta que se apriete la tecla de escape. }
END;
```

CI.CVL (Fichero que contiene el código fuente que se muestra.)

```
BEGIN
  REPEAT
    ClrScr;                                    ( Borra la pantalla y presenta las opciones de elección. )
    GotoXY(31,1); Write('C O N S U L T A S');
    GotoXY(70,1); Write(Fecha);
    Posibilidad(30, 5, 'I N V E N T A R I O S');
    Posibilidad(30, 7, 'D E S C R I P C I O N E S');
    Posibilidad(30, 9, 'R E G I S T R O S');
    GotoXY(55,23); Write('¿Qué elección? (Esc) ');
    PregOpcion(opcion,'IDR' + ESC,'');
    CASE opcion[1] OF                          ( Ejecuta el procedimiento seleccionado. )
      'I' : Inventarios;
      'D' : Descripciones;
      'R' : Registros;
    END;
  UNTIL opcion[1] = ESC;                       ( Se repite hasta que se presiona la tecla de escape. )
END;
```

Los listados mostrados anteriormente cumplen paso a paso lo requerido en los diagramas de flujo para esta sección.

Por medio de este procedimiento es posible hacer consultas a:

+ Inventarios; que pueden ser tanto de materiales (materias primas y subensambles) como de productos terminados. Se permite la introducción opcional de una cadena (criterio de selección) para ambas secciones, con la cual se presentarán tan solo los inventarios de las claves que inicien con dicha cadena. Los datos que se muestran por columnas en pantalla son:

  + Clave del artículo.
  + Descripción del artículo.
  + Cantidad disponible (sólo para productos terminados).
  + Existencias en la bodega.
  + Precio unitario de venta.
  + Mínimo deseable (sólo materiales).
  + Máximo deseable (sólo materiales).
  + Cantidad pedida del artículo al proveedor de materias primas, a maquila o a producción.

Al llenarse la pantalla, se puede pasar a la pantalla siguiente, a la anterior, o bien, salir de esa sección.

+ Descripciones; se pueden consultar las descripciones de proveedores y distribuidores, con la opción de introducir

una cadena como criterio de selección. Se presentarán tan solo aquellas descripciones cuyas claves inicien con dicho criterio de selección, y los datos registrados en cada descripción se mostrarán en pantallas completas. Se pueden avanzar o retroceder las pantallas para consultar otras descripciones o salir de esa sección.

+ Registros; en esta sección se pueden consultar todas las bases de datos utilizadas para registros y éstos son: pedidos, entradas, salidas, dinero (cobros y pagos) y ajustes. Para los tres primeros conceptos se consulta un registro por pantalla; hay que especificar el folio a consultar y a partir de éste avanzar o retroceder. Para los conceptos restantes, se presentan dieciocho registros por pantalla y se puede avanzar o retroceder la pantalla de consulta.

Con esto queda expuesta la sección de consultas del programa principal S.I.A.C. Esta sección podrá ser ampliada según las necesidades que surjan con el tiempo.

4.3.7 SUBPROGRAMA DE PEDIDOS.

Este subprograma está formado por un solo procedimiento de solapamiento. A continuación se presenta el código fuente de los ficheros incluidos que componen este subprograma.

PO.OVL (Fichero que contiene el código fuente que se muestra.)

```
Overlay PROCEDURE Pedidos;                ( Procedimiento de solapamiento para el control de pedidos. )
|
|  VAR
|    nuevo,                               ( Indica que es un pedido nuevo. )
|    ok: Boolean;                         ( Para identificar errores. )
|    loc_dis,                             ( Localización del distribuidor. )
|    loc_ped: Integer;                    ( Localización del pedido. )
|    opcion: TTecla;                      ( Opción seleccionada. )
|    pedido: TPed;                        ( Datos del pedido. )
|    des_dis: TDesDis;                    ( Datos del distribuidor. )
|
|-------------------------------------------------------------------------------------------------
|  |FUNCTION OkDistribuidor               ( Insertar dicha función. )
|  |
|  |-----------------------------------------------------------------------------------------------
|  |PROCEDURE CargaDistribuidor           ( Insertar dicho procedimiento. )
|  |
|  |-----------------------------------------------------------------------------------------------
|  |FUNCTION OkMaterial                   ( Insertar dicha función. )
|  |
|  |-----------------------------------------------------------------------------------------------
|  |FUNCTION OkProducto                   ( Insertar dicha función. )
|  |
|  |-----------------------------------------------------------------------------------------------
|  |PROCEDURE CargaPedido                 ( Insertar dicho procedimiento. )
|  |
|  |-----------------------------------------------------------------------------------------------
|  |PROCEDURE MostrarCabezaPedidoExtra    ( Insertar dicho procedimiento. )
|  |
|  |-----------------------------------------------------------------------------------------------
|  |PROCEDURE MostrarPartidas             ( Insertar dicho procedimiento. )
|  |
|  |-----------------------------------------------------------------------------------------------
|  |PROCEDURE MostrarOtros                ( Insertar dicho procedimiento. )
|  |
|  |-----------------------------------------------------------------------------------------------
|  |FUNCTION OkCambioPrecios              ( Insertar dicha función. )
|  |
|  |-----------------------------------------------------------------------------------------------
|  |PROCEDURE CambiarPrecios              ( Insertar dicho procedimiento. )
|  |
|  |-----------------------------------------------------------------------------------------------
```

```
FUNCTION OkPrimerosDatos: Boolean;          { Función que verifica que los primeros datos del pedido sean correctos. }

  VAR
    material,                               { Indica que el artículo introducido es un material. }
    otra_part,                              { Indica que se espera la entrada de otra partida. }
    salir,                                  { Permite salir de la sección de introducción de partidas. }
    ok: Boolean;                            { Para identificar errores. }
    cont,                                   { Contador. }
    num_part,                               { Número de partida. }
    loc:Integer;                            { Localización del registro de la base de datos a agregar. }
    tot_part,                               { Total de la partida. }
    suma,                                   { Suma de los totales de las partidas. }
    descuento: Real;                        { Importe del descuento. }
    opcion: TTecla;                         { Opción seleccionada. }
    producto: Array[1..MAX_N_P] OF Boolean; { Indica que el artículo de la partida correspondiente es un producto t. }
    loc_artic: Array[1..MAX_N_P] OF Integer; { Localización del artículo de la partida correspondiente. }

  BEGIN
    WITH pedido DO
      BEGIN
        OkPrimerosDatos := FALSE;
        GotoXY(1,2); ClrEol; Write('Pedido: ');
        IF num_ped <> 0                                 { Si hay pedidos registrados. }
          THEN
            IF num_ped = MAX_PED
              THEN                                      { Si el número de pedidos registrados ha alcanzado el }
                BEGIN                                   { máximo permisible, no continuará. }
                  MensajeErr('No se pueden almacenar más de ' + JusEnte(MAX_PED,6) + ' pedidos.');
                  Exit;
                END
              ELSE num := SigFolio(num_ped_f)           { Si no está en el límite, asigna el folio al pedido. }
          ELSE                                          { Si no hay pedidos registrados, pedirá el folio }
            BEGIN                                       { (distinto de cero) del pedido inicial. }
              MensajeErr('No se tienen pedidos registrados. Hay que especificar el primero.');
              REPEAT
                num := 1;
                GotoXY(9,2); PregEnte(num,5,FALSE);
              UNTIL num > 0;
            END;
        GotoXY(9,2); Write(JusFolio(num):6);            { Presenta en pantalla el folio del pedido. }
        factura := 0;                                   { El folio de factura será cero hasta su facturación. }
        MostrarCabeza_PedidoExtra;                      { Muestra el encabezado del pedido aún sin datos. }
        REPEAT
          ok := FALSE;
          GotoXY(10,4); PregCadMay(clave,TAM_CLAVE);    { Pide la clave del distribuidor. }
          IF clave = ''
            THEN Exit                                   { Si no se introduce clave, termina la ejecución. }
            ELSE                                        { Si se ha introducido una clave: }
              BEGIN
                ok := OkDistribuidor(clave,loc_dis);    { Verifica que esté registrada la clave. }
                IF NOT(ok) THEN
                  MensajeErr('No está registrada esa clave.');
```

```
        IF (clave = 'EVENT') AND con_llave THEN      { Si la clave es "EVENT" y el sistema está protegido }
          BEGIN                                      { con la llave, no aceptará dicha clave. }
            MensajeErr('Clave no autorizada.');
            ok := FALSE;
          END;
      END;
    UNTIL ok;                                        { Repite el proceso hasta que todo esté correcto. }
    na := 5;                                         { Abre el archivo de descripciones de distribuidores. }
    bas[na].nom_arch := nom_arch_des_dis;
    AbrirArchivo;
    CargaDistribuidor(loc_dis,des_dis);              { Carga la descripción del distribuidor en memoria. }
    CerrarArchivo;                                   { Cierra el archivo. }
    nombre := des_dis.nombre;                        { Introduce la descripción del distribuidor al pedido. }
    direc  := des_dis.direc;
    telef  := des_dis.telef;
    ciudad := des_dis.ciudad;
    estado := des_dis.estado;
    c_p    := des_dis.c_p;
    emb_a  := des_dis.emb_a;
    cdp    := des_dis.cdp;
    d      := des_dis.d;
    dp     := des_dis.dp;
    MostrarCabezaPedidoExtra;                        { Muestra el encabezado del pedido ya con datos. }
    REPEAT                                           { Permite la edición de los datos del distribuidor. }
      GotoXY(32,4);  PregCad(nombre,TAM_NOMBRE);
      GotoXY(18,6);  PregCad(direc,TAM_DIREC);
      GotoXY(68,6);  PregCad(telef,TAM_TELEF);
      GotoXY(18,7);  PregCad(ciudad,TAM_CIUDAD);
      GotoXY(18,8);  PregCad(estado,TAM_ESTADO);
      GotoXY(18,9);  PregCad(c_p,TAM_C_P);
      GotoXY(18,10); PregCad(emb_a,TAM_EMB_A);
      GotoXY(13,12); PregByte(d,2);
      IF con_llave AND (d > des_dis.d) THEN           { Si tiene llave, no permite aumento en el descuento. }
        BEGIN
          d := des_dis.d;
          GotoXY(13,12); Write(d);
        END;
      GotoXY(74,12); PregByte(dp,2);
      IF con_llave AND (dp > des_dis.dp) THEN         { Si tiene llave, no permite aumento en el descuento }
        BEGIN                                         { por pronto pago. }
          dp := des_dis.dp;
          GotoXY(74,12); Write(dp);
        END;
      GotoXY(24,14); IF NOT(con_llave) THEN           { Si no tiene llave, permite que se ajusten los días }
        PregByte(cdp,3);                              { de crédito. }
      GotoXY(27,18); Write('¿Todo correcto? (S/N) (Esc) ');
      PregOpcion(opcion,'SN' + ESC,'');               { Pide que se tome una opción. }
      IF opcion[1] = ESC THEN Exit;                   { Si se presiona la tecla escape termina la ejecución. }
    UNTIL opcion[1] = 'S';                            { Repite la edición hasta que se presione "S". }
    n_p := 0;
    num_part := 0;
    MostrarPartidas;                                  { Muestra la pantalla de partidas aún sin datos. }
```

```
REPEAT
  salir := FALSE;
  num_part := Succ(num_part);                    { Obtiene el número de partida siguiente. }
  GotoXY(6,5 + num_part);                        { Coloca el cursor en la linea correspondiente. }
  Write('|      |      |                |              |              |');
  WITH part[num_part] DO
    BEGIN
      REPEAT
        producto[num_part] := FALSE;             { Se asumirá por defecto que no es un producto t. }
        material := FALSE;                       { Se asumirá por defecto que no es un material. }
        otra_part := TRUE;
        GotoXY(7,5 + num_part);
        PregCadMay(artic, TAM_ARTIC);            { Pide la clave del artículo. }
        IF artic = ''
          THEN otra_part := FALSE                { Si no se introduce clave, no pedirá más partidas. }
          ELSE
            IF OkProducto(artic,loc_artic[num_part])
              THEN producto[num_part] := TRUE    { Si es producto terminado, lo indica en 'producto'. }
              ELSE
                IF OkMaterial(artic,loc_artic[num_part]) THEN
                  material := TRUE;              { Si es un material, lo indica en 'material'. }
      UNTIL producto[num_part] OR material OR    { Pide la clave hasta que esté registrada o hasta que }
            NOT(otra_part);                      { ya no se quieran introducir más partidas. }
      IF otra_part THEN                          { Si se quieren introducir más partidas: }
        BEGIN
          IF producto[num_part]
            THEN                                 { Si es un producto terminado presenta en pantalla }
              BEGIN                              { las cantidades disponibles, existentes y pedidas }
                GotoXY(25,5 + num_part);         { del mismo. }
                Write('Disponibles->' + JusEnte(pnt_pt[loc_artic[num_part]]^.ca_dis,6));
                GotoXY(64,5 + num_part);
                Write('Exis->' + JusEnte(pnt_pt[loc_artic[num_part]]^.exist,6));
                GotoXY(47,5 + num_part);
                Write('Pedi->' + JusEnte(pnt_pt[loc_artic[num_part]]^.ca_ped,6));
              END
            ELSE                                 { Si es un material presenta en pantalla las }
              BEGIN                              { cantidades disponibles, existentes y pedidas  del }
                GotoXY(64,5 + num_part);         { mismo. }
                Write('Exis->' + JusEnte(pnt_mp[loc_artic[num_part]]^.exist,6));
                GotoXY(47,5 + num_part);
                Write('Pedi->' + JusEnte(pnt_mp[loc_artic[num_part]]^.ca_ped,6));
              END;
          REPEAT                                 { Pide la cantidad hasta que sea mayor a cero. }
            GotoXY(16,5 + num_part); PregEnte(cant,6,FALSE);
          UNTIL cant > 0;
          IF producto[num_part]                  { Pasa la descripción del artículo al pedido. }
            THEN desc := pnt_pt[loc_artic[num_part]]^.desc
            ELSE desc := pnt_mp[loc_artic[num_part]]^.desc;
          GotoXY(24,5 + num_part); PregCad(desc,20);   { Permite la edición de la descripción. }
          IF producto[num_part]                  { Pasa el precio de venta del artículo al pedido. }
            THEN precio := pnt_pt[loc_artic[num_part]]^.precio_v
            ELSE precio := pnt_mp[loc_artic[num_part]]^.precio_v;
```

```
            GotoXY(47,5 + num_part);
            IF con_llave
              THEN Write(JusReal(precio,14,2))          { Si tiene llave, presenta el precio en pantalla. }
              ELSE PregReal(precio,14,2,FALSE);         { Si no tiene llave, permite editar el precio. }
            tot_part := precio * cant;                  { Calcula y presenta el importe de la partida. }
            GotoXY(64,5 + num_part); Write(JusReal(tot_part,14,2));
          END;
      END;
    IF NOT(otra_part) OR (num_part = MAX_N_P) THEN      { Si ya no se quiere o puede introducir otra partida: }
      BEGIN
        IF NOT(otra_part) THEN                          { Si ya no se quieren más partidas, la última no fue }
          num_part := Pred(num_part);                   { introducida. }
        GotoXY(26,23); Write('¿Hacer cambios? (S/N) (Esc) ');
        PregOpcion(opcion,'SN' + ESC,'');               { Pide que se tome una opción. }
        GotoXY(20,23); ClrEol;
        IF opcion[1] = ESC THEN Exit;                   { Si fue escape, termina la ejecución de la función. }
        IF opcion[1] = 'N'
          THEN salir := TRUE                            { Si fue "N", permite continuar. }
          ELSE num_part := 0;                           { Si fue "S", pasa a la primera partida. }
      END;
  UNTIL salir;                                          { Se repite hasta que estén correctas las partidas. }
  IF num_part = 0 THEN Exit;                            { Si no hay partidas, termina la ejecución. }
  n_p := num_part;
  suma := 0;
  FOR cont := 1 TO num_part DO                          { Calcula la suma de los importes de las partidas. }
    suma := suma + part[cont].precio * part[cont].cant;
  descuento := suma * d/100;                            { Calcula el importe del descuento. }
  importe := (suma - descuento) * (1 + IVA);            { Calcula el importe total del pedido. }
  GotoXY(64,17); Write(JusReal(suma,14,2));             { Presenta en pantalla la suma, descuento, total, }
  GotoXY(64,18); Write(JusReal(descuento,14,2));        { impuesto e importe total. }
  GotoXY(64,19); Write(JusReal(importe/(1 + IVA),14,2));
  GotoXY(33,18); Write(JusReal(importe/(1 + 1/IVA),14,2));
  GotoXY(33,19); Write(JusReal(importe,14,2));
  fech_he := fecha;
  GotoXY(1,21); Write('Fecha para entrega: ');
  PregCad(fech_he,8);                                   { Pide la fecha para entrega. }
  GotoXY(29,23); Write('¿Todo correcto? (S/N) ');
  PregOpcion(opcion,'SN','');                           { Pide que se tome una opción. }
  GotoXY(20,23); ClrEol;
  IF opcion[1] = 'N' THEN Exit;                         { Si fue "N", termina la ejecución. }
  OkPrimerosDatos := TRUE;                              { La función se hace verdadera. }
  IF num_ped = 0 THEN num_ped_i := num;                 { Si no había pedidos registrados, será el inicial. }
  num_ped := Succ(num_ped);                             { Incrementa el número de pedidos registrados en uno. }
  loc_ped := num_ped;
  num_ped_f := num;                                     { El pedido nuevo será el último en el registro. }
  cerr := 'N';
  fech := fecha;
  can  := 0;
  na   := 4;
  EscribirEnte(loc_ped,1,num);                          { Registra los datos generales del pedido. }
  EscribirCar(loc_ped,2,cerr);
  EscribirFecha(loc_ped,3,fech);
```

```
        EscribirByte(loc_ped,4,can);
        EscribirFecha(loc_ped,6,fech_he);
        EscribirCad(loc_ped,8,clave);
        EscribirCad(loc_ped,9,nombre);
        EscribirCad(loc_ped,10,direc);
        EscribirCad(loc_ped,11,telef);
        EscribirCad(loc_ped,12,ciudad);
        EscribirCad(loc_ped,13,estado);
        EscribirCad(loc_ped,14,c_p);
        EscribirCad(loc_ped,15,emb_a);
        EscribirByte(loc_ped,16,cdp);
        EscribirByte(loc_ped,17,d);
        EscribirByte(loc_ped,18,dp);
        EscribirReal(loc_ped,21,importe);
        EscribirEnte(loc_ped,26,factura);
        EscribirByte(loc_ped,28,n_p);
        na := 5;                                    { Abre el archivo de las partidas del pedidos. }
        bas[na].nom_arch := nom_arch_ped_p;
        AbrirArchivo;
        ped_part[loc_ped] := bas[na].num_regs + 1;  { Calcula la localización de la 1ª partida del pedido. }
        FOR cont := 1 TO n_p DO                     { Registra las partidas del pedido. }
          WITH part[cont] DO
            BEGIN
              loc := bas[na].num_regs + 1;
              EscribirEnte(loc,1,num);
              EscribirCad(loc,2,artic);
              EscribirEnte(loc,3,cant);
              EscribirCad(loc,4,desc);
              EscribirReal(loc,5,precio);
            END;
        CerrarArchivo;                              { Cierra el archivo de partidas. }
        na := 2;
        FOR cont := 1 TO num_part DO                { Actualiza la cantidad disponible de productos }
          IF producto[cont] THEN                    { terminados. }
            BEGIN
              pnt_pt[loc_artic[cont]]^.ca_dis := pnt_pt[loc_artic[cont]]^.ca_dis - part[cont].cant;
              EscribirEnte(loc_artic[cont],3,pnt_pt[loc_artic[cont]]^.ca_dis);
            END;
      END;
  END;
```

PA.INC { Fichero que contiene el código fuente que se muestra. }

```
{PROCEDURE Anticipo;                                    { Procedimiento que registra el anticipo a un pedido. }

  VAR
    anticipo: Real;                                     { Monto del anticipo. }
    opcion: TTecla;                                     { Opción tomada. }

  BEGIN
    WITH pedido DO
      BEGIN
        ok := TRUE;                                     { Todo correcto por defecto. }
        IF pag_ant > 0 THEN                             { Si ya hay un anticipo, hay error. }
          BEGIN
            ok := FALSE;
            MensajeErr('Ya se tiene registrado un anticipo.');
          END;
        IF pag_tot <> 0 THEN                            { Si ya está pagado el pedido, hay error. }
          BEGIN
            ok := FALSE;
            MensajeErr('Este pedido ya está pagado.');
          END;
        IF NOT(ok) THEN Exit;                           { Si se localizó algún error, termina la ejecución. }
        IF (fech_ent = '') THEN                         { Si no ha sido entregado: }
          IF OkCambioPrecios THEN                       { Si hay cambio en los precios: }
            BEGIN
              GotoXY(1,23); ClrEol; Write('Hay cambio en los precios (no entregado). ¿Actualizarlos? (S/N) ');
              PregOpcion(opcion,'SN','');               { Pide que se tome una opción. }
              IF opcion[1] = 'S'
                THEN CambiarPrecios                     { Si fue "S", actualiza los precios. }
                ELSE
                  IF con_llave THEN                     { Si fue "N" y tiene llave, hay error. }
                    BEGIN
                      ok := FALSE;
                      MensajeErr('No se puede continuar sin actualizar los precios.');
                    END;
            END;
        IF NOT(ok) THEN Exit;                           { Si se localizó algún error, termina la ejecución. }
        REPEAT
          GotoXY(1,23); ClrEol; Write('¿Importe del anticipo? ');
          PregReal(anticipo,14,2,FALSE);                { Pide el importe del anticipo. }
          IF anticipo >= importe THEN MensajeErr('El pedido sólo es por ' + JusReal(importe,14,2) + '.');
        UNTIL (anticipo > 0) AND (anticipo < importe);  { Se repite hasta que el anticipo sea mayor a cero y }
                                                        { menor al importe del pedido. }
        IF des_dis.saldo >= anticipo
          THEN                                          { Si el saldo acreedor del distribuidor alcanza a }
            BEGIN                                       { cubrir el anticipo: }
              GotoXY(1,23); ClrEol; GotoXY(29,23); Write('¿Todo correcto? (S/N) ');
              PregOpcion(opcion,'SN','');               { Pide que se tome una opción. }
              IF opcion[1] = 'N' THEN Exit;             { Si fue "N", termina el proceso. }
```

```
            na := 4;
            pag_ant := anticipo;
            EscribirReal(loc_ped,22,pag_ant);          { Registra el anticipo. }
            fech_pa := fecha;
            EscribirFecha(loc_ped,23,fech_pa);         { Registra la fecha del anticipo. }
            na := 5;                                   { Abre el archivo de descripciones de distribuidores. }
            bas[na].nom_arch := nom_arch_des_dis;
            AbrirArchivo;
            des_dis.saldo := des_dis.saldo - pag_ant;
            EscribirReal(loc_dis,12,des_dis.saldo);    { Actualiza el saldo acreedor del distribuidor. }
            IF fech_ent <> '' THEN
              BEGIN
                des_dis.cred := des_dis.cred - pag_ant;
                EscribirReal(loc_dis,12,des_dis.cred); { Actualiza el importe del crédito del distribuidor. }
              END;
            CerrarArchivo;                             { Cierra archivo de descripciones de distribuidores. }
            MostrarOtros;                              { Muestra los datos del estado del pedido. }
          END
        ELSE MensajeErr('Sólo tiene ' + JusReal(des_dis.saldo,14,2) + ' de saldo.');
    END;
END;
```

PP.INC { Fichero que contiene el código fuente que se muestra. }

```
PROCEDURE PagoTotal;                                    { Registra el pago total de un pedido. }

  VAR
    ok: Boolean;                                        { Indica que todo está correcto. }
    loc: Integer;                                       { Localización del registro de pago a agregar. }
    por_pagar,                                          { Cantidad pendiente de pago. }
    desc_ppp: Real;                                     { Importe del descuento por pronto pago. }
    opcion: TTecla;                                     { Opción seleccionada. }

  FUNCTION CadenaExterna(acceso: Cad4): Cad78;          { Función para introducción de cadenas en un formato }
                                                        { de impresión en base a una clave de acceso. }
    BEGIN
      CadenaExterna := '';                              { Por defecto será la cadena vacía. }
      WITH pedido DO
        CASE acceso[1] OF                               { Para cada caso del primer carácter de 'acceso' será: }
          'e' : CadenaExterna := JusFolio(nua_not_cred); { El folio de la nota de crédito siguiente. }
          'f' : CadenaExterna := fecha;                 { La fecha actual. }
          'g' : CadenaExterna := JusFolio(num);         { El folio del pedido. }
          'h' : IF factura > 0 THEN                     { Si ya fue facturado; el folio de la factura. }
                  CadenaExterna := JusFolio(factura);
          'i' : CadenaExterna := nombre;                { El nombre del cliente. }
          'j' : CadenaExterna := direc;                 { La dirección. }
          'k' : CadenaExterna := telef;                 { El teléfono. }
          'l' : CadenaExterna := ciudad;                { El nombre de la ciudad. }
          'm' : CadenaExterna := estado;                { El nombre del estado. }
          'n' : CadenaExterna := c_p;                   { El código postal. }
          'o' : CadenaExterna := JusReal(desc_ppp / (1 + IVA),14,2);      { El descuento antes de I.V.A. }
          'p' : CadenaExterna := JusEnte(Round(IVA * 100),2);             { El porcentaje del I.V.A. }
          'q' : CadenaExterna := JusReal(desc_ppp * IVA / (1 + IVA),14,2);        { El importe del I.V.A. }
          'r' : CadenaExterna := JusReal(desc_ppp,14,2);                  { El importe del descuento. }
          's' : CadenaExterna := 'DESCUENTO POR PRONTO PAGO';             { La razón de la nota de crédito. }
          't' : CadenaExterna := 'Y A   A B O N A D A';                   { La cadena "YA ABONADA". }
        END;
    END;

  FUNCTION OkImprimirForma                  { Insertar dicha función. }

  BEGIN
    WITH pedido DO
      BEGIN
        ok := TRUE;                                     { Todo correcto por defecto. }
        IF pag_tot > 0 THEN                             { Si ya está pagado el pedido, hay error. }
          BEGIN
            ok := FALSE;
            MensajeErr('Ya está pagado este pedido.');
```

```
    END;
  IF NOT(ok) THEN Exit;                          { Si se localizó algún error, termina la ejecución. }
  IF (fech_ent = '') AND (pag_ant = 0) THEN      { Si no ha sido entregado y no hay anticipo: }
    IF OkCambioPrecios THEN                      { Si hay cambio en los precios: }
      BEGIN
        GotoXY(1,23); ClrEol;
        Write('Hay cambio en precios (no entregado ni hay anticipo). ¿Actualizarlos? (S/N) ');
        PregOpcion(opcion,'SN',''');             { Pide que se tome una opción. }
        IF opcion[1] = 'S'
          THEN CambiarPrecios                    { Si fue "S", actualiza los precios. }
          ELSE
            IF con_llave THEN                    { Si fue "N" y tiene llave, hay error. }
              BEGIN
                ok := FALSE;
                MensajeErr('No se puede continuar sin actualizar los precios.');
              END;
      END;
  IF NOT(ok) THEN Exit;                          { Si se localizó algún error, termina la ejecución. }
  por_pagar := importe - pag_ant;                { Calcula la cantidad a pagar. }
  desc_ppp := 0;
  IF (fech_ent = '') OR (fech_ent = fecha) THEN  { Si no ha sido entregado o se entregó en la fecha }
    IF dp > 0 THEN                               { actual, calcula el descuento por pronto pago. }
      BEGIN
        MensajeErr('Nota: Se asignará el descuento por pronto pago.');
        desc_ppp := Int(importe * dp + 0.5)/100;
      END;
  IF des_dis.saldo >= por_pagar - desc_ppp
    THEN                                         { Si el saldo acreedor del distribuidor alcanza a }
      BEGIN                                      { cubrir la cantidad a pagar: }
        GotoXY(1,23); ClrEol; GotoXY(29,23); Write('¿Todo correcto? (S/N) ');
        PregOpcion(opcion,'SN',''');             { Pide que se tome una opción. }
        IF opcion[1] = 'N' THEN Exit;            { Si fue "N", termina el proceso. }
        IF desc_ppp > 0 THEN                     { Si hay descuento por pronto pago: }
          BEGIN
            IF NOT(OkImprimirForma('FORMATOS\F_NC.TXT',1)) THEN     { Si no se imprimió la nota de crédito, }
              Exit;                                                 { termina el proceso. }
            na := 5;                             { Abre el archivo de registro de cobros. }
            bas[na].nom_arch := nom_arch_cob;
            AbrirArchivo;
            loc := bas[na].num_regs + 1;         { Calcula la localización del cobro a registrar. }
            EscribirCad(loc,1,clave);            { Registra la nota de crédito como un cobro hecho. }
            EscribirFecha(loc,2,fecha);
            EscribirCad(loc,3,'NC ' + JusFolio(num_not_cred));
            EscribirReal(loc,4,desc_ppp);
            CerrarArchivo;                       { Cierra el archivo de registro de cobros. }
            num_not_cred := SigFolio(num_not_cred);  { Actualiza el contador de folio de nota de crédito. }
            na := 3;                             { Registra el nuevo contador de nota de crédito. }
            EscribirEnte(3,2,num_not_cred);
          END;
        na := 4;
        pag_tot := por_pagar;
        EscribirReal(loc_ped,24,pag_tot);        { Registra el importe del pago total. }
```

```
            fech_pt := fecha;
            EscribirFecha(loc_ped,25,fech_pt);          { Registra la fecha del pago total, que es la actual. }
            IF desc_ppp > 0                             { Registra la variable 'ap' que indica si se aplicó }
              THEN ap := 'S'                            { o no el descuento por pronto pago. }
              ELSE ap := 'N';
            EscribirCar(loc_ped,20,ap);
            na := 5;                                    { Abre el archivo de descripciones de distribuidores. }
            bas[na].nom_arch := nom_arch_des_dis;
            AbrirArchivo;
            des_dis.saldo := des_dis.saldo + desc_ppp - pag_tot;
            EscribirReal(loc_dis,12,des_dis.saldo);     { Registra el nuevo saldo acreedor del distribuidor. }
            IF fech_ent <> '' THEN                      { Si ya fue entregada la mercancía del pedido, }
              BEGIN                                     { el importe del crédito otorgado al distribuidor. }
                des_dis.cred := des_dis.cred - pag_tot;
                EscribirReal(loc_dis,14,des_dis.cred);
              END;
            CerrarArchivo;                              { Cierra el archivo de descripción de distribuidores. }
            MostrarOtros;                               { Muestra el estado del pedido. }
          END
        ELSE MensajeErr('Sólo tiene ' + JusReal(des_dis.saldo,14,2) +
                        ' de saldo, y es por ' + JusReal(por_pagar - desc_ppp,14,2) + '.');
    END;
END;
```

PI.INC (Fichero que contiene el código fuente que se muestra.)

```
{PROCEDURE ImprimirPedido;                                  ( Imprime un pedido. )

 {FUNCTION CadenaExterna(acceso: Cad4): Cad78;              ( Función para introducción de cadenas en un formato )
 {                                                          ( de impresión en base a una clave de acceso. )
 {  BEGIN
 {    CadenaExterna := '';                                  ( Por defecto será la cadena vacia. )
 {    WITH pedido DO
 {      CASE acceso[1] OF                                   ( Para cada caso del primer carácter de 'acceso' será: )
 {        'e' : CadenaExterna := JusFolio(num);             ( El folio del pedido. )
 {        'f' : CadenaExterna := cerr;                      ( "S" para cerrado y "N" para no cerrado. )
 {        'g' : CadenaExterna := fech;                      ( La fecha del pedido. )
 {        'h' : CadenaExterna := JusEnte(can,3);            ( El código de cancelación. )
 {        'i' : CadenaExterna := fech_can;                  ( La fecha de cancelación. )
 {        'j' : CadenaExterna := fech_he;                   ( La fecha hipotética de entrega. )
 {        'k' : CadenaExterna := fech_ent;                  ( La fecha de entrega. )
 {        'l' : CadenaExterna := clave;                     ( La clave del distribuidor. )
 {        'm' : CadenaExterna := nombre;                    ( El nombre del cliente. )
 {        'n' : CadenaExterna := direc;                     ( La dirección. )
 {        'o' : CadenaExterna := telef;                     ( El teléfono. )
 {        'p' : CadenaExterna := ciudad;                    ( El nombre de la ciudad. )
 {        'q' : CadenaExterna := estado;                    ( El nombre del estado. )
 {        'r' : CadenaExterna := c_p;                       ( El código postal. )
 {        's' : CadenaExterna := emb_a;                     ( El lugar de embarque. )
 {        't' : IF cdp = 0                                  ( Las condiciones de pago. )
 {                THEN CadenaExterna := 'C.O.D.'
 {                ELSE CadenaExterna := JusEnte(cdp,3) + ' Dias';
 {        'u' : CadenaExterna := JusEnte(d,2);              ( El descuento en porcentaje. )
 {        'v' : CadenaExterna := JusEnte(dp,2);             ( El descuento por pronto pago en porcentaje. )
 {        'w' : CadenaExterna := fech_cob;                  ( La fecha para cobro. )
 {        'x' : CadenaExterna := ap;                        ( "S" si se aplicó el 'dp' y "N" si no. )
 {        'y' : CadenaExterna := JusReal(importe,14,2);     ( El importe del pedido. )
 {        'z' : CadenaExterna := JusReal(pag_ant,14,2);     ( El importe del anticipo. )
 {        '{' : CadenaExterna := fech_pa;                   ( La fecha en que se recibió el anticipo. )
 {        '|' : CadenaExterna := JusReal(pag_tot,14,2);     ( El importe del pago total. )
 {        '}' : CadenaExterna := fech_pt;                   ( La fecha en que se recibió el pago total. )
 {        '~' : IF factura > 0 THEN                         ( Si ya se facturó; el folio de la factura. )
 {                CadenaExterna := JusFolio(factura);
 {        ' ' : CadenaExterna := fech_fac;                  ( La fecha de facturación. )
 {        'Ç' : CadenaExterna := JusEnte(n_p,2);            ( El número de partidas del pedido. )
 {        'ü' : CadenaExterna := JusReal(importe/(1 -d/100)/(1 + IVA),14,2);          ( El importe de las partidas. )
 {        'é' : CadenaExterna := JusReal(importe/(1 -d/100)/(1 + IVA) * d/100,14,2);  ( El descuento. )
 {        'â' : CadenaExterna := JusReal(importe/(1 + IVA),14,2);                     ( El total antes de I.V.A. )
 {        'ä' : CadenaExterna := JusReal(importe/(1 + IVA) * IVA,14,2);               ( El importe del I.V.A. )
 {        'à' : CadenaExterna := JusReal(importe,14,2);     ( El importe total del pedido. )
 {        'å' : CadenaExterna := JusEnte(Round(IVA * 100),2);  ( El I.V.A. en porcentaje. )
 {        'æ' : IF Ord(acceso[2]) - 47 <= n_p THEN          ( Para la interpretación de las partidas. )
 {                WITH part[Ord(acceso[2]) - 47] DO         ( Identifica el número de partida. )
```

```
                CASE acceso[3] OF                          { Para cada caso del tercer carácter de 'acceso' será: }
                  '1' : CadenaExterna := JusFolio(num);    { El número de pedido asociado a la partida. }
                  '2' : CadenaExterna := artic;            { La clave del artículo. }
                  '3' : CadenaExterna := JusEnte(cant,6);  { La cantidad. }
                  '4' : CadenaExterna := desc;                          { La descripción del artículo. }
                  '5' : CadenaExterna := JusReal(precio,14,2);          { El precio unitario del artículo. }
                  '6' : CadenaExterna := JusReal(precio * cant,14,2);   { El importe de la partida. }
                  ELSE MensajeErr('El formato tiene especificado un campo de partidas no definido');
                END;
        ELSE                                               { Si no se pudo interpretar, regresa la cadena de }
          BEGIN                                            { acceso. }
            MensajeErr('El formato tiene especificado un campo no definido.');
            CadenaExterna := '@' + acceso;
          END;
      END;
  END;

{FUNCTION OkImprimirForma                  { Insertar dicha función. }
```

```
  BEGIN
    ok := OkImprimirForma('FORMATOS\F_P.TXT',0);                  { Manda a imprimir el pedido sin copias. }
  END;
```

PF.INC { Fichero que contiene el código fuente que se muestra. }

```
{PROCEDURE Facturar;                                    { Procedimiento para facturar el pedido actual. }
 |
 |  VAR
 |    nueva,                                            { Para indicar si es una factura nueva. }
 |    ok: Boolean;                                      { Para identificar errores. }
 |
 |  {FUNCTION CadenaExterna                 { Insertar dicha función (del procedimiento 'Imprimir'). }
 |   |
 |  {FUNCTION OkImprimirForma               { Insertar dicha función. }
 |   |
 |  BEGIN
 |    WITH pedido DO
 |      BEGIN
 |        IF factura > 0 THEN                           { Si ya fue facturado: }
 |          IF con_llave
 |            THEN                                     { Si tiene llave, no continuará el proceso. }
 |              BEGIN
 |                MensajeErr('Este pedido ya está facturado (factura ' + JusFolio(factura) +').');
 |                Exit;
 |              END
 |            ELSE                                     { Si no tiene llave, pide que se confirme para poder }
 |              BEGIN                                  { continuar el proceso. Si la respuesta fue "N" }
 |                GotoXY(1,23); ClrEol;                { termina el proceso y si fue "S" continuará. }
 |                GotoXY(17,23); Write('El pedido ya fue facturado. ¿continuar? (S/N) ');
 |                PregOpcion(opcion,'SN','');
 |                IF opcion[1] = 'N' THEN Exit;
 |              END;
 |        nueva := FALSE;                              { Por defecto será una nueva factura. }
 |        IF factura = 0 THEN                          { Si no ha sido facturado: }
 |          BEGIN
 |            nueva := TRUE;                          { Indica que será una nueva factura. }
 |            fech_fac := fecha;                      { Asigna la fecha actual como la fecha de facturación. }
 |            IF num_fact < 0 THEN num_fact := 1;     { Si el folio de la factura es inválido, lo hace uno. }
 |            factura := num_fact;                    { Asigna el folio de la factura. }
 |          END;
 |        ok := OkImprimirForma('FORMATOS\F_F.TXT',0); { Imprime la factura sin copias. }
 |        IF nueva THEN                               { Si es una factura nueva: }
 |          IF ok
 |            THEN                                    { Si se imprimió correctamente: }
 |              BEGIN
 |                na := 4;
 |                EscribirEnte(loc_ped,26,factura);   { Escribe en el disco el folio de la factura. }
 |                EscribirFecha(loc_ped,27,fech_fac); { Escribe en el disco la fecha de facturación. }
 |                na := 3;
 |                num_fact := SigFolio(num_fact);     { Calcula el folio para la próxima factura. }
```

```
                EscribirEnte(2,2,num_fact);          { Escribe en el disco el próximo folio de factura. }
                MostrarOtros;
              END
            ELSE                                     { Si no se imprimió correctamente, regresa a los }
              BEGIN                                  { valores originales. }
                factura := 0;
                fech_fac := '';
              END;
      END;
  END;
```

PC.INC { Fichero que contiene el código fuente que se muestra. }

```
!PROCEDURE Cancelar;                                 { Cancela el pedido actual. }
!
!  VAR
!    ok,                                             { Indica que está correcto. }
!    artic_espec: Boolean;                           { Indica que es un artículo especial. }
!    razon: Byte;                                    { Razón codificada de la cancelación. }
!    loc,                                            { Para memorizar localizaciones de registros. }
!    cont: Integer;                                  { Contador de partidas. }
!    monto_pago: Real;                               { Monto de la nota de crédito. }
!    opcion: TTecla;                                 { Opción seleccionada. }
!
!  !FUNCTION CadenaExterna(acceso: Cad4): Cad78;     { Función para introducción de cadenas en un formato }
!  !                                                 { de impresión en base a una clave de acceso. }
!  !  BEGIN
!  !    CadenaExterna := '';                         { Por defecto será la cadena vacía. }
!  !    WITH pedido DO
!  !      CASE acceso[1] OF                          { Para cada caso del primer carácter de 'acceso' será: }
!  !        'e' : CadenaExterna := JusFolio(num_not_cred);   { El folio de la nota de crédito siguiente. }
!  !        'f' : CadenaExterna := fecha;            { La fecha actual. }
!  !        'g' : CadenaExterna := JusFolio(nua);    { El folio del pedido. }
!  !        'h' : IF factura > 0 THEN                { Si ya fue facturado; el folio de la factura. }
!  !                CadenaExterna := JusFolio(factura);
!  !        'i' : CadenaExterna := nombre;           { El nombre del cliente. }
!  !        'j' : CadenaExterna := direc;            { La dirección. }
!  !        'k' : CadenaExterna := telef;            { El teléfono. }
!  !        'l' : CadenaExterna := ciudad;           { El nombre de la ciudad. }
!  !        'm' : CadenaExterna := estado;           { El nombre del estado. }
!  !        'n' : CadenaExterna := c_p;              { El código postal. }
!  !        'o' : CadenaExterna := JusReal(monto_pago / (1 + IVA),14,2);   { El importe de la n. c. antes de I.V.A. }
!  !        'p' : CadenaExterna := JusEnte(Round(IVA * 100),2);            { El porcentaje del I.V.A. }
!  !        'q' : CadenaExterna := JusReal(monto_pago * IVA / (1 + IVA),14,2);   { El importe del I.V.A. }
!  !        'r' : CadenaExterna := JusReal(monto_pago,14,2);               { El importe de la nota de crédito. }
!  !        's' : CadenaExterna := 'CANCELACION DEL PEDIDO';               { La razón de la nota de crédito. }
!  !        't' : IF pedido.clave <> 'EVENT' THEN                          { La cadena "YA ABONADA". }
```

```
        CadenaExterna := 'Y A   A B O N A D A';
    END;
  END;

FUNCTION OkImprimirForma                ( Insertar dicha función. )

BEGIN
  WITH pedido DO
    BEGIN
      ok := TRUE;                                   ( Por defecto, está correcto. )
      IF fech_ent <> '' THEN                        ( Si ya fue entregado, hay error. )
        BEGIN
          ok := FALSE;
          MensajeErr('No se puede cancelar un pedido entregado.');
        END;
      IF fech_fac <> '' THEN                        ( Si ya fue facturado, hay error. )
        BEGIN
          ok := FALSE;
          MensajeErr('No se puede cancelar un pedido facturado.');
        END;
      artic_espec := FALSE;                         ( Por defecto no habrá artículos especiales. )
      FOR cont := 1  TO n_p DO                      ( Contados desde uno hasta el número de partidas. )
        IF (Pos('11',part[cont].artic) = 1) OR      ( Si la clave del artículo de la partida en análisis )
           (Pos('12',part[cont].artic) = 1) OR      ( cuaple con las condiciones necesarias, será la clave )
           (Pos('31',part[cont].artic) = 1) THEN    ( de un artículo especial. )
            artic_espec := TRUE;
      IF artic_espec THEN                           ( Si se localizó algún artículo especial: )
        IF con_llave
        THEN                                        ( Si tiene llave, hay error. )
          BEGIN
            ok := FALSE;
            MensajeErr('No se puede cancelar un pedido con artículos especiales.');
          END
        ELSE                                        ( Si no tiene llave: )
          IF ok THEN                                ( Si está correcto, pide que se confirme la orden de )
            BEGIN                                   ( cancelación. )
              GotoXY(1,23); ClrEol; Write('Este pedido contiene productos especiales. ¿Cancelarlo? (S/N) ');
              PregOpcion(opcion,'SN','');
              IF opcion[1] = 'N' THEN ok := FALSE;  ( Si se presionó "N", indicará error. )
            END;
      IF NOT(ok) THEN Exit;                         ( Si no está correcto, termina la ejecución. )
      monto_pago := pag_ant + pag_tot;
      IF ap = 'S' THEN                              ( Si se asignó el descuento por pronto pago, deducirá )
        BEGIN                                       ( el monto del descuento. )
          MensajeErr('Este pedido tiene el descuento PPP, se abonará la diferencia.');
          monto_pago := monto_pago * (1 - dp/100);
        END;
      razon := 0;
      REPEAT                                        ( Pide el código de la razón de cancelación. )
        GotoXY(1,23); ClrEol; Write('¿Razón para cancelar? (código) ');
```

```
        PregByte(razon,3);
      UNTIL razon > 0;                                  { Lo vuelve a pedir hasta que sea mayor a cero. }
      GotoXY(1,23); ClrEol;                             { Pide que se confirme la orden de cancelación. }
      Write('¿Seguro de querer cancelar el pedido? (S/N) ');
      PregOpcion(opcion,'SN','');
      IF opcion[1] = 'N' THEN Exit;                     { Si se canceló la orden con "N", termina el proceso. }
      IF monto_pago > 0 THEN                            { Si se asignó algún pago al pedido: }
        BEGIN
          IF NOT(OkImprimirForma('FORMATOS\F_NC.TXT',1)) THEN { Si no se imprimió correctamente la nota de crédito, }
            Exit;                                       { termina el proceso. }
          IF clave <> 'EVENT' THEN                      { Si el cliente no es eventual: }
            BEGIN
              na := 5;                                  { Abre el archivo de descripción de distribuidores. }
              bas[na].noa_arch := noa_arch_des_dis;
              AbrirArchivo;
              des_dis.saldo := des_dis.saldo + monto_pago;  { Actualiza el saldo del distribuidor. }
              EscribirReal(loc_dis,12,des_dis.saldo);
              CerrarArchivo;                            { Cierra el archivo de descripción de distribuidores. }
              bas[na].noa_arch := noa_arch_cob;         { Abre el archivo de registro de cobros. }
              AbrirArchivo;
              loc := bas[na].nua_regs + 1;              { Localiza el registro de cobro a agregar. }
              EscribirCad(loc,1,clave);                 { Registra el cobro en base a una nota de crédito. }
              EscribirFecha(loc,2,fecha);
              EscribirCad(loc,3,'NC ' + JusFolio(nua_not_cred));
              EscribirReal(loc,4,monto_pago);
              CerrarArchivo;                            { Cierra el archivo de registro de cobros. }
            END;
          na := 3;
          nua_not_cred := SigFolio(nua_not_cred);       { Calcula el folio de la nota de crédito próxima. }
          EscribirEnte(3,2,num_not_cred);               { Escribe en el disco dicho folio. }
        END;
      na := 4;
      cerr := 'S';                                      { Indica el pedido que ya fue cerrado. }
      can := razon;                                     { Establece la razón de cancelación en el pedido. }
      fech_can := fecha;                                { La fecha de cancelación será la actual. }
      EscribirCar(loc_ped,2,cerr);                      { Escribe en el disco los cambios hechos al pedido. }
      EscribirByte(loc_ped,4,can);
      EscribirFecha(loc_ped,5,fech_can);
      na := 2;
      FOR cont := 1 TO n_p DO                           { Actualiza en memoria y disco la cantidad disponible }
        WITH part[cont] DO                              { de productos terminados. }
          IF OkProducto(artic,loc) THEN
            BEGIN
              pnt_pt[loc]^.ca_dis := pnt_pt[loc]^.ca_dis + cant;
              EscribirEnte(loc,3,pnt_pt[loc]^.ca_dis);
            END;
    END;
  END;
```

PI.OVL (Fichero que contiene el código fuente que se muestra.)

```
BEGIN
  na := 4;                                  ( Abre el archivo de pedidos. )
  bas[na].nom_arch := nom_arch_ped;
  AbrirArchivo;
  REPEAT
    ClrScr;                                 ( Presenta el encabezado. )
    GotoXY(33,1); Write('P E D I D O S');
    GotoXY(70,1); Write(fecha);
    FillChar(pedido,SizeOf(pedido),0);      ( Borra los datos que contiene la variable que almacena los pedidos y  )
    FillChar(des_dis,SizeOf(des_dis),0);    ( las descripciones de distribuidores. )
    ok := TRUE;                             ( Todo correcto por defecto. )
    GotoXY(1,2); Write('¿Nuevo pedido? (S/N) (Esc) ');
    PregOpcion(opcion,'SN' + ESC,'');       ( Pide que se tome una opción. )
    IF opcion[1] = ESC THEN                 ( Si se presionó la tecla de escape, cierra el archivo de pedidos y se )
      BEGIN                                 ( sale del procedimiento. )
        na := 4;
        CerrarArchivo;
        Exit;
      END;
    WITH pedido DO
      BEGIN
        IF opcion[1] = 'S'
          THEN                              ( Si se presionó la "S", indica en 'nuevo' que es un pedido nuevo y )
            BEGIN                           ( ejecuta la función 'OkPrimerosDatos' para que se introduzcan los )
              nuevo := TRUE;                ( datos iniciales del pedido. )
              ok := OkPrimerosDatos;
            END
          ELSE nuevo := FALSE;              ( Si se presionó la "N", indica en 'nuevo' que es un pedido ya existente. )
        IF ok THEN                          ( Si todo va bien: )
          BEGIN
            IF num_ped = 0 THEN             ( Si no hay pedidos registrados, está incorrecto. )
              BEGIN
                MensajeErr('No hay pedidos registrados.');
                ok := FALSE;
              END;
            IF NOT(nuevo) AND ok THEN       ( Si no es un pedido nuevo y todo está correcto: )
              BEGIN
                num := num_ped_i;           ( Propone el primer pedido registrado. )
                REPEAT
                  GotoXY(1,2); ClrEol;
                  Write('Pedido: (',JusFolio(num_ped_i),'..',JusFolio(num_ped_f),') ');
                  PregEnte(num,5,FALSE);                    ( Pide el folio del pedido a utilizar. )
                  GotoXY(22,2); Write(JusFolio(num),' ');
                UNTIL OkFolio(num,num_ped_i,num_ped_f);     ( Se repite hasta que el folio elegido esté correcto. )
                na := 5;                                    ( Abre el archivo de pedidos por partida. )
                bas[na].nom_arch := nom_arch_ped_p;
                AbrirArchivo;
                CargaPedido(num,loc_ped,pedido);            ( Lee del disco los datos del pedido seleccionado. )
                na := 5;
```

```
        CerrarArchivo;                          { Cierra el archivo de pedidos por partida. }
        ok := FALSE;                            { Está incorrecto por defecto. }
        IF cerr = 'N'
          THEN ok := TRUE                       { Si no es un pedido cerrado, está correcto. }
          ELSE MensajeErr('El pedido ya está cerrado. Sólo se puede consultar.');
        IF ok THEN                              { Si todo está correcto: }
          BEGIN
            IF OkDistribuidor(clave,loc_dis)
              THEN                              { Si el distribuidor está todavía registrado. }
               BEGIN
                 na := 5;                       { Abre el archivo de descripciones de distribuidores. }
                 has[na].nom_arch := nom_arch_des_dis;
                 AbrirArchivo;
                 CargaDistribuidor(loc_dis,des_dis);  { Lee del disco la descripción del distribuidor. }
                 CerrarArchivo;                 { Cierra el archivo de descripciones de dist. }
               END
              ELSE                              { Si el distribuidor ya no está registrado: }
               BEGIN                            { No permite continuar con el mismo pedido. }
                 MensajeErr('Ya no está registrado ese distribuidor. No se puede continuar.');
                 ok := FALSE;
               END;
          END;
      END;
    IF ok THEN                                  { Si todo está correcto: }
      BEGIN
        MostrarOtros;                           { Presenta en pantalla los datos generales del pedido }
        Posibilidad(1,21,'Anticipo'); ClrEol;   { y las diferentes posibilidades de elección. }
        Posibilidad(1,22,'Pago total'); ClrEol;
        Posibilidad(25,21,'Imprimir pedido');
        Posibilidad(25,22,'Facturar');
        Posibilidad(50,21,'Cancelar pedido');
        REPEAT
          GotoXY(1,23); ClrEol; Write('¿Qué elección? <F1 Cabecera> <F2 Partidas> <F3 Otros> <Esc> ');
          PregOpcion(opcion,'APIFC' + ESC,#59#60#61);   { Pide que se tome una opción. }
          CASE opcion[2] OF                     { Ejecuta el procedimiento indicado. }
            #59 : MostrarCabezaPedidoExtra;
            #60 : MostrarPartidas;
            #61 : MostrarOtros;
          END;
          CASE opcion[1] OF                     { Ejecuta el procedimiento indicado. }
            'A' : Anticipo;
            'P' : PagoTotal;
            'I' : ImprimirPedido;
            'F' : Facturar;
            'C' : Cancelar;
          END;
        UNTIL (opcion[1] = ESC) OR (cerr = 'S');    { Se repite hasta que se presione la tecla 'Esc' o }
```

```
                opcion[1] := #0;                    { hasta que se cierre el pedido. }
              END;
            END;
          END;
      UNTIL opcion[1] = ESC;                        { Se repite el procedimiento hasta que se presione la }
  END;                                              { tecla 'Esc' al inicio del procedimiento. }
```

Este procedimiento se utiliza para registrar nuevos pedidos y también en casi todas las operaciones relativas a los mismos. La descripción del funcionamiento, tanto de esta sección como de las restantes, se presentó en el Capítulo III.

4.3.8 SUBPROGRAMA DE ENTRADAS.

Este subprograma está formado por un solo procedimiento de solapamiento que a su vez contiene otros dos procedimentos de solapamiento. A continuación se muestran los listados de los ficheros incluidos que forman el procedimiento de entradas.

E0.OVL (Fichero que contiene el código fuente que se muestra.)

```
{Overlay PROCEDURE Entradas;                    ( Procedimiento de solapamiento para el control de )
                                                ( entradas. )
  VAR
    loc_ent: Integer;                           ( Localización de la entrada. )
    opcion: TTecla;                             ( Opción seleccionada. )
    entrada: TEnt;                              ( Datos de la entrada. )

  {FUNCTION OkMaterial                          ( Insertar dicha función. )

  {FUNCTION OkProducto                          ( Insertar dicha función. )

  {FUNCTION OkProveedor                         ( Insertar dicha función. )

  {FUNCTION OkDistribuidor                      ( Insertar dicha función. )

  {Overlay PROCEDURE Nuevas;                        ( Procedimiento de solapamiento para el registro de )
                                                    ( nuevas entradas. )
      VAR
        opcion: TTecla;                             ( Opción seleccionada. )
```

EC.INC { Fichero que contiene el código fuente que se muestra. }

```
{PROCEDURE Compras;                                    { Registro de entradas por concepto de compras. }
;
;  LABEL
;    Principio;                                        { Para regresar al principio del procedimiento. }
;
;  VAR
;    ok,                                               { Indica que está correcto. }
;    otra_part: Boolean;                               { Indica que se va a introducir otra partida. }
;    cont,                                             { Contador. }
;    num_part,                                         { Contador de partida. }
;    loc_pro,                                          { Localización del proveedor. }
;    loc:       Integer;                               { Localización de algún registro. }
;    tot_part,                                         { Importe de la partida. }
;    suma,                                             { Suma de los importes de las partidas. }
;    descuento,                                        { Importe del descuento. }
;    total:     Real;                                  { Total a pagar. }
;    opcion: TTecla;                                   { Opción seleccionada. }
;    nombre: TNombre;                                  { Nombre del proveedor. }
;    loc_artic: Array[1..MAX_N_P] OF Integer;          { Localización de la materia prima para cada partida. }
;
;  BEGIN
;    REPEAT
;      Principio:
;      ClrScr;                                         { Presenta el encabezado. }
;      GotoXY(25,1); Write('ENTRADAS/NUEVAS-> C O M P R A S');
;      GotoXY(70,1); Write(fecha);
;      FillChar(entrada,SizeOf(entrada),0);            { Borra totalmente la variable 'entrada'. }
;      WITH entrada DO
;        BEGIN
;          GotoXY(1,2); ClrEol; Write('Entrada: ');
;          IF num_ent <> 0
;            THEN                                      { Si hay entradas registradas: }
;              IF num_ent = MAX_ENT
;                THEN                                  { Si ya no se aceptan más entradas, termina la }
;                  BEGIN                               { ejecución. }
;                    MensajeErr('No se pueden almacenar más de ' + JusEnte(MAX_ENT,6) + ' entradas.');
;                    Exit;
;                  END
;                ELSE num := SigFolio(num_ent_f)       { Si se puede aceptar la entrada, asigna su folio. }
;            ELSE                                  { Si no hay entradas registradas, pide que se especifique el }
;              BEGIN                               { folio (distinto de cero) de la primera entrada. }
;                MensajeErr('No se tienen entradas registradas. Hay que especificar la primera.');
;                REPEAT
;                  num := 1;
;                  GotoXY(10,2); PregEnte(num,5,FALSE);
;                UNTIL num > 0;
;              END;
;          GotoXY(10,2); Write(JusFolio(num),' ');     { Presenta en pantalla el folio de la entrada. }
```

```
  GotoXY(15,2); Write('Clave:');
  REPEAT
    REPEAT
      GotoXY(22,2); PregCadMay(clave,TAM_CLAVE);        { Pide la clave del proveedor. }
      IF clave = '' THEN Exit;                          { Si no se introduce la clave, termina la ejecución. }
    UNTIL OkProveedor(clave, loc_pro);                  { Se repite hasta que la clave esté registrada. }
    na := 5;                                            { Abre el archivo de descripciones de proveedores. }
    bas[na].nom_arch := nom_arch_des_pro;
    AbrirArchivo;
    LeerCad(loc_pro,2,nombre);                          { Lee el nombre del proveedor. }
    CerrarArchivo;                                      { Cierra el archivo de descr. de proveedor. }
    GotoXY(30,2);  Write('Nombre: ',JusCad(nombre,40)); { Presenta el nombre en pantalla. }
    GotoXY(1,3);   Write('Su pedido: ');
    PregEnte(pedido,5,FALSE);                           { Pide el folio del pedido del proveedor. }
    GotoXY(12,3);  Write(JusFolio(pedido),' ');
    GotoXY(30,3);  Write('A pagar: ');
    PregReal(importe,14,2,FALSE);                       { Pide la cantidad a pagar por la compra. }
    GotoXY(55,3);  Write('Fecha de pago: ');
    fech_pag := fecha; PregCad(fech_pag,8);             { Pide la fecha para pago. }
    GotoXY(29,23); Write('¿Está correcto? (S/N) ');
    PregOpcion(opcion,'SN','');                         { Pide que se tome una opción. }
    GotoXY(29,23); ClrEol;
  UNTIL opcion[1] = 'S';                                { Se repite hasta que se presione "S". }
  GotoXY(1,4);                                          { Presenta la pantalla para las partidas. }
  WriteLn('
  WriteLn('  #P  |  ARTIC  |  CANT  |      PRECIO      |      IMPORTE      |   PRECIO REAL   ');
  WriteLn('
  WriteLn('   1 |          |        |                  |                   |                 ');
  WriteLn('   2 |          |        |                  |                   |                 ');
  WriteLn('   3 |          |        |                  |                   |                 ');
  WriteLn('   4 |          |        |                  |                   |                 ');
  WriteLn('   5 |          |        |                  |                   |                 ');
  WriteLn('   6 |          |        |                  |                   |                 ');
  WriteLn('   7 |          |        |                  |                   |                 ');
  WriteLn('   8 |          |        |                  |                   |                 ');
  WriteLn('   9 |          |        |                  |                   |                 ');
  WriteLn('  10 |          |        |                  |                   |                 ');
  WriteLn('
  WriteLn('                                      SUMA                                       ');
  WriteLn('  I.V.A.                           DESCUENTO                                    ');
  WriteLn(' TOT/IVA                             TOTAL                                     ');
  WriteLn('
  num_part := 0;
  REPEAT
    num_part := Succ(num_part);                         { Pasa a la partida siguiente. }
    GotoXY(6,6 + num_part);                             { Coloca el cursor en la partida correspondiente. }
    Write('|          |        |                  |                   |                 |');
    WITH part[num_part] DO
      BEGIN
        ok := FALSE;                                    { Por defecto no está correcto. }
        REPEAT
          otra_part := TRUE;                            { Por defecto se espera otra partida. }
```

```
        GotoXY(8,6 + num_part);
        PregCadMay(artic, TAM_ARTIC);                { Pide la clave del artículo. }
        IF artic = ''
          THEN otra_part := FALSE                    { Si no se introdujo clave, no pasará a otra partida. }
          ELSE                                       { Si se introdujo alguna clave: }
            IF OkMaterial(artic,loc_artic[num_part])
              THEN                                   { Si es un material: }
                IF (artic[1] <> CLA_SUBE) AND
                   (artic[1] <> CLA_MAQ)
                THEN                                 { Si es una materia prima, está correcto y, }
                  BEGIN                              { si no es el mismo proveedor lo notificará. }
                    ok := TRUE;
                    IF pnt_ap[loc_artic[num_part]]^.clave <> clave THEN
                      MensajeErr('Nota: este artículo lo surtía ' + pnt_ap[loc_artic[num_part]]^.clave);
                  END
                ELSE MensajeErr('No se pueden comprar subensambles ni productos de maquila.')
              ELSE MensajeErr ('No está registrado este material.');
      UNTIL ok OR NOT(otra_part);                    { Se repite hasta que esté correcto y no se deseen más partidas. }
      ok := FALSE;                                   { Por defecto no está correcto. }
      IF otra_part THEN                              { Si se quiere otra partida: }
        BEGIN
          REPEAT                                     { Pide la cantidad de unidades. }
            cant := pnt_ap[loc_artic[num_part]]^.ca_ped;
            GotoXY(17,6 + num_part); Write('      ');
            GotoXY(17,6 + num_part); PregEnte(cant,6,FALSE);
          UNTIL cant > 0;                            { Lo repite hasta que la cantidad es mayor que cero. }
          precio := pnt_ap[loc_artic[num_part]]^.precio_ult;
          GotoXY(26,6 + num_part); PregReal(precio,14,2,FALSE);  { Pide el precio de la materia prima. }
          tot_part := precio * cant;                 { Calcula y presenta el total de la partida. }
          GotoXY(43,6 + num_part); Write(JusReal(tot_part,14,2));
        END;
       IF NOT(otra_part) OR                          { Si ya no se quiere o puede añadir otra partida: }
          (num_part = MAX_N_P) THEN
        BEGIN
          IF NOT(otra_part) THEN                     { Si ya no se quiere otra partida, ajusta el número de }
            num_part := Pred(num_part);              { partidas. }
          GotoXY(29,23); Write('¿Hacer cambios? (S/N) ');
          PregOpcion(opcion,'SN',' ');               { Pide que se tome una opción. }
          GotoXY(29,23); ClrEol;
          IF opcion[1] = 'N'
            THEN ok := TRUE                          { Si fue "N", indicará que todo está correcto. }
            ELSE num_part := 0;                      { Si fue "S", pasa a la primera partida. }
        END;
    END;
  UNTIL ok;                                          { Lo repite hasta que todo esté correcto. }
  IF num_part = 0 THEN Goto Principio;               { Si no se introdujeron partidas, pasa a 'Principio'. }
  n_p := num_part;
  suma := 0;
  FOR cont := 1 TO n_p DO                            { Calcula la suma de los importes de las partidas. }
    suma := suma + part[cont].precio * part[cont].cant;
  suma := Int(suma * 100 + 0.5)/100;                 { Redondea la cifra a dos decimales. }
  GotoXY(43,18); Write(JusReal(suma,14,2));          { Presenta en pantalla la suma. }
```

```
total := 0;
REPEAT
  GotoXY(43,20); PregReal(total,14,2,FALSE);          { Pide el total de la compra. }
UNTIL total <= suma;                                  { Lo repite hasta si el total es mayor a la suma. }
descuento := suma - total;                            { Calcula el descuento que se recibió. }
GotoXY(43,19); Write(JusReal(descuento,14,2));        { Presenta dicho descuento. }
GotoXY(12,19); Write(JusReal(total * IVA,14,2));      { Presenta el importe del I.V.A. }
GotoXY(12,20); Write(JusReal(total * (1 + IVA),14,2)); { Presenta el total ya con I.V.A. }
FOR cont := 1 TO n_p DO                               { Calcula el precio real de cada artículo antes de }
  WITH part[cont] DO                                  { I.V.A. }
    BEGIN
      precio := precio * total / suma;
      GotoXY(60,6 + cont); Write(JusReal(precio,14,2));
    END;
GotoXY(29,23); Write('¿Todo correcto? (S/N) ');
PregOpcion(opcion,'SN','');                           { Pide que se tome una opción. }
GotoXY(29,23); ClrEol;
IF opcion[1] = 'N' THEN Goto Principio;               { Si fue "N", pasa a 'Principio'. }
na := 1;
FOR cont := 1 TO n_p DO                               { Contador desde uno hasta el número de partidas. }
  WITH part[cont] DO
    BEGIN                                             { Actualiza las existencias y la cantidad pedida. }
      pnt_ap[loc_artic[cont]]^.exist := pnt_ap[loc_artic[cont]]^.exist + cant;
      EscribirEnte(loc_artic[cont],2,pnt_ap[loc_artic[cont]]^.exist);
      pnt_ap[loc_artic[cont]]^.ca_ped := pnt_ap[loc_artic[cont]]^.ca_ped - cant;
      IF pnt_ap[loc_artic[cont]]^.ca_ped < 0 THEN     { No acepta una cantidad pedida negativa. }
        pnt_ap[loc_artic[cont]]^.ca_ped := 0;
      IF pnt_ap[loc_artic[cont]]^.ca_ped > 0 THEN     { Si aún quedarán artículos pendientes de entrega, }
        BEGIN                                         { da la oportunidad de que se ajuste a cero. }
          GotoXY(1,22); Write('Quedan ',JusEnte(pnt_ap[loc_artic[cont]]^.ca_ped ,6), ' ',
                              artic,' pendientes.');
          GotoXY(1,23); Write('¿Que queden pendientes? (S/N) ');
          PregOpcion(opcion,'SN','');
          IF opcion[1] = 'N' THEN pnt_ap[loc_artic[cont]]^.ca_ped := 0;
          GotoXY(1,22); ClrEol;
          GotoXY(1,23); ClrEol;
        END;                                          { Actualiza la cantidad pedida. }
      EscribirEnte(loc_artic[cont],3,pnt_ap[loc_artic[cont]]^.ca_ped);
      IF pnt_ap[loc_artic[cont]]^.clave <> clave THEN  { Si se cambió de proveedor, registra el cambio. }
        BEGIN
          pnt_ap[loc_artic[cont]]^.clave := clave;
          EscribirCad(loc_artic[cont],8,pnt_ap[loc_artic[cont]]^.clave);
        END;
      IF pnt_ap[loc_artic[cont]]^.precio_ult <> precio THEN     { Si hubo cambio en el precio, lo actualiza. }
        BEGIN
          pnt_ap[loc_artic[cont]]^.precio_ult := precio;
          EscribirReal(loc_artic[cont],10,pnt_ap[loc_artic[cont]]^.precio_ult);
        END;
      IF pnt_ap[loc_artic[cont]]^.fech <> fecha THEN            { Si hubo cambio en la última fecha de }
        BEGIN                                                   { compra, actualiza dicha fecha. }
          pnt_ap[loc_artic[cont]]^.fech := fecha;
          EscribirFecha(loc_artic[cont],11,pnt_ap[loc_artic[cont]]^.fech);
```

```
            END;
          END;
        IF num_ent = 0 THEN                    { Si no habia entradas registradas, el folio de la }
          num_ent_i := num;                    { primera será el de la entrada actual. }
        num_ent := Succ(num_ent);              { Actualiza el número de entradas registradas. }
        loc_ent := num_ent;
        num_ent_f := num;                      { El folio de la última entrada será el de la actual. }
        origen := 'C';                         { Indica que el origen de la entrada es por compra. }
        fech := fecha;                         { La fecha de la entrada será la actual. }
        na := 4;                               { Abre el archivo de registro de entradas. }
        bas[na].nom_arch := nom_arch_ent;
        AbrirArchivo;
        EscribirEnte(loc_ent,1,num);           { Registra la entrada. }
        EscribirCar(loc_ent,2,origen);
        EscribirByte(loc_ent,3,razon);
        EscribirCad(loc_ent,4,clave);
        EscribirEnte(loc_ent,5,pedido);
        EscribirFecha(loc_ent,6,fech);
        EscribirFecha(loc_ent,7,fech_pag);
        EscribirFecha(loc_ent,8,fech_pt);
        EscribirReal(loc_ent,9,importe);
        EscribirByte(loc_ent,10,n_p);
        CerrarArchivo;                         { Cierra el archivo de registro de entradas. }
        bas[na].nom_arch := nom_arch_ent_p;    { Abre el archivo de registro de partidas. }
        AbrirArchivo;
        ent_part[loc_ent] := bas[na].num_regs + 1;   { Calcula la localización primera partida a agregar. }
        FOR cont := 1 TO n_p DO                { Registra las partidas. }
          WITH part[cont] DO
            BEGIN
              loc := bas[na].num_regs + 1;
              EscribirEnte(loc,1,num);
              EscribirCad(loc,2,artic);
              EscribirEnte(loc,3,cant);
              EscribirReal(loc,4,precio);
            END;
        CerrarArchivo;                         { Cierra el archivo de partidas. }
      END;
  UNTIL opcion[1] = ESC;                       { Lo repite hasta que se presiona la tecla de escape. }
END;
```

EP.INC (Fichero que contiene el código fuente que se muestra.)

```
PROCEDURE Produccion;                                   { Registro de entradas por concepto de producción. }

  LABEL
    Principio;                                          { Para regresar al principio del procedimiento. }

  VAR
    ok,                                                 { Indica que está correcto. }
    otra_part: Boolean;                                 { Indica que se va a introducir otra partida. }
    cont,                                               { Contador. }
    cont_part,                                          { Contador de partida. }
    num_part,                                           { Contador de partida que se introduce. }
    loc:       Integer;                                 { Localización de algún registro. }
    suma:      Real;                                    { Suma de los importes de las partidas. }
    opcion: TTecla;                                     { Opción seleccionada. }
    producto:  Array[1..MAX_N_P] OF Boolean;            { Indicador de producto terminado por partida. }
    loc_artic: Array[1..MAX_N_P] OF Integer;            { Localización del artículo para cada partida. }
    loc_prod:  Array[1..MAX_N_P] OF Integer;            { Localización de la descripción para producción p.p. }
    part_usad: Array[1..MAX_N_P] OF TPartUsad;          { Partes usadas para la producción de cada artículo. }

  FUNCTION OkSubensamble                { Insertar dicha función. }

  FUNCTION OkEsProduccion               { Insertar dicha función. }

  FUNCTION OkPartesProduccion           { Insertar dicha función. }

  BEGIN
    REPEAT
      Principio:
      ClrScr;                                           { Presenta el encabezado. }
      GotoXY(21,1); Write('ENTRADAS/NUEVAS-> P R O D U C C I O N');
      GotoXY(70,1); Write(fecha);
      FillChar(entrada,SizeOf(entrada),0);              { Borra totalmente la variable 'entrada'. }
      WITH entrada DO
        BEGIN
          GotoXY(1,2); ClrEol; Write('Entrada: ');
          IF num_ent <> 0
            THEN                                        { Si hay entradas registradas: }
              IF num_ent = MAX_ENT
                THEN                                    { Si ya no se aceptan más entradas, termina la }
                  BEGIN                                 { ejecución. }
                    MensajeErr('No se pueden almacenar más de ' + JusEnte(MAX_ENT,6) + ' entradas.');
                    Exit;
                  END
                ELSE num := SigFolio(num_ent_f)         { Si se puede aceptar la entrada, asigna su folio. }
```

```
    ELSE                                    { Si no hay entradas registradas, pide que se especifique el }
      BEGIN                                 { folio (distinto de cero) de la primera entrada. }
        MensajeErr('No se tienen entradas registradas. Hay que especificar la primera.');
        REPEAT
          num := 1;
          GotoXY(10,2); PregEnte(num,5,FALSE);
        UNTIL num > 0;
      END;
  GotoXY(10,2); Write(JusFolio(num),' ');       { Presenta en pantalla el folio de la entrada. }
  GotoXY(20,2); clave := EMPRESA; Write('Clave: ');   { Propone la clave de la empresa. }
  PregCadMay(clave,TAM_CLAVE);
  IF clave <> EMPRESA THEN Exit;                { Si se modificada la clave, termina la ejecución. }
  GotoXY(1,4);                                  { Presenta la pantalla de entrada de partidas. }
  WriteLn('                        ');
  WriteLn('| #P | ARTIC | CANT |  ');
  WriteLn('|----|--------|------|  ');
  WriteLn('|  1 |        |      |  ');
  WriteLn('|  2 |        |      |  ');
  WriteLn('|  3 |        |      |  ');
  WriteLn('|  4 |        |      |  ');
  WriteLn('|  5 |        |      |  ');
  WriteLn('|  6 |        |      |  ');
  WriteLn('|  7 |        |      |  ');
  WriteLn('|  8 |        |      |  ');
  WriteLn('|  9 |        |      |  ');
  WriteLn('| 10 |        |      |  ');
  WriteLn('|____|________|______|  ');
  num_part := 0;
  REPEAT
    num_part := Succ(num_part);                 { Pasa a la partida siguiente. }
    GotoXY(6,6 + num_part);                     { Coloca el cursor en la partida correspondiente. }
    Write('|    |        |      |');
    WITH part[num_part] DO
      BEGIN
        ok := FALSE;                            { Por defecto no está correcto. }
        REPEAT
          otra_part := TRUE;                    { Por defecto se espera otra partida. }
          GotoXY(8,6 + num_part);
          PregCadMay(artic, TAM_ARTIC);         { Pide la clave del articulo. }
          IF artic = ''
            THEN otra_part := FALSE             { Si no se introdujo clave, no pasará a otra partida. }
            ELSE                                { Si se introdujo alguna clave: }
              BEGIN
                IF artic[1] = CLA_SUBE
                  THEN                          { Si contiene el carácter para subensambles, indica }
                    BEGIN                       { que no es un producto terminado y verifica que esté }
                      producto[num_part] := FALSE;      { registrada la clave del subensamble. }
                      ok := OkSubensamble(artic,loc_artic[num_part]);
                      IF NOT(ok) THEN MensajeErr('No está registrado este subensamble.');
                    END
                  ELSE                          { Si no contiene el carácter para subensambles, indica }
                    BEGIN                       { que es un producto terminado y verifica que esté }
```

```
                producto[num_part] := TRUE;        { registrada la clave del producto terminado. }
                ok := OkProducto(artic,loc_artic[num_part]);
                IF NOT(ok) THEN MensajeErr('No está registrado este producto terminado.');
              END;
            IF ok THEN                             { Si todo va bien, verifica y extrae las partes para }
              ok := OkPartesProduccion(artic,loc_prod[num_part],part_usad[num_part]);      { producción. }
          END;
      UNTIL ok OR NOT(otra_part);          { Se repite hasta que esté correcto y no se deseen más partidas. }
      ok := FALSE;                         { Por defecto no está correcto. }
      IF otra_part THEN                    { Si se quiere otra partida: }
        BEGIN
          REPEAT
            IF producto[num_part]          { Propone la cantidad pedida. }
              THEN cant := pnt_pt[loc_artic[num_part]]^.ca_ped
              ELSE cant := pnt_ep[loc_artic[num_part]]^.ca_ped;
            GotoXY(17,6 + num_part);
            PregEnte(cant,6,FALSE);                { Pide la cantidad de unidades. }
            IF producto[num_part]                  { Aparece un aviso si se recibe más de lo enviado. }
              THEN
                IF cant > pnt_pt[loc_artic[num_part]]^.ca_ped
                  THEN MensajeErr('Nota: Se recibe más de lo enviado.')
                  ELSE
              ELSE
                IF cant > pnt_ep[loc_artic[num_part]]^.ca_ped THEN
                  MensajeErr('Nota: Se recibe más de lo enviado.');
            UNTIL cant > 0;                        { Lo repite hasta que la cantidad es mayor que cero. }
          END;
        IF NOT(otra_part) OR                       { Si ya no se quiere o puede añadir otra partida: }
          (num_part = MAX_N_P) THEN
          BEGIN
            IF NOT(otra_part) THEN                 { Si ya no se quiere otra partida, ajusta el número de }
              num_part := Pred(num_part);          { partidas. }
            GotoXY(29,23); Write('¿Hacer cambios? (S/N) ');
            PregOpcion(opcion,'SN','');            { Pide que se tome una opción. }
            GotoXY(29,23); ClrEol;
            IF opcion[1] = 'N'
              THEN ok := TRUE                      { Si fue "N", indicará que todo está correcto. }
              ELSE num_part := 0;                  { Si fue "S", pasa a la primera partida. }
          END;
      END;
  UNTIL ok;                                        { Lo repite hasta que todo esté correcto. }
  IF num_part = 0 THEN Goto Principio;             { Si no se introdujeron partidas pasa a 'Principio'. }
  n_p := num_part;
  GotoXY(29,23); Write('¿Todo correcto? (S/N) ');
  PregOpcion(opcion,'SN','');                      { Pide que se tome una opción. }
  GotoXY(29,23); ClrEol;
  IF opcion[1] = 'N' THEN Goto Principio;          { Si fue "N", pasa a 'Principio'. }
  FOR cont := 1 TO n_p DO                          { Contador desde uno hasta el número de partidas. }
    WITH part[cont] DO
      BEGIN
        suma := 0;
        FOR cont_part := 1 TO part_usad[cont,0].cant DO  { Calcula el último costo del artículo. }
```

```
        suma := suma + pnt_ap[part_usad[cont,cont_part].numero]^.precio_ult;
      precio := suma;
      IF producto[cont]
        THEN                                    { Si es un producto terminado: }
          BEGIN
            na := 2;                            { Actualiza las existencias y la cantidad disponible. }
            pnt_pt[loc_artic[cont]]^.exist := pnt_pt[loc_artic[cont]]^.exist + cant;
            EscribirEnte(loc_artic[cont],2,pnt_pt[loc_artic[cont]]^.exist);
            pnt_pt[loc_artic[cont]]^.ca_dis := pnt_pt[loc_artic[cont]]^.ca_dis + cant;
            EscribirEnte(loc_artic[cont],3,pnt_pt[loc_artic[cont]]^.ca_dis);
            pnt_pt[loc_artic[cont]]^.ca_ped := pnt_pt[loc_artic[cont]]^.ca_ped - cant;
            IF pnt_pt[loc_artic[cont]]^.ca_ped < 0 THEN     { No acepta una cantidad pedida negativa. }
              pnt_pt[loc_artic[cont]]^.ca_ped := 0;         { Actualiza la cantidad pedida. }
            EscribirEnte(loc_artic[cont],4,pnt_pt[loc_artic[cont]]^.ca_ped);
            IF pnt_pt[loc_artic[cont]]^.precio_ult <> suma THEN  { Si hubo cambio en el costo del articulo, }
              BEGIN                                     { lo actualiza. }
                pnt_pt[loc_artic[cont]]^.precio_ult := suma;
                EscribirReal(loc_artic[cont],10,pnt_pt[loc_artic[cont]]^.precio_ult);
              END;
            IF pnt_pt[loc_artic[cont]]^.fech <> fecha THEN   { Si hubo cambio en la última fecha de }
              BEGIN                                     { producción, la actualiza. }
                pnt_pt[loc_artic[cont]]^.fech := fecha;
                EscribirFecha(loc_artic[cont],11,pnt_pt[loc_artic[cont]]^.fech);
              END;
          END
        ELSE                                    { Si es un subensamble: }
          BEGIN
            na := 1;                            { Actualiza las existencias del artículo. }
            pnt_ap[loc_artic[cont]]^.exist := pnt_ap[loc_artic[cont]]^.exist + cant;
            EscribirEnte(loc_artic[cont],2,pnt_ap[loc_artic[cont]]^.exist);
            pnt_ap[loc_artic[cont]]^.ca_ped := pnt_ap[loc_artic[cont]]^.ca_ped - cant;
            IF pnt_ap[loc_artic[cont]]^.ca_ped < 0 THEN     { No acepta una cantidad pedida negativa. }
              pnt_ap[loc_artic[cont]]^.ca_ped := 0;         { Actualiza la cantidad pedida. }
            EscribirEnte(loc_artic[cont],3,pnt_ap[loc_artic[cont]]^.ca_ped);
            IF pnt_ap[loc_artic[cont]]^.precio_ult <> suma THEN  { Si hubo cambio en el costo del artículo, }
              BEGIN                                     { lo actualiza. }
                pnt_ap[loc_artic[cont]]^.precio_ult := suma;
                EscribirReal(loc_artic[cont],10,pnt_ap[loc_artic[cont]]^.precio_ult);
              END;
            IF pnt_ap[loc_artic[cont]]^.fech <> fecha THEN   { Si hubo cambio en la última fecha de }
              BEGIN                                     { producción, la actualiza. }
                pnt_ap[loc_artic[cont]]^.fech := fecha;
                EscribirFecha(loc_artic[cont],11,pnt_ap[loc_artic[cont]]^.fech);
              END;
          END;
    END;
  IF num_ent = 0 THEN                       { Si no se tienen entradas registradas, el folio de la }
    num_ent_i := num;                       { primera será el de la entrada actual. }
  num_ent := Succ(num_ent);                 { Actualiza el número de entradas registradas. }
  loc_ent := num_ent;
  num_ent_f := num;                         { El folio de la última entrada será el de la actual. }
  origen := 'P';                            { Indica que el origen de la entrada es producción. }
```

```
        fech := fecha;                          { La fecha de la entrada será la actual. }
        fech_pag := fecha;                      { La fecha para pago será la actual. }
        fech_pt := fecha;                       { La fecha de pago será la actual. }
        na := 4;                                { Abre el archivo de registro de entradas. }
        bas[na].nom_arch := nom_arch_ent;
        AbrirArchivo;
        EscribirEnte(loc_ent,1,num);            { Registra la entrada. }
        EscribirCar(loc_ent,2,origen);
        EscribirByte(loc_ent,3,razon);
        EscribirCad(loc_ent,4,clave);
        EscribirEnte(loc_ent,5,pedido);
        EscribirFecha(loc_ent,6,fech);
        EscribirFecha(loc_ent,7,fech_pag);
        EscribirFecha(loc_ent,8,fech_pt);
        EscribirReal(loc_ent,9,importe);
        EscribirByte(loc_ent,10,n_p);
        CerrarArchivo;                          { Cierra el archivo de registro de entradas. }
        bas[na].nom_arch := nom_arch_ent_p;     { Abre el archivo de registro de partidas. }
        AbrirArchivo;
        ent_part[loc_ent] := bas[na].num_regs + 1;  { Calcula la localización de la primera partida. }
        FOR cont := 1 TO n_p DO                 { Registra las partidas. }
          WITH part[cont] DO
            BEGIN
              loc := bas[na].num_regs + 1;
              EscribirEnte(loc,1,num);
              EscribirCad(loc,2,artic);
              EscribirEnte(loc,3,cant);
              EscribirReal(loc,4,precio);
            END;
        CerrarArchivo;                          { Cierra el archivo de partidas. }
      END;
  UNTIL opcion[1] = ESC;                        { Lo repite hasta que se presiona la tecla de escape. }
END;
```

EM.INC (Fichero que contiene el código fuente que se muestra.)

```
PROCEDURE Maquilas;                                    ( Registro de entradas por concepto de maquila. )

  LABEL
    Principio;                                         ( Para regresar al principio del procedimiento. )

  VAR
    ok,                                                ( Indica que está correcto. )
    otra_part: Boolean;                                ( Indica que se va a introducir otra partida. )
    cont,                                              ( Contador. )
    cont_part,                                         ( Contador de partida. )
    num_part,                                          ( Contador de partida que se introduce. )
    loc_pro,                                           ( Localización del proveedor. )
    loc:       Integer;                                ( Localización de algún registro. )
    tot_part,                                          ( Total de la partida. )
    suma,                                              ( Suma de los importes de las partidas. )
    descuento,                                         ( Importe del descuento. )
    total:     Real;                                   ( Total de la entrada. )
    opcion: TTecla;                                    ( Opción seleccionada. )
    nombre: TNombre;                                   ( Nombre del proveedor. )
    loc_artic: Array[1..MAX_N_P] OF Integer;           ( Localización del artículo para cada partida. )
    loc_maq:   Array[1..MAX_N_P] OF Integer;           ( Localización de la descripción para maquila p.p. )
    part_usad: Array[1..MAX_N_P] OF TPartUsad;         ( Partes usadas para la maquila de cada artículo. )

  FUNCTION OkMaquila                    ( Insertar dicha función. )

  FUNCTION OkDesMaquila                 ( Insertar dicha función. )

  FUNCTION OkPartesMaquila              ( Insertar dicha función. )

  BEGIN
    REPEAT
      Principio:
      ClrScr;                                          ( Presenta el encabezado. )
      GotoXY(23,1); Write('ENTRADAS/NUEVAS-> M A Q U I L A S');
      GotoXY(70,1); Write(fecha);
      FillChar(entrada,SizeOf(entrada),0);             ( Borra totalmente la variable 'entrada'. )
      WITH entrada DO
        BEGIN
          GotoXY(1,2); ClrEol; Write('Entrada: ');
          IF num_ent <> 0
            THEN                                       ( Si hay entradas registradas: )
              IF num_ent = MAX_ENT
                THEN                                   ( Si ya no se aceptan más entradas, termina la )
                  BEGIN                                ( ejecución. )
```

```
            MensajeErr('No se pueden almacenar más de ' + JusEnte(MAX_ENT,6) + ' entradas.');
            Exit;
          END
        ELSE num := SigFolio(num_ent_f)                    { Si se puede aceptar la entrada, asigna su folio. }
      ELSE                                          { Si no hay entradas registradas, pide que se especifique el }
        BEGIN                                       { folio (distinto de cero) de la primera entrada. }
          MensajeErr('No se tienen entradas registradas. Hay que especificar la primera.');
          REPEAT
            num := 1;
            GotoXY(10,2); PregEnte(num,5,FALSE);
          UNTIL num > 0;
        END;
      GotoXY(10,2); Write(JusFolio(num),' ');              { Presenta en pantalla el folio de la entrada. }
      GotoXY(15,2); Write('Clave:');
      REPEAT
        REPEAT
          GotoXY(22,2); PregCadMay(clave,TAM_CLAVE);
          IF clave = '' THEN Exit;
        UNTIL OkProveedor(clave, loc_pro);
        na := 5;
        bas[na].nom_arch := nom_arch_des_pro;
        AbrirArchivo;
        LeerCad(loc_pro,2,nombre);
        CerrarArchivo;
        GotoXY(30,2);  Write('Nombre: ',JusCad(nombre,40));
        GotoXY(1,3);   Write('Su pedido: '); PregEnte(pedido,5,FALSE);
        GotoXY(12,3);  Write(JusFolio(pedido),' ');
        GotoXY(30,3);  Write('A pagar: '); PregReal(importe,14,2,FALSE);
        GotoXY(55,3);  Write('Fecha de pago: '); fech_pag := fecha; PregCad(fech_pag,8);
        GotoXY(29,23); Write('¿Está correcto? (S/N) '); PregOpcion(opcion,'SN','');
        GotoXY(29,23); ClrEol;
      UNTIL opcion[1] = 'S';
      GotoXY(1,4);
      WriteLn('
      WriteLn('
      WriteLn('
      WriteLn('
      WriteLn('
      WriteLn('
      WriteLn('
      WriteLn('
      WriteLn('
      WriteLn('
      WriteLn('
      WriteLn('
      WriteLn('
      WriteLn('
      WriteLn('
      WriteLn('
      num_part := 0;
```

#P	ARTIC	CANT	PRECIO	IMPORTE	PRECIO REAL
1					
2					
3					
4					
5					
6					
7					
8					
9					
10					

		SUMA	
I.V.A.		DESCUENTO	
TOT/IVA		TOTAL	

```
REPEAT
  num_part := Succ(num_part);                    { Pasa a la partida siguiente. }
  GotoXY(6,6 + num_part);                        { Coloca el cursor en la partida correspondiente. }
  Write('|       |       |               |                 |');
  WITH part[num_part] DO
    BEGIN
      ok := FALSE;                               { Por defecto no está correcto. }
      REPEAT
        otra_part := TRUE;                       { Por defecto se espera otra partida. }
        GotoXY(8,6 + num_part);
        PregCadMay(artic, TAM_ARTIC);            { Pide la clave del artículo. }
        IF artic = ''
          THEN otra_part := FALSE                { Si no se introdujo clave, no pasará a otra partida. }
          ELSE                                   { Si se introdujo alguna clave: }
            IF OkMaquila(artic,loc_artic[num_part]) THEN       { Si está registrada la clave y las partes }
              IF OkPartesMaquila(artic,loc_maq[num_part],      { para maquila, estará correcto. Si lo }
                                 part_usad[num_part]) THEN     { maquilaba otro proveedor, lo anuncia. }
                BEGIN
                  ok := TRUE;
                  IF pnt_ep[loc_artic[num_part]]^.clave <> clave THEN
                    MensajeErr('Este artículo lo maquilaba ' + pnt_ep[loc_artic[num_part]]^.clave);
                END;
      UNTIL ok OR NOT(otra_part);                { Se repite hasta que esté correcto y no se deseen más partidas. }
      ok := FALSE;                               { Por defecto no está correcto. }
      IF otra_part THEN                          { Si se quiere otra partida: }
        BEGIN
          REPEAT
            cant := pnt_ep[loc_artic[num_part]]^.ca_ped;       { Propone la cantidad pedida. }
            GotoXY(17,6 + num_part);
            PregEnte(cant,6,FALSE);                           { Pide la cantidad de unidades. }
            IF cant > pnt_ep[loc_artic[num_part]]^.ca_ped THEN { Aparece un aviso si se recibe más de lo }
              MensajeErr('Nota: se recibe más de lo enviado.'); { enviado. }
          UNTIL cant > 0;                        { Lo repite hasta que la cantidad es mayor que cero. }
          precio := pnt_maq[loc_maq[num_part]]^.precio_ult;
          GotoXY(26,6 + num_part);
          PregReal(precio,14,2,FALSE);                        { Pide el precio unitario de la maquila. }
          tot_part := precio * cant;                          { Calcula y presenta el total de la partida. }
          GotoXY(43,6 + num_part); Write(JusReal(tot_part,14,2));
        END;
      IF NOT(otra_part) OR                       { Si ya no se quiere o puede añadir otra partida: }
        (num_part = MAX_N_P) THEN
        BEGIN
          IF NOT(otra_part) THEN                 { Si ya no se quiere otra partida, ajusta el número de }
            num_part := Pred(num_part);          { partidas. }
          GotoXY(29,23); Write('¿Hacer cambios? (S/N) ');
          PregOpcion(opcion,'SN','');            { Pide que se haga una elección. }
          GotoXY(29,23); ClrEol;
          IF opcion[1] = 'N'
            THEN ok := TRUE                      { Si fue "N", indicará que todo está correcto. }
            ELSE num_part := 0;                  { Si fue "S", pasa a la primera partida. }
        END;
    END;
```

```
    UNTIL ok;                                          { Lo repite hasta que todo esté correcto. }
    IF num_part = 0 THEN Goto Principio;               { Si no se introdujeron partidas pasa a 'Principio'. }
    n_p := num_part;
    suma := 0;
    FOR cont := 1 TO n_p DO                            { Calcula la suma de los importes de las partidas. }
      suma := suma + part[cont].precio * part[cont].cant;
    GotoXY(43,18); Write(JusReal(suma,14,2));          { Presenta en pantalla la suma. }
    total := 0;
    REPEAT
      GotoXY(43,20); PregReal(total,14,2,FALSE);       { Pide el total a pagar antes de I.V.A. }
    UNTIL total <= suma;                               { Lo repite si el total es mayor a la suma. }
    descuento := suma - total;                         { Calcula el descuento. }
    GotoXY(43,19); Write(JusReal(descuento,14,2));     { Presenta el descuento. }
    GotoXY(12,19); Write(JusReal(total * IVA,14,2));   { Presenta el importe del I.V.A. }
    GotoXY(12,20); Write(JusReal(total * (1 + IVA),14,2)); { Presenta el total ya con I.V.A. }
    FOR cont := 1 TO n_p DO                            { Calcula el precio real unitario de la maquila. }
      WITH part[cont] DO
        BEGIN
          IF suma <> 0
            THEN precio := precio * total / suma
            ELSE precio := 0;
          GotoXY(60,6 + cont); Write(JusReal(precio,14,2));
        END;
    GotoXY(29,23); Write('¿Todo correcto? (S/N) ');
    PregOpcion(opcion,'SN','');                        { Pide que se tome una opción. }
    GotoXY(29,23); ClrEol;
    IF opcion[1] = 'N' THEN Goto Principio;            { Si fue "N", pasa a 'Principio'. }
    na := 4;                                           { Abre el archivo de descripción de partes de maquila. }
    bas[na].nom_arch := nom_arch_maq;
    AbrirArchivo;
    FOR cont := 1 TO n_p DO                            { Contador desde uno hasta el número de partidas. }
      WITH part[cont] DO
        BEGIN
          na := 1;
          suma := 0;
          FOR cont_part := 1 TO part_usad[cont,0].cant DO { Calcula el último costo del artículo. }
            suma := suma + pnt_ap[part_usad[cont,cont_part].numero]^.precio_ult;
          pnt_ap[loc_artic[cont]]^.exist :=            { Actualiza las existencias del artículo. }
            pnt_ap[loc_artic[cont]]^.exist + cant;
          EscribirEnte(loc_artic[cont],2,pnt_ap[loc_artic[cont]]^.exist);
          pnt_ap[loc_artic[cont]]^.ca_ped := pnt_ap[loc_artic[cont]]^.ca_ped - cant;
          IF pnt_ap[loc_artic[cont]]^.ca_ped < 0 THEN  { No acepta una cantidad pedida negativa. }
            pnt_ap[loc_artic[cont]]^.ca_ped := 0;      { Actualiza la cantidad pedida. }
          EscribirEnte(loc_artic[cont],3,pnt_ap[loc_artic[cont]]^.ca_ped);
          IF pnt_ap[loc_artic[cont]]^.clave <> clave THEN { Si hubo cambio en el proveedor, lo registra. }
            BEGIN
              pnt_ap[loc_artic[cont]]^.clave := clave;
              EscribirCad(loc_artic[cont],8,pnt_ap[loc_artic[cont]]^.clave);
            END;
          IF pnt_ap[loc_artic[cont]]^.precio_ult <> suma + precio THEN { Si el precio de la pieza cambió, }
            BEGIN                                      { se actualiza. }
              pnt_ap[loc_artic[cont]]^.precio_ult := suma + precio;
```

```
            EscribirReal(loc_artic[cont],10,pnt_ap[loc_artic[cont]]^.precio_ult);
          END;
        IF pnt_ap[loc_artic[cont]]^.fech <> fecha THEN  { Si cambió la última fecha de cotización, la }
          BEGIN                                         { actualiza. }
            pnt_ap[loc_artic[cont]]^.fech := fecha;
            EscribirFecha(loc_artic[cont],11,pnt_ap[loc_artic[cont]]^.fech);
          END;
        na := 4;
        IF pnt_maq[loc_maq[cont]]^.precio_ult <> precio THEN      { Si el precio de la maquila cambió, lo }
          BEGIN                                                   { actualiza. }
            pnt_maq[loc_maq[cont]]^.precio_ult := precio;
            EscribirReal(loc_maq[cont],3,pnt_maq[loc_maq[cont]]^.precio_ult);
          END;
        IF pnt_maq[loc_maq[cont]]^.fech <> fecha THEN            { Si cambió la última fecha de cotización de }
          BEGIN                                                 { la maquila, la actualiza. }
            pnt_maq[loc_maq[cont]]^.fech := fecha;
            EscribirFecha(loc_maq[cont],4,pnt_maq[loc_maq[cont]]^.fech);
          END;
      END;
    na := 4;
    CerrarArchivo;                                  { Cierra el archivo de descripciones de maquila. }
    IF num_ent = 0 THEN                             { Si no se tienen entradas registradas el folio de la }
      num_ent_i := num;                             { primera será el de la entrada actual. }
    num_ent := Succ(num_ent);                       { Actualiza el número de entradas registradas. }
    loc_ent := num_ent;
    num_ent_f := num;                               { El folio de la última entrada será el de la actual. }
    origen := 'M';                                  { Indica que el origen de la entrada es maquila. }
    fech := fecha;                                  { La fecha de la entrada será la actual. }
    bas[na].nom_arch := nom_arch_ent;               { Abre el archivo de registro de entradas. }
    AbrirArchivo;
    EscribirEnte(loc_ent,1,num);                    { Registra la entrada. }
    EscribirCar(loc_ent,2,origen);
    EscribirByte(loc_ent,3,razon);
    EscribirCad(loc_ent,4,clave);
    EscribirEnte(loc_ent,5,pedido);
    EscribirFecha(loc_ent,6,fech);
    EscribirFecha(loc_ent,7,fech_pag);
    EscribirFecha(loc_ent,8,fech_pt);
    EscribirReal(loc_ent,9,importe);
    EscribirByte(loc_ent,10,n_p);
    CerrarArchivo;                                  { Cierra el archivo de registro de entradas. }
    bas[na].nom_arch := nom_arch_ent_p;             { Abre el archivo de registro de partidas. }
    AbrirArchivo;
    ent_part[loc_ent] := bas[na].num_regs + 1;      { Calcula la localización de la primera partida. }
    FOR cont := 1 TO n_p DO                         { Registra las partidas. }
      WITH part[cont] DO
        BEGIN
          loc := bas[na].num_regs + 1;
          EscribirEnte(loc,1,num);
          EscribirCad(loc,2,artic);
          EscribirEnte(loc,3,cant);
          EscribirReal(loc,4,precio);
```

```
        END;
      CerrarArchivo;                          { Cierra el archivo de partidas. }
    END;
  UNTIL opcion[1] = ESC;                      { Lo repite hasta que se presiona la tecla de escape. }
END;
```

ED.INC { Fichero que contiene el código fuente que se muestra. }

```
{PROCEDURE Devoluciones;                      { Registro de entradas por concepto de devolución. }

  LABEL
    Principio;                                { Para regresar al principio del procedimiento. }

  VAR
    ok,                                       { Indica que está correcto. }
    otra_part: Boolean;                       { Indica que se va a introducir una partida. }
    desc_porc: Byte;                          { Descuento en porcentaje. }
    cont,                                     { Contador. }
    num_part,                                 { Contador de partidas. }
    loc_dis,                                  { Localización del distribuidor. }
    loc: Integer;                             { Localización de algún registro. }
    tot_part,                                 { Importe de la partida. }
    suma,                                     { Suma de los importes de las partidas. }
    descuento,                                { Importe del descuento. }
    total: Real;                              { Total a pagar. }
    opcion: TTecla;                           { Opción seleccionada. }
    des_dis: TDesDis;                         { Descripción del distribuidor. }
    producto: Array[1..MAX_N_P] OF Boolean;   { Indicador de producto terminado por partida. }
    loc_artic: Array[1..MAX_N_P] OF Integer;  { Localización del artículo por cada partida. }

  {PROCEDURE CargaDistribuidor                { Insertar dicho procedimiento. }

  {FUNCTION CadenaExterna(acceso: Cad4): Cad78;   { Función para introducción de cadenas en un formato }
                                                  { de impresión en base a una clave de acceso. }
  BEGIN
    CadenaExterna := '';                          { Por defecto será la cadena vacía. }
    WITH des_dis DO
      CASE acceso[1] OF                           { Para cada caso del primer carácter de 'acceso' será: }
        'e' : CadenaExterna := JusFolio(num_not_cred);   { El folio de la nota de crédito siguiente. }
        'f' : CadenaExterna := fecha;                    { La fecha actual. }
        'g' : ;
        'h' : ;
        'i' : CadenaExterna := nombre;                   { El nombre del cliente. }
        'j' : CadenaExterna := direc;                    { La dirección. }
        'k' : CadenaExterna := telef;                    { El teléfono. }
        'l' : CadenaExterna := ciudad;                   { El nombre de la ciudad. }
        'm' : CadenaExterna := estado;                   { El nombre del estado. }
```

```
        'n' : CadenaExterna := c_p;                          { El código postal. }
        'o' : CadenaExterna := JusReal(entrada.importe,14,2); { EL importe de la n. c. antes de I.V.A. }
        'p' : CadenaExterna := JusEnte(Round(IVA * 100),2);          { El porcentaje del I.V.A. }
        'q' : ;
        'r' : ;
        's' : CadenaExterna := 'DEVOLUICION DE MERCANCIA';   { La razón de la nota de crédito. }
        't' : IF clave <> 'EVENT' THEN                       { La cadena "YA ABONADA". }
                CadenaExterna := 'Y A   A B O N A D A';
      END;
  END;

{FUNCTION OkImprimirForma                  { Insertar dicha función. }


BEGIN
  REPEAT
    Principio:
    ClrScr;                                  { Presenta el encabezado. }
    GotoXY(19,1); Write('ENTRADAS/NUEVAS-> D E V O L U C I O N E S');
    GotoXY(70,1); Write(fecha);
    IF con_llave THEN                        { Si el sistema tiene llave, se sale del procedimiento. }
      BEGIN
        MensajeErr('Zona no autorizada.');
        Exit;
      END;
    FillChar(entrada,SizeOf(entrada),0);     { Borra totalmente la variable 'entrada'. }
    WITH entrada DO
      BEGIN
        GotoXY(1,2); ClrEol; Write('Entrada: ');
        IF num_ent <> 0
          THEN                               { Si hay entradas registradas: }
            IF num_ent = MAX_ENT
              THEN                           { Si ya no se aceptan más entradas, termina la }
                BEGIN                        { ejecución. }
                  MensajeErr('No se pueden almacenar más de ' + JusEnte(MAX_ENT,6) + ' entradas.');
                  Exit;
                END
              ELSE num := SigFolio(num_ent_f)    { Si se puede aceptar la entrada, asigna su folio. }
          ELSE                               { Si no hay entradas registradas, pide que se especifique el }
            BEGIN                            { folio (distinto de cero) de la primera entrada. }
              MensajeErr('No se tienen entradas registradas. Hay que especificar la primera.');
              REPEAT
                num := 1;
                GotoXY(10,2); PregEnte(num,5,FALSE);
              UNTIL num > 0;
            END;
        GotoXY(10,2); Write(JusFolio(num),' ');    { Presenta en pantalla el folio de la entrada. }
        GotoXY(15,2); Write('Clave:');
        REPEAT
          REPEAT
            GotoXY(22,2); PregCadMay(clave,TAM_CLAVE);       { Pide la clave del distribuidor. }
```

```
    IF clave = '' THEN Exit;                    { Si no se introduce clave, termina la ejecución. }
  UNTIL OkDistribuidor(clave, loc_dis);         { Se repite hasta que la clave esté registrada. }
  na := 5;                                  { Abre el archivo de descripciones de distribuidores. }
  bas[na].nom_arch := nom_arch_des_dis;
  AbrirArchivo;
  CargaDistribuidor(loc_dis,des_dis);       { Lee del disco la descripción del distribuidor. }
  CerrarArchivo;                            { Cierra el archivo de descripciones de distribuidores. }
  pedido := num_devol;                      { Asigna el folio de la devolución. }
  GotoXY(30,2);  Write('Nombre: ',JusCad(des_dis.nombre,40));    { Presenta en pantalla el nombre del }
  GotoXY(1,3);   Write('Devolución: ',JusFolio(pedido));         { distribuidor y el folio de la devolución. }
  GotoXY(29,23); Write('¿Está correcto? (S/N) ');
  PregOpcion(opcion,'SN',' ');              { Pide que se tome una opción. }
  GotoXY(29,23); ClrEol;
UNTIL opcion[1] = 'S';                      { Se repite hasta que se seleccione la opción "S". }
GotoXY(1,4);                                { Presenta la pantalla de entrada de partidas. }
WriteLn('
WriteLn('  #P  |  ARTIC  |  CANT  |  PRECIO  |  IMPORTE  ');
WriteLn('
WriteLn('   1  |        |        |          |           ');
WriteLn('   2  |        |        |          |           ');
WriteLn('   3  |        |        |          |           ');
WriteLn('   4  |        |        |          |           ');
WriteLn('   5  |        |        |          |           ');
WriteLn('   6  |        |        |          |           ');
WriteLn('   7  |        |        |          |           ');
WriteLn('   8  |        |        |          |           ');
WriteLn('   9  |        |        |          |           ');
WriteLn('  10  |        |        |          |           ');
WriteLn('
WriteLn('                                    SUMA      ');
WriteLn('  I.V.A.                          % DESC      ');
WriteLn('  TOT/IVA                         TOTAL     ');
WriteLn('
num_part := 0;
REPEAT
  num_part := Succ(num_part);               { Pasa a la partida siguiente. }
  GotoXY(6,6 + num_part);                   { Coloca el cursor en la partida siguiente. }
  Write('|        |        |          |           |');
  WITH part[num_part] DO
    BEGIN
      ok := FALSE;                          { Por defecto no está correcto. }
      REPEAT
        otra_part := TRUE;                  { Por defecto se espera otra partida. }
        GotoXY(8,6 + num_part); PregCadMay(artic, TAM_ARTIC);
        IF artic = ''
          THEN otra_part := FALSE           { Si no se introdujo clave, no pasará a otra partida. }
          ELSE                              { Si se introdujo alguna clave: }
            BEGIN
              producto[num_part] := OkProducto(artic,loc_artic[num_part]);
              IF producto[num_part]
                THEN ok := TRUE             { Si es un producto terminado, indica que está correcto. }
                ELSE                        { Si no es un producto terminado: }
```

```
                    IF OkMaterial(artic,loc_artic[num_part])
                      THEN ok := TRUE          { Si es un material, indica que está correcto. }
                      ELSE MensajeErr('No está registrado ese articulo.');
                  END;
              UNTIL ok OR NOT(otra_part);      { Se repite hasta que esté correcto y no se deseen más partidas. }
              ok := FALSE;                     { Por defecto no está correcto. }
              IF otra_part THEN                { Si se quiere otra partida. }
                BEGIN
                  REPEAT
                    GotoXY(17,6 + num_part); Write('      ');
                    GotoXY(17,6 + num_part); PregEnte(cant,6,FALSE);
                  UNTIL cant > 0;              { Pide la cantidad hasta que ésta sea mayor a cero. }
                  IF producto[num_part]        { Propone el precio público actual. }
                    THEN precio := pnt_pt[loc_artic[num_part]]^.precio_v
                    ELSE precio := pnt_ep[loc_artic[num_part]]^.precio_v;
                  GotoXY(26,6 + num_part);
                  PregReal(precio,14,2,FALSE);  { Pide el precio de la mercancía. }
                  tot_part := precio * cant;    { Calcula y muestra el importe de la partida. }
                  GotoXY(43,6 + num_part); Write(JusReal(tot_part,14,2));
                END;
              IF NOT(otra_part) OR             { Si ya no se quiere o puede añadir otra partida: }
                 (num_part = MAX_N_P) THEN
                BEGIN
                  GotoXY(1,22); Write('¿Razón de devolución? (código) ');
                  PregByte(razon,3);           { Pide el código de la razón de la cancelación. }
                  IF NOT(otra_part) THEN       { Si ya no se quiere otra partida, ajusta el número de }
                    num_part := Pred(num_part);{ partidas. }
                  GotoXY(29,23); Write('¿Hacer cambios? (S/N) ');
                  PregOpcion(opcion,'SN',#0);  { Pide que se tome una opción. }
                  GotoXY(29,23); ClrEol;
                  IF opcion[1] = 'N'
                    THEN ok := TRUE            { Si fue "N", indicará que todo está correcto. }
                    ELSE num_part := 0;        { Si fue "S", pasa a la primera partida. }
                END;
            END;
          UNTIL ok;                            { Lo repite hasta que todo esté correcto. }
          IF num_part = 0 THEN Goto Principio; { Si no se introdujeron partidas pasa a 'Principio'. }
          n_p := num_part;
          suma := 0;
          FOR cont := 1 TO n_p DO              { Calcula y muestra la suma de los importes de las partidas. }
            suma := suma + part[cont].precio * part[cont].cant;
          GotoXY(43,18); Write(JusReal(suma,14,2));
          desc_porc := des_dis.d;              { Propone el descuento del cliente. }
          GotoXY(31,19); PregByte(desc_porc,2);{ Pide el descuento a asignar. }
          descuento := suma * (desc_porc/100); { Calcula y muestra el importe del descuento. }
          GotoXY(43,19); Write(JusReal(descuento,14,2));
          total := suma - descuento;           { Calcula y muestra el total del pedido antes de impuestos. }
          GotoXY(43,20); Write(JusReal(total,14,2));
          GotoXY(12,19); Write(JusReal(total * IVA,14,2));          { Muestra el importe del I.V.A. }
          GotoXY(12,20); Write(JusReal(total * (1 + IVA),14,2));    { Muestra el total después de impuestos. }
          GotoXY(40,3); importe := total; Write('A pagar: ');
          PregReal(importe,14,2,FALSE);        { Permite que se altere manualmente el importe a pagar. }
```

```
        GotoXY(29,23); Write('¿Todo correcto? (S/N) ');
        PregOpcion(opcion,'SN',#);                  { Pide que se tome una opción. }
        GotoXY(29,23); ClrEol;
        IF opcion[1] = 'N' THEN Goto Principio;     { Si fue "N", termina la ejecución. }
        IF NOT(OkImprimirForma('FORMATOS/F_NC.TXT',1))
          THEN Goto Principio;                      { Si no se imprimió la nota de crédito, pasa a 'Principio'. }
        FOR cont := 1 TO n_p DO                     { Contador desde uno hasta el número de partidas, para }
          WITH part[cont] DO                        { actualizar los inventarios tanto de materiales como de }
            BEGIN                                   { productos terminados. }
              IF producto[cont]
                THEN
                  BEGIN
                    na := 2;
                    pnt_pt[loc_artic[cont]]^.exist := pnt_pt[loc_artic[cont]]^.exist + cant;
                    EscribirEnte(loc_artic[cont],2,pnt_pt[loc_artic[cont]]^.exist);
                    pnt_pt[loc_artic[cont]]^.ca_dis := pnt_pt[loc_artic[cont]]^.ca_dis + cant;
                    EscribirEnte(loc_artic[cont],3,pnt_pt[loc_artic[cont]]^.ca_dis);
                  END
                ELSE
                  BEGIN
                    na := 1;
                    pnt_mp[loc_artic[cont]]^.exist := pnt_mp[loc_artic[cont]]^.exist + cant;
                    EscribirEnte(loc_artic[cont],2,pnt_mp[loc_artic[cont]]^.exist);
                  END;
            END;
        num_devol := SigFolio(num_devol);           { Actualiza en memoria y disco el folio de la devolución }
        na := 3;                                    { siguiente. }
        EscribirEnte(1,2,num_devol);
        IF num_ent = 0                              { Si no se tienen entradas registradas, el folio de la }
          THEN num_ent_i := num;                    { primera será el de la entrada actual. }
        num_ent := Succ(num_ent);                   { Actualiza el número de entradas registradas. }
        loc_ent := num_ent;
        num_ent_f := num;                           { El folio de la última entrada será el de la actual. }
        origen := 'D';                              { Indica que el origen de la entrada es por devolución. }
        fech := fecha;                              { La fecha de la entrada será la actual. }
        fech_pag := fecha;                          { La fecha para pago será la actual. }
        fech_pt := fecha;                           { La fecha de pago total será la actual. }
        IF clave <> 'EVENT' THEN                    { Si el distribuidor no es eventual: }
          BEGIN
            na := 4;                                        { Abre el archivo de descripciones de distribuidores. }
            bas[na].nom_arch := nom_arch_des_dis;
            AbrirArchivo;
            des_dis.saldo := des_dis.saldo + importe;       { Incrementa el saldo del cliente. }
            EscribirReal(loc_dis,12,des_dis.saldo);
            CerrarArchivo;                                  { Cierra el archivo de descripciones de dist. }
            bas[na].nom_arch := nom_arch_cob;               { Abre el archivo de registro de cobros. }
            AbrirArchivo;
            loc := bas[na].num_regs + 1;                    { Agrega el cobro hecho por nota de crédito ya }
            EscribirCad(loc,1,clave);                       { abonada. }
            EscribirFecha(loc,2,fecha);
            EscribirCad(loc,3,'NC ' + JusFolio(num_not_cred));
            EscribirReal(loc,4,importe);
```

```
            CerrarArchivo;                          { Cierra el archivo de registro de cobros. }
          END;
        na := 3;                                    { Actualiza en memoria y disco el folio de la nota de }
        num_not_cred := SigFolio(num_not_cred);     { crédito siguiente. }
        EscribirEnte(3,2,num_not_cred);
        na := 4;                                    { Abre el archivo de registro de entradas. }
        bas[na].nom_arch := nom_arch_ent;
        AbrirArchivo;
        EscribirEnte(loc_ent,1,num);                { Registra la entrada. }
        EscribirCar(loc_ent,2,origen);
        EscribirByte(loc_ent,3,razon);
        EscribirCad(loc_ent,4,clave);
        EscribirEnte(loc_ent,5,pedido);
        EscribirFecha(loc_ent,6,fech);
        EscribirFecha(loc_ent,7,fech_pag);
        EscribirFecha(loc_ent,8,fech_pt);
        EscribirReal(loc_ent,9,importe);
        EscribirByte(loc_ent,10,n_p);
        CerrarArchivo;                          { Cierra el archivo de registro de entradas. }
        bas[na].nom_arch := nom_arch_ent_p;     { Abre el archivo de registro de partidas. }
        AbrirArchivo;
        ent_part[loc_ent] := bas[na].num_regs + 1;  { Calcula la localización de la primera partida. }
        FOR cont := 1 TO n_p DO                 { Registra las partidas. }
          WITH part[cont] DO
            BEGIN
              loc := bas[na].num_regs + 1;
              EscribirEnte(loc,1,num);
              EscribirCad(loc,2,artic);
              EscribirEnte(loc,3,cant);
              EscribirReal(loc,4,precio);
            END;
        CerrarArchivo;                          { Cierra el archivo de partidas. }
      END;
  UNTIL opcion[1] = ESC;                        { Lo repite hasta que se presiona la tecla de escape. }
END;
```

El.OVL (Fichero que contiene el código fuente que se muestra.)

```
    BEGIN
      REPEAT
        ClrScr;                                    ( Presenta el encabezado del procedimiento. )
        GotoXY(29,1); Write('ENTRADAS-> N U E V A S');
        GotoXY(70,1); Write(fecha);
        Posibilidad(28, 5,'C O M P R A S');        ( Muestra las posibilidades de elección. )
        Posibilidad(28, 7,'P R O D U C C I O N');
        Posibilidad(28, 9,'M A Q U I L A S');
        Posibilidad(29,11,'D E V O L U C I O N E S');
        GotoXY(55,23); Write('¿Qué elección? (Esc) ');
        PregOpcion(opcion,'CPMD' + ESC,'');        ( Pide que se tome una opción. )
        CASE opcion[1] OF                          ( Ejecuta el procedimiento seleccionado. )
          'C' : Compras;
          'P' : Produccion;
          'M' : Maquilas;
          'D' : Devoluciones;
        END;
      UNTIL opcion[1] = ESC;                       ( Se repite hasta que se presione la tecla de escape. )
    END;

{Overlay PROCEDURE YaExistentes;                   ( Procedimiento para el control de las entradas ya existentes. )

  VAR
    opcion: TTecla;                                ( Opción seleccionada. )

  {PROCEDURE CargaEntrada                          ( Insertar dicho procedimiento. )

  {PROCEDURE MostrarEntrada                        ( Insertar dicho procedimiento. )

  {FUNCTION CadenaExterna(acceso: Cad4): Cad78;    ( Función para introducción de cadenas en un formato )
                                                   ( de impresión en base a una clave de acceso. )
    VAR
      cont: Integer;                               ( Contador. )
      suma: Real;                                  ( Suma de los importes de las partidas. )

    BEGIN
      CadenaExterna := '';                         ( Por defecto será la cadena vacía. )
      WITH entrada DO
        CASE acceso[1] OF                          ( Para cada caso del primer carácter de 'acceso' será: )
          'e' : CadenaExterna := JusFolio(num);    ( El folio de la entrada. )
          'f' : CadenaExterna := origen;           ( El origen de la entrada. )
          'g' : CadenaExterna := JusEnte(razon,3); ( El código de la razón de la entrada. )
          'h' : CadenaExterna := clave;            ( Clave del que envió la mercancía. )
          'i' : CadenaExterna := JusFolio(pedido); ( Folio del pedido o referencia. )
          'j' : CadenaExterna := fech;             ( Fecha de la entrada. )
```

```
        'k' : CadenaExterna := fech_pag;                    { Fecha para pago. }
        'l' : CadenaExterna := fech_pt;                     { Fecha en que se asignó el pago total. }
        'm' : CadenaExterna := JusReal(importe,14,2);       { Importe de la entrada. }
        'n' : CadenaExterna := JusEnte(n_p,2);              { Número de partidas de la entrada. }
        'o' : BEGIN                                         { El importe de las partidas. }
                suma := 0;
                FOR cont := 1 TO n_p DO
                  WITH part[cont] DO
                    suma := suma + precio * cant;
                CadenaExterna := JusReal(suma,14,2);
              END;
        'P' : IF Ord(acceso[2]) - 47 <= n_p THEN            { Para la interpretación de las partidas. }
                WITH part[Ord(acceso[2]) - 47] DO           { Identifica el número de partida. }
                  CASE acceso[3] OF                         { Para cada caso del tercer carácter de 'acceso' será: }
                    '1' : CadenaExterna := JusFolio(num);   { El folio de la entrada asociado a la partida. }
                    '2' : CadenaExterna := artic;           { La clave del artículo. }
                    '3' : CadenaExterna := JusEnte(cant,6); { La cantidad. }
                    '4' : CadenaExterna := JusReal(precio,14,2);          { El precio unitario del artículo. }
                    '5' : CadenaExterna := JusReal(precio * cant,14,2);   { El importe de la partida. }
                    ELSE MensajeErr('El formato tiene especificado un campo de partidas no definido.');
                  END;
        ELSE                                                { Si no se pudo interpretar, regresa la cadena de }
          BEGIN                                             { acceso. }
            MensajeErr('El formato tiene especificado un campo no definido.');
            CadenaExterna := '@' + acceso;
          END;
      END;
  END;

{-------------------------------------------------------------------------------}
{FUNCTION OkImprimirForma                 { Insertar dicha función. }
{-------------------------------------------------------------------------------}
{PROCEDURE ImprimirEntrada;               { Procedimiento que imprime el comprobante de la entrada. }
{
  BEGIN
    ok := OkImprimirForma('FORMATOS\F_E.TXT',0);
  END;
{-------------------------------------------------------------------------------}
{PROCEDURE ImprimirEntradaAlmacen;        { Procedimiento que imprime la entrada al almacén. }
{
  BEGIN
    ok := OkImprimirForma('FORMATOS\F_EA.TXT',0);
  END;
{-------------------------------------------------------------------------------}
{PROCEDURE PagoTotal;                     { Procedimiento que asigna el pago total de la entrada. }
{
  VAR
    loc: Integer;                         { Localización de algún registro. }
    saldo: Real;                          { Saldo con el proveedor. }
```

```
  opcion: TTecla;                          { Opción seleccionada. }

BEGIN
  WITH entrada DO
    BEGIN
      IF fech_pt <> '' THEN                { Si ya se asignó el pago, termina la ejecución. }
        BEGIN
          MensajeErr('Ya está pagada esta entrada.');
          Exit;
        END;
      CASE origen OF                       { Para cada origen de la entrada: }

        'C',                               { Para entradas por compras y maquila: }
        'M' : BEGIN
              IF NOT(OkProveedor(clave,loc)) THEN      { Si el proveedor ya no está registrado: }
                BEGIN
                  GotoXY(1,22); Write('Este proveedor ya no está registrado.');
                  GotoXY(25,23); Write('¿Convertirlo en EVENT? (S/N) ');
                  PregOpcion(opcion,'SN',#0);          { Pide que se tome una opción. }
                  GotoXY(1,22); ClrEol;
                  GotoXY(25,23); ClrEol;
                  IF opcion[1] = 'N' THEN Exit;        { Si fue "N", termina la ejecución. }
                  na := 4;
                  clave := 'EVENT';                    { Actualiza la clave del proveedor en memoria y disco. }
                  EscribirCad(loc_ent,4,clave);
                  IF NOT(OkDistribuidor(clave,loc)) THEN  { Si no está regist. la clave, termina la ejecución. }
                    BEGIN
                      MensajeErr('No existe el registro para EVENT. Hay que agregarlo.');
                      Exit;
                    END;
                END;
              na := 5;                             { Abre el archivo de descripciones de proveedores. }
              bas[na].nom_arch := nom_arch_des_pro;
              AbrirArchivo;
              LeerReal(loc,9,saldo);               { Lee del disco el saldo con el proveedor. }
              IF saldo >= importe
                THEN                               { Si el saldo alcanza a cubrir el importe de la }
                  BEGIN                            { entrada: }
                    GotoXY(18,23); Write('¿Seguro de querer registrar el pago? (S/N) ');
                    PregOpcion(opcion,'SN',#0);    { Pide que se tome una opción. }
                    GotoXY(18,23); ClrEol;
                    IF opcion[1] = 'S' THEN        { Si fue "S", actualiza el saldo del proveedor en el }
                      BEGIN                        { disco y registra en el archivo de entradas la fecha }
                        saldo := saldo - importe;  { en que se asignó el pago. }
                        EscribirReal(loc,9,saldo);
                        na := 4;
                        fech_pt := fecha;
                        EscribirFecha(loc_ent,8,fech_pt);
                      END;
                  END
                ELSE MensajeErr('Sólo se tiene ' + JustReal(saldo,14,2) + ' de saldo.');
              na := 5;                             { Se cierra el archivo de desc. de proveedores. }
```

```
                CerrarArchivo;
              END;

        'P' : BEGIN                              { Para entradas por producción, no se requiere el pago. }
                MensajeErr('Las entradas por concepto de producción no se pagan.');
                Exit;
              END;

        'D' : BEGIN
                MensajeErr('Las entradas por concepto de devolución ya están pagadas.');
                Exit;
              END;
      END;
      GotoXY(70,5); Write(JusCad(fech_pt,8));    { Presenta en pantalla la fecha de pago total. }
    END;
  END;
```

```
BEGIN
  IF nua_ent = 0 THEN                            { Si no hay entradas registradas, sale del procedimiento. }
    BEGIN
      MensajeErr('No hay entradas registradas.');
      Exit;
    END;
  na := 4;                                       { Abre el archivo de registro de entradas. }
  bas[na].nom_arch := nom_arch_ent;
  AbrirArchivo;
  REPEAT
    ClrScr;                                      { Presenta el encabezado del procedimiento. }
    GotoXY(22,1); Write('ENTRADAS-> Y A   E X I S T E N T E S');
    GotoXY(70,1); Write(fecha);
    FillChar(entrada,SizeOf(entrada),0);         { Borra completamente la variable 'entrada'. }
    WITH entrada DO
      BEGIN
        nua := nua_ent_i;                        { Propondrá el folio de la primera entrada registrada. }
        REPEAT
          GotoXY(1,2); ClrEol; Write('Entrada: (',JusFolio(nua_ent_i),'..',JusFolio(nua_ent_f),') ');
          PregEnte(nua,5,FALSE);                 { Pide el folio de la entrada a utilizar. }
          GotoXY(23,2); Write(JusFolio(nua),' ');
        UNTIL OkFolio(nua,nua_ent_i,nua_ent_f);  { Se repite hasta que el folio esté correcto. }
        na := 5;                                 { Abre el archivo de registro de entradas por partida. }
        bas[na].nom_arch := nom_arch_ent_p;
        AbrirArchivo;
        CargaEntrada(nua,loc_ent,entrada);       { Lee del disco los datos de la entrada. }
        na := 5;
        CerrarArchivo;                           { Cierra el archivo de registro de entradas por partida. }
        MostrarEntrada;                          { Muestra los datos de la entrada. }
        Posibilidad(64,16,'Comprobante');        { Presenta las diferentes posibilidades de elección. }
        Posibilidad(64,18,'Entrada almacén');
        Posibilidad(64,20,'Pago total');
        REPEAT
          GotoXY(29,23); Write('¿Qué elección? (Mouse) (Esc) ');
```

```
        PregOpcion(opcion,'CEP' + ESC,#71);      { Pide que se tome una opción. }
        GotoXY(29,23); ClrEol;
        CASE opcion[1] OF                        { Para cada caso de la opción seleccionada: }
          'C' : ImprimirEntrada;                 { Imprime el comprobante de la entrada. }
          'E' : ImprimirEntradaAlmacen;          { Imprime la entrada a almacén. }
          'P' : PagoTotal;                       { Asigna el pago total a la entrada. }
        END;
      UNTIL (opcion[1] = ESC) OR                 { Se repite hasta que se apriete la tecla "Esc" o "Home". }
            (opcion[2] = #71);
    END;
  UNTIL opcion[1] = ESC;                         { Se repite hasta que se apriete la tecla de escape. }
  na := 4;                                       { Cierra el archivo de entradas. }
  CerrarArchivo;
END;
```

```
BEGIN
  REPEAT
    ClrScr;                                      { Muestra el encabezado del procedimiento. }
    GotoXY(32,1); Write('E N T R A D A S');
    GotoXY(70,1); Write(fecha);
    Posibilidad(28, 5,'Y A   E X I S T E N T E S');   { Presenta las posibilidades de elección. }
    Posibilidad(28, 7,'N U E V A S');
    GotoXY(55,23); Write('¿Qué elección? <Esc> ');
    PregOpcion(opcion,'YN' + ESC,'');            { Pide que se tome una opción. }
    CASE opcion[1] OF                            { Ejecuta el procedimiento seleccionado. }
      'Y' : YaExistentes;
      'N' : Nuevas;
    END;
  UNTIL opcion[1] = ESC;                         { Se repite hasta que se presiona la tecla de escape. }
END;
```

Este procedimiento se usa para el registro de entradas, para la impresión de las mismas en diferentes formatos y, en su caso, para el control de pagos de dichas entradas.

4.3.9 SUBPROGRAMA DE SALIDAS.

Este subprograma está formado, al igual que el subprograma de entradas, por un procedimiento de solapamiento que a su vez contiene otros dos procedimientos de solapamiento. A continuación se listan los ficheros incluidos que contienen el código fuente del procedimiento de salidas.

SO.OVL (Fichero que contiene el código fuente que se muestra.)

```
!Overlay PROCEDURE Salidas;                    ( Procedimiento de solapamiento para el control de salidas. )
!
!  VAR
!    loc_sal: Integer;                         ( Localización de la salida. )
!    opcion: TTecla;                           ( Opción seleccionada. )
!    salida: TSal;                             ( Datos de la salida. )
!
-------------------------------------------------------------------------------
  !FUNCTION OkMaterial                        ( Insertar dicha función. )
  !
-------------------------------------------------------------------------------
  !FUNCTION OkProducto                        ( Insertar dicha función. )
  !
-------------------------------------------------------------------------------
  !FUNCTION OkProveedor                       ( Insertar dicha función. )
  !
-------------------------------------------------------------------------------
  !FUNCTION OkDistribuidor                    ( Insertar dicha función. )
  !
-------------------------------------------------------------------------------
  !Overlay PROCEDURE Nuevas;                  ( Procedimiento de solapamiento para el registro de nuevas )
  !                                           ( salidas. )
  !  VAR
  !    opcion: TTecla;                        ( Opción seleccionada. )
```

SV.INC (Fichero que contiene el código fuente que se muestra.)

```
PROCEDURE Ventas;                          ( Registro de salidas por concepto de ventas. )

  VAR
    ok: Boolean;                           ( Indica que está correcto. )
    loc_ped,                               ( Localización del pedido. )
    loc_dis: Integer;                      ( Localización del distribuidor. )
    des_dis: TDesDis;                      ( Descripción del distribuidor. )
    pedido: TPed;                          ( Datos del pedido. )
    opcion: TTecla;                        ( Opción seleccionada. )

  PROCEDURE CargaDistribuidor              ( Insertar dicho procedimiento. )

  PROCEDURE CargaPedido                    ( Insertar dicho procedimiento. )

  PROCEDURE MostrarCabezaPedidoExtra       ( Insertar dicho procedimiento. )

  PROCEDURE MostrarPartidas                ( Insertar dicho procedimiento. )

  PROCEDURE MostrarOtros                   ( Insertar dicho procedimiento. )

  FUNCTION OkCambioPrecios                 ( Insertar dicha función. )

  PROCEDURE CambiarPrecios                 ( Insertar dicho procedimiento. )

  PROCEDURE RegSalida;                     ( Procedimiento que lleva a cabo el registro de la salida. )

    VAR
      ok: Boolean;                         ( Indica que todo está correcto. )
      cont,                                ( Contador. )
      loc: Integer;                        ( Localización de algún registro. )
      suma: Real;                          ( Suma de los importes de las partidas. )
      opcion: TTecla;                      ( Opción seleccionada. )
      producto: Array[1..MAX_N_P] OF Boolean;   ( Indicador de producto terminado por partida. )
      loc_artic: Array[1..MAX_N_P] OF Integer;  ( Localización del artículo para cada partida. )

    BEGIN
      FillChar(salida,SizeOf(salida),0);   ( Borra totalmente la variable 'salida'. )
      WITH pedido DO
        BEGIN
          ok := FALSE;                     ( Por defecto no está correcto. )
```

```
      IF cerr = 'S'                          { Si el pedido está abierto y no entregado, estará correcto. }
        THEN MensajeErr('El pedido ya está cerrado. Sólo se puede consultar.')
        ELSE
          IF fech_ent <> ''
            THEN MensajeErr('El pedido ya fue entregado.')
            ELSE ok := TRUE;
    END;
  IF NOT(ok) THEN Exit;                      { Si no está correcto, termina la ejecución. }
  WITH salida DO
    BEGIN
      GotoXY(60,2); ClrEol; Write('Salida: ');
      IF num_sal <> 0
        THEN                                 { Si hay salidas registradas: }
          IF num_sal = MAX_SAL
            THEN                             { Si ya no se aceptan más salidas, termina la ejecución. }
              BEGIN
                MensajeErr('No se pueden almacenar más de ' + JusEnte(MAX_SAL,6) + ' salidas.');
                Exit;
              END
            ELSE num := SigFolio(num_sal_f)  { Si se puede aceptar la salida, asigna su folio. }
        ELSE                                 { Si no hay salidas registradas, pide que se especifique el }
          BEGIN                              { folio (distinto de cero) de la primera salida. }
            MensajeErr('No se tienen salidas registradas. Hay que especificar la primera.');
            REPEAT
              num := 1;
              GotoXY(69,2); PregEnte(num,5,FALSE);
            UNTIL num > 0;
          END;
      GotoXY(68,2); Write(JusFolio(num),' ');  { Presenta en pantalla el folio de la salida. }
    END;
  ok := TRUE;                                { Por defecto está correcto. }
  WITH pedido DO
    BEGIN
      FOR cont := 1 TO n_p DO                { Contador desde uno hasta el número de partidas del pedido. }
        WITH part[cont] DO
          BEGIN
            IF OkProducto(artic,loc_artic[cont])
              THEN                           { Si es un producto terminado: }
                BEGIN
                  producto[cont] := TRUE;    { Indica que es un producto terminado. }
                  IF pnt_pt[loc_artic[cont]]^.exist < cant THEN
                    BEGIN                    { Si no alcanzan las existencias del artículo para }
                      ok := FALSE;           { cubrir la partida, indicará que no está correcto. }
                      MensajeErr('Faltan ' + JusEnte(cant - pnt_pt[loc_artic[cont]]^.exist,6) +
                                 ' ' + artic);
                    END;
                END
              ELSE                           { Si no es un producto terminado: }
                IF OkMaterial(artic,loc_artic[cont])
                  THEN                       { Si es un material: }
                    BEGIN
                      producto[cont] := FALSE;  { Indica que no es un producto terminado. }
```

```
                IF pnt_ap[loc_artic[cont]]^.exist < cant THEN
                  BEGIN                          { Si no alcanzan las existencias del artículo para }
                    ok := FALSE;                 { cubrir la partida, indicará que no está correcto. }
                    MensajeErr('Faltan ' + JusEnte(cant - pnt_ap[loc_artic[cont]]^.exist,6) +
                                 ' ' + artic);
                  END;
              END
            ELSE                                 { Si no es un material, indicará que no está correcto. }
              BEGIN
                ok := FALSE;
                MensajeErr('El artículo ' + artic +
                           ' ya no está registrado. Registrarlo o cancelar pedido.');
              END;
        END;
      IF (pag_ant = 0) AND (pag_tot = 0) THEN  { Si no hay pagos registrados en el pedido: }
        IF OkCambioPrecios THEN                { Si hay cambio en los precios: }
          BEGIN
            GotoXY(1,23); Write('Hay cambio en los precios y no hay anticipo. ¿Actualizarlos? (S/N) ');
            PregOpcion(opcion,'SN',#0);        { Pide que se tome una opción. }
            GotoXY(1,23); ClrEol;
            IF opcion[1] = 'S'
              THEN
                BEGIN                          { Si fue "S": }
                  na := 4;                     { Abre el archivo de registro de pedidos. }
                  bas[na].nom_arch := nom_arch_ped;
                  AbrirArchivo;
                  CambiarPrecios;              { Actualiza los precios. }
                  CerrarArchivo;
                END
              ELSE                             { Si fue "N": }
                IF con_llave
                  THEN                         { Si el sistema tiene llave, indicará que no está correcto. }
                    BEGIN
                      MensajeErr('Se deben de actualizar los precios antes de entregar.');
                      ok := FALSE;
                    END
                  ELSE MensajeErr('Se entregará un pedido sin actualizar precios.');
        END;
      IF fech_pt = '' THEN                     { Si el pedido no ha sido pagado: }
        IF cdp = 0
          THEN                                 { Si el cliente no tiene dias de crédito: }
            IF con_llave
              THEN                             { Si el sistema tiene llave, indicará que no está correcto. }
                BEGIN
                  ok := FALSE;
                  MensajeErr('El pedido no está pagado y el distribuidor no tiene crédito.');
                END
              ELSE MensajeErr('Se entregará un pedido no pagado y el distribuidor no tiene crédito.')
          ELSE                                 { Si el cliente tiene crédito. }
            IF importe - pag_ant > des_dis.l_cred - des_dis.cred THEN     { Si sobrepasa su límite de crédito: }
              IF con_llave
                THEN                           { Si el sistema tiene llave, indicará que no está correcto. }
```

```
                BEGIN
                  ok := FALSE;
                  MensajeErr('El pedido sobrepasa el límite de crédito del distribuidor.');
                END
              ELSE MensajeErr('Se entregará un pedido que sobrepasa el límite de crédito del dist.');
      IF NOT(ok) THEN Exit;                    { Si se encontró algo incorrecto, termina la ejecución. }
      fech_cob := fecha;                       { Por defecto la fecha de cobro será la actual. }
      IF fech_pt = '' THEN                     { Si no está pagado el pedido, pide la fecha de cobro. }
        BEGIN
          MostrarOtros;
          GotoXY(57,11);
          PregCad(fech_cob,8);
        END;
    END;
  GotoXY(28,23); Write('¿Todo correcto? (S/N) ');
  PregOpcion(opcion,'SN','');              { Pide que se tome una opción. }
  IF opcion[1] = 'N' THEN Exit;            { Si fue "N", termina la ejecución. }
  salida.n_p := pedido.n_p;                { Especifica el número de partidas de la salida. }
  suma := 0;
  FOR cont := 1 TO salida.n_p DO           { Inicia un contador desde uno hasta el número de partidas. }
    WITH salida.part[cont] DO
      BEGIN
        artic  := pedido.part[cont].artic; { Pasa las claves y cantidades de los artículos del pedido a }
        cant   := pedido.part[cont].cant;  { la salida. }
        IF producto[cont]                  { Especifica el costo como el precio unitario del artículo y }
          THEN                             { actualiza las existencias, tanto para productos terminados }
            BEGIN                          { como para materiales. }
              precio := pnt_pt[loc_artic[cont]]^.precio_ult;
              na := 2;
              pnt_pt[loc_artic[cont]]^.exist := pnt_pt[loc_artic[cont]]^.exist - cant;
              EscribirEnte(loc_artic[cont],2,pnt_pt[loc_artic[cont]]^.exist);
            END
          ELSE
            BEGIN
              precio := pnt_ep[loc_artic[cont]]^.precio_ult;
              na := 1;
              pnt_ep[loc_artic[cont]]^.exist := pnt_ep[loc_artic[cont]]^.exist - cant;
              EscribirEnte(loc_artic[cont],2,pnt_ep[loc_artic[cont]]^.exist);
            END;
        suma := suma + precio * cant;      { Calcula la suma de los importes de las partidas. }
      END;
  salida.importe := suma;
  IF num_sal = 0 THEN                      { Si no se tienen salidas registradas, el folio de la }
    num_sal_i := salida.num;               { primera será el de la salida actual. }
  num_sal := Succ(num_sal);                { Actualiza el número de salidas registradas. }
  loc_sal := num_sal;
  num_sal_f := salida.num;                 { El folio de la última salida será el de la actual. }
  salida.origen := 'V';                    { Indica que el origen de la entrada es por ventas. }
  salida.clave := pedido.clave;            { La clave de referencia será la clave del distribuidor. }
  salida.pedido := pedido.num;             { El folio de referencia será el folio del pedido. }
  salida.fech := fecha;                    { La fecha de la salida será la actual. }
  WITH pedido DO
```

```
    WITH des_dis DO
      BEGIN
        na := 4;                              { Abre el archivo de registro de pedidos. }
        bas[na].nom_arch := nom_arch_ped;
        AbrirArchivo;
        fech_ent := fecha;                    { Registra la fecha en que se entregó la mercancía. }
        EscribirFecha(loc_ped,7,fech_ent);
        EscribirFecha(loc_ped,19,fech_cob);   { Registra la fecha de cobro. }
        CerrarArchivo;                        { Cierra el archivo de registro de pedidos. }
        IF fech_pt = '' THEN                  { Si el pedido no está pagado: }
          BEGIN
            bas[na].nom_arch := nom_arch_des_dis;     { Abre el archivo de desc. de distribuidores. }
            AbrirArchivo;
            cred := cred + importe - pag_ant;         { Calcula el nuevo importe del crédito. }
            EscribirReal(loc_dis,14,cred);            { Lo registra en el disco. }
            CerrarArchivo;                            { Cierra el archivo de desc. de distribuidores. }
          END;
      END;
    na := 4;                                  { Abre el archivo de registro de salidas. }
    bas[na].nom_arch := nom_arch_sal;
    AbrirArchivo;
    WITH salida DO
      BEGIN                                   { Registra los datos de la salida. }
        EscribirEnte(loc_sal,1,num);
        EscribirCar(loc_sal,2,origen);
        EscribirByte(loc_sal,3,razon);
        EscribirCad(loc_sal,4,clave);
        EscribirEnte(loc_sal,5,pedido);
        EscribirFecha(loc_sal,6,fech);
        EscribirReal(loc_sal,7,importe);
        EscribirByte(loc_sal,8,n_p);
        CerrarArchivo;                        { Cierra el archivo de registro de salidas. }
        bas[na].nom_arch := nom_arch_sal_p;   { Abre el archivo de registro de salidas por partida. }
        AbrirArchivo;
        sal_part[loc_sal] := bas[na].num_regs + 1;     { Calcula la localización de la primera partida. }
        FOR cont := 1 TO n_p DO               { Registra las partidas. }
          WITH part[cont] DO
            BEGIN
              loc := bas[na].num_regs + 1;
              EscribirEnte(loc,1,num);
              EscribirCad(loc,2,artic);
              EscribirEnte(loc,3,cant);
              EscribirReal(loc,4,precio);
            END;
        CerrarArchivo;                        { Cierra el archivo de registro de salidas por partida. }
      END;
    MostrarOtros;                                 { Muestra los datos del estado del pedido. }
  END;
```

```
  BEGIN
    REPEAT
```

```
    ClrScr;                                 { Presenta el encabezado del procedimiento. }
    GotoXY(26,1); Write('SALIDAS/NUEVAS-> V E N T A S');
    GotoXY(70,1); Write(fecha);
    IF num_ped = 0 THEN                     { Si no hay pedidos registrados, termina la ejecución. }
      BEGIN
        MensajeErr('No hay pedidos registrados.');
        Exit;
      END;
    FillChar(pedido,SizeOf(pedido),0);      { Borra totalmente los datos de las variables 'pedido' y }
    FillChar(des_dis,SizeOf(des_dis),0);    { 'des_dis'. }
    WITH pedido DO
      BEGIN
        num := num_ped_i;                   { Propondrá el folio del primer pedido registrado. }
        REPEAT
          GotoXY(1,2); Write('Pedido: (',JusFolio(num_ped_i),'..',JusFolio(num_ped_f),') ');
          PregEnte(num,5,FALSE);            { Pide el folio del pedido a utilizar. }
          GotoXY(22,2); Write(JusFolio(num),' ');
        UNTIL OkFolio(num,num_ped_i,num_ped_f);  { Se repite hasta que el folio esté correcto. }
        na := 4;                            { Abre el archivo de registro de pedidos. }
        bas[na].noa_arch := noa_arch_ped;
        AbrirArchivo;
        na := 5;                            { Abre el archivo de registro de pedidos por partida. }
        bas[na].noa_arch := noa_arch_ped_p;
        AbrirArchivo;
        CargaPedido(num,loc_ped,pedido);    { Lee del disco los datos del pedido. }
        na := 5;
        CerrarArchivo;                      { Cierra el archivo de registro de pedidos por partida. }
        ok := FALSE;                        { Por defecto no está correcto. }
        REPEAT
          IF OkDistribuidor(clave,loc_dis)
            THEN                            { Si la clave del distribuidor está registrada: }
              BEGIN
                na := 5;                    { Abre el archivo de descripciones de distribuidores. }
                bas[na].noa_arch := noa_arch_des_dis;
                AbrirArchivo;
                CargaDistribuidor(loc_dis,des_dis);     { Lee del disco la descripción del distribuidor. }
                CerrarArchivo;              { Cierra el archivo de desc. de distribuidores. }
                ok := TRUE;                 { Indica que está correcto. }
              END
            ELSE                            { Si la clave del distribuidor no está registrada: }
              BEGIN
                GotoXY(1,23); Write('El distribuidor ya no está registrado. ¿Pasarlo a EVENT? (S/N) ');
                PregOpcion(opcion,'SN','');     { Pide que se tome una opción. }
                GotoXY(1,23); ClrEol;
                IF opcion[1] = 'S'
                  THEN                      { Si fue "S": }
                    BEGIN
                      pedido.clave := 'EVENT';  { Cambia la clave en memoria a "EVENT". }
                      na := 4;
                      EscribirCad(loc_ped,8,pedido.clave);  { Registra en disco la nueva clave. }
                    END
                  ELSE                      { Si fue "N", termina la ejecución. }
```

```
                    BEGIN
                      MensajeErr('No se puede entregar la mercancía a un distribuidor no registrado.');
                      Exit;
                    END;
                END;
          UNTIL ok;                                  { Se repite hasta que esté correcto. }
          na := 4;                                   { Cierra el archivo de registro de pedidos. }
          CerrarArchivo;
        END;
      MostrarOtros;                                  { Muestra los datos del estado del pedido. }
      REPEAT
        Posibilidad(37,21,'Salida');                 { Presenta las posibilidades de elección. }
        GotoXY(6,23); ClrEol; Write('¿Qué elección? <F1 Cabecera> <F2 Partidas> <F3 Otros> <Home> <Esc> ');
        PregOpcion(opcion,'S' + ESC,#59#60#61#71);   { Pide que se tome una opción. }
        GotoXY(6,23); ClrEol;
        CASE opcion[2] OF                            { Ejecuta el procedimiento seleccionado. }
          #59 : MostrarCabezaPedidoExtra;            { F1; presenta los datos generales del pedido. }
          #60 : MostrarPartidas;                     { F2; presenta las partidas del pedido. }
          #61 : MostrarOtros;                        { F3; presenta los datos del estado del pedido. }
        END;
        IF opcion[1] = 'S' THEN RegSalida;           { "S"; registra la salida, de ser posible. }
      UNTIL (opcion[2] = #71) OR (opcion[1] = ESC); { Se repite hasta que se presione la tecla "Home" o "Esc". }
    UNTIL opcion[1] = ESC;                           { Se repite hasta que se presione la tecla de escape. }
  END;
```

SP.INC (Fichero que contiene el código fuente que se muestra.)

```
PROCEDURE Produccion;                          ( Registro de salidas por concepto de producción. )

  LABEL
    Principio;                                 ( Para regresar al principio del procedimiento. )

  VAR
    ok,                                        ( Indica que está correcto. )
    otra_part: Boolean;                        ( Indica que se va a introducir otra partida. )
    cont,                                      ( Contador. )
    cont_part,                                 ( Contador de partida. )
    num_part,                                  ( Contador de partida que se introduce. )
    maximo,                                    ( Máximo número de artículos que se pueden producir. )
    limitante,                                 ( Localización del artículo limitante. )
    posibles,                                  ( Cantidad que es posible producir con un solo insumo. )
    loc:       Integer;                        ( Localización de algún registro. )
    tot_part,                                  ( Reporte de la partida. )
    suma:      Real;                           ( Suma de los reportes de las partidas. )
    part_usad: TPartUsad;                      ( Partes usadas por el artículo. )
    opcion:    TTecla;                         ( Opción seleccionada. )
    loc_artic,                                 ( Localización del artículo por partida. )
    loc_prod:  Array[1..MAX_N_P] OF Integer;   ( Localización de la descripción de partes para producción. )
    inv_mp:    Array[1..MAX_M_P] OF Integer;   ( Nuevo inventario de materiales. )

  FUNCTION OkSubensamble                       ( Insertar dicha función. )

  FUNCTION OkDesProduccion                     ( Insertar dicha función. )

  FUNCTION OkPartesProduccion                  ( Insertar dicha función. )

  BEGIN
    REPEAT
      Principio:
      ClrScr;                                  ( Presenta el encabezado del procedimiento. )
      GotoXY(22,1); Write('SALIDAS/NUEVAS-> P R O D U C C I O N');
      GotoXY(70,1); Write(fecha);
      FillChar(salida,SizeOf(salida),0);       ( Borra totalmente la variable 'salida'. )
      WITH salida DO
        BEGIN
          GotoXY(1,2); ClrEol; Write('Salida: ');
          IF num_sal <> 0
            THEN                               ( Si hay salidas registradas. )
              IF num_sal = MAX_SAL
                THEN                           ( Si ya no se aceptan más salidas, termina la ejecución. )
                  BEGIN
```

```
          MensajeErr('No se pueden almacenar más de ' + JusEnte(MAX_SAL,6) + ' salidas.');
          Exit;
        END
      ELSE num := SigFolio(num_sal_f)      { Si se puede aceptar la salida, asigna su folio. }
  ELSE                                     { Si no hay salidas registradas, pide que se especifique el }
    BEGIN                                  { folio (distinto de cero) de la primera salida. }
      MensajeErr('No se tienen salidas registradas. Hay que especificar la primera.');
      REPEAT
        num := 1;
        GotoXY(9,2); PregEnte(num,5,FALSE);
      UNTIL num > 0;
    END;
clave := EMPRESA;                          { Propone la clave de la empresa. }
GotoXY(15,2); Write('Clave: ');
PregCadMay(clave,TAM_CLAVE);               { Pide la clave de la empresa. }
IF clave <> EMPRESA THEN Exit;             { Si se modifica la clave, termina la ejecución. }
GotoXY(10,2); Write(JusFolio(num),' ');    { Presenta el folio de la salida. }
pedido := num_ord_prod;                    { La referencia será el folio de la orden de producción. }
GotoXY(1,3); Write('Orden de producción: ',JusFolio(pedido));
GotoXY(1,4);                               { Presenta la  pantalla de entrada de partidas. }
WriteLn('                                                                    ');
WriteLn('| NP |  ARTIC  | CANT |      PRECIO      |      IMPORTE      |');
WriteLn('|----|---------|------|------------------|-------------------|');
WriteLn('|  1 |         |      |                  |                   |');
WriteLn('|  2 |         |      |                  |                   |');
WriteLn('|  3 |         |      |                  |                   |');
WriteLn('|  4 |         |      |                  |                   |');
WriteLn('|  5 |         |      |                  |                   |');
WriteLn('|  6 |         |      |                  |                   |');
WriteLn('|  7 |         |      |                  |                   |');
WriteLn('|  8 |         |      |                  |                   |');
WriteLn('|  9 |         |      |                  |                   |');
WriteLn('| 10 |         |      |                  |                   |');
WriteLn('|----|---------|------|------------------|-------------------|');
WriteLn('                                  TOTAL |                   |');
WriteLn('                                        |___________________|');
num_part := 0;
REPEAT
  IF num_part = 0 THEN                     { Si todavía no se pasa a la primera partida, pasa a la }
    FOR cont := 1 TO num_ep DO             { variable 'inv_ep' el inventario actual de materiales. }
      inv_ep[cont] := pnt_ep[cont]^.exist;
  num_part := Succ(num_part);              { Pasa a la partida siguiente. }
  GotoXY(6,6 + num_part);                  { Coloca el cursor en la posición correspondiente. }
  Write('|    |         |      |                  |                   |');
  WITH part[num_part] DO
    BEGIN
      ok := FALSE;                         { Por defecto no está correcto. }
      REPEAT
        otra_part := TRUE;                 { Por defecto se espera otra partida. }
        GotoXY(8,6 + num_part);
        PregCadMay(artic, TAM_ARTIC);      { Pide la clave del artículo. }
        IF artic = ''
```

```
        THEN otra_part := FALSE          { Si no se introdujo clave, no pasará a otra partida. }
        ELSE                             { Si se introdujo alguna clave: }
          BEGIN
            IF artic[1] <> CLA_SUBE
              THEN                       { Si no contiene el carácter para subensambles: }
                IF OkProducto(artic,loc_artic[num_part])
                  THEN ok := TRUE        { Si es un producto terminado, indica que está correcto. }
                  ELSE MensajeErr('No está registrado este producto terminado.')
              ELSE                       { Si contiene el carácter para subensambles y la clave está }
                IF OkSubensamble(artic,loc_artic[num_part]) THEN      { registrada, estará correcto. }
                  ok := TRUE;
            IF ok THEN                   { Si estuvo correcto, verifica y obtiene las partes para }
              ok := OkPartesProduccion(artic,loc_prod[num_part],part_usad);     { producción. }
          END;
      IF ok THEN                         { Si estuvo correcto: }
        BEGIN
          maximo := MAXINT;              { Encuentra la cantidad máxima que se puede producir del }
          FOR cont := 1 TO part_usad[0].cant DO   { artículo y el insumo limitante. }
            BEGIN
              posibles := inv_ep[part_usad[cont].numero] DIV part_usad[cont].cant;
              IF posibles < maximo THEN
                BEGIN
                  maximo := posibles;
                  limitante := part_usad[cont].numero;
                END;
            END;
          GotoXY(1,22); ClrEol; Write('Se pueden producir ',JusEnte(maximo,6),
                                      ' limitado por ',pnt_ep[limitante]^.artic);
          IF maximo = 0 THEN ok := FALSE;          { Si no se puede producir, estará incorrecto. }
        END;
    UNTIL ok OR NOT(otra_part);          { Se repite hasta que esté correcto y no se desee otra partida. }
    ok := FALSE;                         { Por defecto no está correcto. }
    IF otra_part THEN                    { Si se quiere otra partida: }
      BEGIN
        REPEAT                                  { Pide la cantidad a enviar. }
          GotoXY(17,6 + num_part); PregEnte(cant,6,FALSE);
        UNTIL (cant > 0) AND (cant <= maximo);    { Se repite hasta que la cantidad sea válida. }
        precio := 0;
        GotoXY(1,22); ClrEol;
        FOR cont := 1 TO part_usad[0].cant DO     { Calcula el nuevo inventario de materiales. }
          inv_ep[part_usad[cont].numero] := inv_ep[part_usad[cont].numero] -
                                            part_usad[cont].cant * cant;
        IF artic[1] = CLA_SUBE                    { El precio unitario será el costo último. }
          THEN precio := pnt_ep[loc_artic[num_part]]^.precio_ult
          ELSE precio := pnt_pt[loc_artic[num_part]]^.precio_ult;
        GotoXY(26,6 + num_part); Write(JusReal(precio,14,2));    { Presenta en pantalla el precio }
        tot_part := precio * cant;                               { unitario y el importe de la partida. }
        GotoXY(43,6 + num_part); Write(JusReal(tot_part,14,2));
      END;
    IF NOT(otra_part) OR                 { Si ya no se quiere o puede añadir otra partida: }
      (num_part = MAX_N_P) THEN
      BEGIN
```

```
            IF NOT(otra_part) THEN          { Si ya no se quiere otra partida, ajusta el número de partidas. }
              num_part := Pred(num_part);
            GotoXY(29,23); Write('¿Hacer cambios? (S/N) ');
            PregOpcion(opcion,'SN',''');    { Pide que se tecee una opción. }
            GotoXY(29,23); ClrEol;
            IF opcion[1] = 'N'
              THEN ok := TRUE               { Si fue "N", indicará que todo está correcto. }
              ELSE num_part := 0;           { Si fue "S", pasa a la primera partida. }
          END;
      END;
    UNTIL ok;                               { Lo repite hasta que todo esté correcto. }
    IF num_part = 0 THEN Goto Principio;    { Si no se introdujeron partidas, pasa a 'Principio'. }
    n_p := num_part;
    importe := 0;
    FOR cont := 1 TO n_p DO                 { Calcula el importe de la salida. }
      importe := importe + part[cont].precio * part[cont].cant;
    GotoXY(43,18); Write(JustReal(importe,14,2));
    GotoXY(29,23); Write('¿Todo correcto? (S/N) ');
    PregOpcion(opcion,'SN',''');            { Pide que se tecee una opción. }
    GotoXY(29,23); ClrEol;
    IF opcion[1] = 'N' THEN Goto Principio; { Si fue "N", pasa a 'Principio'. }
    FOR cont := 1 TO n_p DO                 { Inicia un contador desde uno hasta el número de partidas. }
      WITH part[cont] DO
        BEGIN                               { Actualiza la cantidad pedida para materiales y productos }
          IF artic[1] = CLA_SUBE            { terminados tanto en la memoria como en el disco. }
            THEN
              BEGIN
                na := 1;
                pnt_mp[loc_artic[cont]]^.ca_ped := pnt_mp[loc_artic[cont]]^.ca_ped + cant;
                EscribirEnte(loc_artic[cont],3,pnt_mp[loc_artic[cont]]^.ca_ped);
              END
            ELSE
              BEGIN
                na := 2;
                pnt_pt[loc_artic[cont]]^.ca_ped := pnt_pt[loc_artic[cont]]^.ca_ped + cant;
                EscribirEnte(loc_artic[cont],4,pnt_pt[loc_artic[cont]]^.ca_ped);
              END;
        END;
    na := 1;
    FOR cont := 1 TO num_mp DO              { Actualiza las existencias de materiales en memoria y disco. }
      IF inv_mp[cont] <> pnt_mp[cont]^.exist THEN
        BEGIN
          pnt_mp[cont]^.exist := inv_mp[cont];
          EscribirEnte(cont,2,pnt_mp[cont]^.exist);
        END;
    num_ord_prod := SigFolio(num_ord_prod); { Actualiza en memoria y disco el folio de la orden de }
    na := 3;                                { producción siguiente. }
    EscribirEnte(5,2,num_ord_prod);
    IF num_sal = 0 THEN                     { Si no se tienen salidas registradas, el folio de la }
      num_sal_i := num;                     { primera será el de la salida actual. }
    num_sal := Succ(num_sal);               { Actualiza el número de salidas registradas. }
    loc_sal := num_sal;
```

```
        num_sal_f := num;                    { El folio de la última salida será el de la actual. }
        origen := 'P';                       { Indica que el origen de la salida es producción. }
        fech := fecha;                       { La fecha de la salida será la actual. }
        na := 4;                             { Abre el archivo de registro de salidas. }
        bas[na].nom_arch := nom_arch_sal;
        AbrirArchivo;
        EscribirEnte(loc_sal,1,num);         { Registra la salida. }
        EscribirCar(loc_sal,2,origen);
        EscribirByte(loc_sal,3,razon);
        EscribirCad(loc_sal,4,clave);
        EscribirEnte(loc_sal,5,pedido);
        EscribirFecha(loc_sal,6,fech);
        EscribirReal(loc_sal,7,importe);
        EscribirByte(loc_sal,8,n_p);
        CerrarArchivo;                       { Cierra el archivo de registro de salidas. }
        bas[na].nom_arch := nom_arch_sal_p;  { Abre el archivo de registro de salidas por partida. }
        AbrirArchivo;
        sal_part[loc_sal] := bas[na].num_regs + 1;    { Calcula la localización de la primera partida. }
        FOR cont := 1 TO n_p DO              { Registra las partidas. }
          WITH part[cont] DO
            BEGIN
              loc := bas[na].num_regs + 1;
              EscribirEnte(loc,1,num);
              EscribirCad(loc,2,artic);
              EscribirEnte(loc,3,cant);
              EscribirReal(loc,4,precio);
            END;
        CerrarArchivo;                       { Cierra el archivo de registro de salidas por partida. }
      END;
  UNTIL opcion[1] = ESC;                     { Se repite hasta que se presione la tecla de escape. }
END;
```

SM.INC { Fichero que contiene el código fuente que se muestra. }

```
{PROCEDURE Maquilas;                          { Registro de salidas por concepto de maquilas. }
 }
 { LABEL
 {   Principio;                               { Para regresar al principio del procedimiento. }
 }
 { VAR
 {   ok,                                      { Indica que está correcto. }
 {   otra_part: Boolean;                      { Indica que se va a introducir otra partida. }
 {   cont,                                    { Contador. }
 {   cont_part,                               { Contador de partida. }
 {   num_part,                                { Contador de partida que se introduce. }
 {   maxieo,                                  { Máximo número de artículos que se pueden producir. }
 {   limitante,                               { Localización del artículo limitante. }
 {   posibles,                                { Cantidad que es posible producir con un solo insumo. }
 {   loc_pro,                                 { Localización del proveedor. }
 {   loc: Integer;                            { Localización de algún registro. }
 {   tot_part,                                { Importe de la partida. }
 {   suma: Real;                              { Suma de los importes de las partidas. }
 {   part_usad: TPartUsad;                    { Partes usadas por el artículo. }
 {   nombre: TNombre;                         { Nombre del proveedor. }
 {   opcion: TTecla;                          { Opción seleccionada. }
 {   loc_artic,                               { Localización del artículo por partida. }
 {   loc_maq: Array[1..MAX_M_P] OF Integer;   { Localización de la descripción para maquila por partida. }
 {   inv_mp: Array[1..MAX_M_P] OF Integer;    { Nuevo inventario de materiales. }
 {
 {-------------------------------------------------------------------------------
 {  {FUNCTION OkMaquila                       { Insertar dicha función. }
 {  {
 {-------------------------------------------------------------------------------
 {  {FUNCTION OkDesMaquila                    { Insertar dicha función. }
 {  {
 {-------------------------------------------------------------------------------
 {  {FUNCTION OkPartesMaquila                 { Insertar dicha función. }
 {  {
 {-------------------------------------------------------------------------------
 { BEGIN
 {   REPEAT
 {     Principio:
 {     ClrScr;                                { Presenta el encabezado del procedimiento. }
 {     GotoXY(24,1); Write('SALIDAS/NUEVA-> M A Q U I L A S');
 {     GotoXY(70,1); Write(fecha);
 {     FillChar(salida,SizeOf(salida),0);     { Borra totalmente la variable 'salida'. }
 {     WITH salida DO
 {       BEGIN
 {         GotoXY(1,2); ClrEol; Write('Salida: ');
 {         IF num_sal <> 0
 {           THEN                             { Si hay salidas registradas: }
 {             IF num_sal = MAX_SAL
```

```
        THEN                                { Si ya no se aceptan más salidas, termina la ejecución. }
          BEGIN
            MensajeErr('No se pueden almacenar más de ' + JusEnte(MAX_SAL,6) + ' salidas.');
            Exit;
          END
        ELSE num := SigFolio(num_sal_f)     { Si se puede aceptar la salida, asigna su folio. }
      ELSE                                  { Si no hay salidas registradas, pide que se especifique el }
        BEGIN                               { folio (distinto de cero) de la primera salida. }
          MensajeErr('No se tienen salidas registradas. Hay que especificar la primera.');
          REPEAT
            num := 1;
            GotoXY(9,2); PregEnte(num,5,FALSE);
          UNTIL num > 0;
        END;
      GotoXY(10,2); Write(JusFolio(num),' ');   { Presenta el folio de la salida. }
      GotoXY(15,2); Write('Clave:');
      REPEAT
        REPEAT                              { Pide la clave del proveedor. }
          GotoXY(22,2); PregCadMay(clave,TAM_CLAVE);
          IF clave = '' THEN Exit;          { Si no se introduce clave, termina la ejecución. }
        UNTIL DiProveedor(clave, loc_pro);  { Se repite hasta que la clave esté registrada. }
        na := 5;                            { Abre el archivo de descripciones de proveedores. }
        bas[na].nom_arch := nom_arch_des_pro;
        AbrirArchivo;
        LeerCad(loc_pro,2,noabre);          { Lee del disco el nombre del proveedor. }
        CerrarArchivo;                      { Cierra el archivo de descripciones de proveedores. }
        GotoXY(30,2);  Write('Nombre: ',JusCad(noabre,40));     { Presenta el nombre del proveedor. }
        pedido := num_ord_maq;              { La referencia será el folio de la orden de maquila. }
        GotoXY(1,3);   Write('Orden de maquila: ',JusFolio(pedido));
        GotoXY(29,23); Write('¿Está correcto? (S/N) ');
        PregOpcion(opcion,'SN',' ');        { Pide que se tome una opción. }
        GotoXY(29,23); ClrEol;
      UNTIL opcion[1] = 'S';                { Se repite hasta que la opción tomada sea la "S". }
      GotoXY(1,4);                          { Presenta la pantalla de entrada de partidas. }
      WriteLn('                                                                          ');
      WriteLn('  NP  |  ARTIC  |  CANT  |        PRECIO        |        IMPORTE        ');
      WriteLn('                                                                          ');
      WriteLn('   1  |         |        |                      |                       ');
      WriteLn('   2  |         |        |                      |                       ');
      WriteLn('   3  |         |        |                      |                       ');
      WriteLn('   4  |         |        |                      |                       ');
      WriteLn('   5  |         |        |                      |                       ');
      WriteLn('   6  |         |        |                      |                       ');
      WriteLn('   7  |         |        |                      |                       ');
      WriteLn('   8  |         |        |                      |                       ');
      WriteLn('   9  |         |        |                      |                       ');
      WriteLn('  10  |         |        |                      |                       ');
      WriteLn('                                                                          ');
      WriteLn('                                  TOTAL                                   ');
      WriteLn('                                                                          ');
      num_part := 0;
      REPEAT
```

```
IF num_part = 0 THEN                    { Si todavia no se pasa a la primera partida, pasa a la }
  FOR cont := 1 TO num_ep DO            { variable 'inv_ep', el inventario actual de materiales. }
    inv_ep[cont] := pnt_ep[cont]^.exist;
num_part := Succ(num_part);             { Pasa a la partida siguiente. }
GotoXY(6,6 + num_part);                 { Coloca el cursor en la posición correspondiente. }
Write('|       |       |                 |                 |');
WITH part[num_part] DO
  BEGIN
    ok := FALSE;                        { Por defecto no está correcto. }
    REPEAT
      otra_part := TRUE;                { Por defecto se espera otra partida. }
      GotoXY(8,6 + num_part);
      PregCadMay(artic, TAM_ARTIC);     { Pide la clave del artículo. }
      IF artic = ''
        THEN otra_part := FALSE         { Si no se introdujo clave, no pasará a otra partida. }
        ELSE                            { Si se introdujo alguna clave: }
          IF OkMaquila(artic,loc_artic[num_part]) THEN      { Si es un material de maquila: }
            IF OkPartesMaquila(artic,loc_maq[num_part],part_usad)
              THEN                      { Si está correcta la descripción de partes de maquila, estará }
                BEGIN                   { correcto y si hay cambio en el maquilador lo notificará. }
                  ok := TRUE;
                  IF pnt_ep[loc_artic[num_part]]^.clave <> clave THEN
                    MensajeErr('Este artículo lo maquilaba ' + pnt_ep[loc_artic[num_part]]^.clave);
                END;
       IF ok THEN                       { Si está correcto. }
        BEGIN
          maximo := MAXINT;             { Encuentra la cantidad máxima que se puede producir del }
          FOR cont := 1 TO part_usad[0].cant DO   { artículo y el insumo limitante. }
            BEGIN
              posibles := inv_ep[part_usad[cont].numero] DIV part_usad[cont].cant;
              IF posibles < maximo THEN
                BEGIN
                  maximo := posibles;
                  limitante := part_usad[cont].numero;
                END;
            END;
          GotoXY(1,22); ClrEol; Write('Se pueden producir ',JusEnte(maximo,6),
                                      ' limitado por ',pnt_ep[limitante]^.artic);
          IF maximo = 0 THEN ok := FALSE;       { Si no se puede producir, estará incorrecto. }
        END;
      UNTIL ok OR NOT(otra_part);       { Se repite hasta que esté correcto y no se desee otra partida. }
      ok := FALSE;                      { Por defecto no está correcto. }
      IF otra_part THEN                 { Si se quiere otra partida: }
        BEGIN
          REPEAT                                { Pide la cantidad a enviar. }
            GotoXY(17,6 + num_part); PregEnte(cant,6,FALSE);
          UNTIL (cant > 0) AND (cant <= maximo);    { Se repite hasta que la cantidad sea válida. }
          GotoXY(1,22); ClrEol;
          FOR cont := 1 TO part_usad[0].cant DO     { Calcula el nuevo inventario de materiales. }
            inv_ep[part_usad[cont].numero] := inv_ep[part_usad[cont].numero] -
                                              part_usad[cont].cant * cant;
          precio := pnt_ep[loc_artic[num_part]]^.precio_ult;  { El precio unitario será el costo último. }
```

```
        GotoXY(26,6 + num_part); Write(JusReal(precio,14,2));    { Presenta en pantalla el precio }
        tot_part := precio * cant;                               { unitario y el importe de la partida. }
        GotoXY(43,6 + num_part); Write(JusReal(tot_part,14,2));
      END;
    IF NOT(otra_part) OR                    { Si ya no se quiere o puede añadir otra partida: }
      (num_part = MAX_N_P) THEN
      BEGIN
        IF NOT(otra_part) THEN              { Si ya no se quiere otra partida, ajusta el número de partidas. }
          num_part := Pred(num_part);
        GotoXY(29,23); Write('¿Hacer cambios? (S/N) ');
        PregOpcion(opcion,'SN',' ');        { Pide que se tecle una opción. }
        GotoXY(29,23); ClrEol;
        IF opcion[1] = 'N'
          THEN ok := TRUE                   { Si fue "N", indicará que todo está correcto. }
          ELSE num_part := 0;               { Si fue "S", pasa a la primera partida. }
      END;
  END;
UNTIL ok;                                   { Se repite hasta que todo esté correcto. }
IF num_part = 0 THEN Goto Principio;        { Si no se introdujeron partidas, pasa a 'Principio'. }
n_p := num_part;
importe := 0;
FOR cont := 1 TO n_p DO                     { Calcula el importe de la salida. }
  importe := importe + part[cont].precio * part[cont].cant;
GotoXY(43,18); Write(JusReal(importe,14,2));
GotoXY(29,23); Write('¿Todo correcto? (S/N) ');
PregOpcion(opcion,'SN',' ');                { Pide que se tecle una opción. }
GotoXY(29,23); ClrEol;
IF opcion[1] = 'N' THEN Goto Principio;     { Si fue "N", pasa a 'Principio'. }
na := 1;
FOR cont := 1 TO n_p DO                     { Inicia un contador desde uno hasta el número de partidas para }
  WITH part[cont] DO                        { actualizar la cantidad pedida de materiales tanto en la }
    BEGIN                                   { memoria como en el disco. }
      pnt_ap[loc_artic[cont]]^.ca_ped := pnt_ap[loc_artic[cont]]^.ca_ped + cant;
      EscribirEnte(loc_artic[cont],3,pnt_ap[loc_artic[cont]]^.ca_ped);
    END;
FOR cont := 1 TO num_ap DO                  { Actualiza las existencias de materiales tanto en la memoria }
  IF inv_ap[cont] <> pnt_ap[cont]^.exist THEN       { como en el disco. }
    BEGIN
      pnt_ap[cont]^.exist := inv_ap[cont];
      EscribirEnte(cont,2,pnt_ap[cont]^.exist);
    END;
num_ord_maq := SigFolio(num_ord_maq);       { Actualiza en memoria y disco el folio de la orden de maquila. }
na := 3;
EscribirEnte(4,2,num_ord_maq);
IF num_sal = 0 THEN                         { Si no se tienen salidas registradas, el folio de la primera }
  num_sal_i := num;                         { será el de la salida actual. }
num_sal := Succ(num_sal);                   { Actualiza el número de salidas registradas. }
loc_sal := num_sal;
num_sal_f := num;                           { El folio de la última salida será el de la actual. }
origen := 'M';                              { Indica que el origen de la salida es maquila. }
fech := fecha;                              { La fecha de la salida será la actual. }
na := 4;                                    { Abre el archivo de registro de salidas. }
```

```
        bas[na].nom_arch := nom_arch_sal;
        AbrirArchivo;
        EscribirEnte(loc_sal,1,num);          { Registra la salida. }
        EscribirCar(loc_sal,2,origen);
        EscribirByte(loc_sal,3,razon);
        EscribirCad(loc_sal,4,clave);
        EscribirEnte(loc_sal,5,pedido);
        EscribirFecha(loc_sal,6,fech);
        EscribirReal(loc_sal,7,importe);
        EscribirByte(loc_sal,8,n_p);
        CerrarArchivo;                        { Cierra el archivo de registro de salidas. }
        bas[na].nom_arch := nom_arch_sal_p;   { Abre el archivo de registro de salidas por partida. }
        AbrirArchivo;
        sal_part[loc_sal] := bas[na].num_regs + 1;   { Calcula la localización de la primera partida. }
        FOR cont := 1 TO n_p DO               { Registra las partidas. }
          WITH part[cont] DO
            BEGIN
              loc := bas[na].num_regs + 1;
              EscribirEnte(loc,1,num);
              EscribirCad(loc,2,artic);
              EscribirEnte(loc,3,cant);
              EscribirReal(loc,4,precio);
            END;
        CerrarArchivo;                        { Cierra el archivo de registro de salidas por partida. }
      END;
  UNTIL opcion[1] = ESC;                      { Se repite hasta que se presiona la tecla de escape. }
END;
```

S1.OVL (Fichero que contiene el código fuente que se muestra.)

```
    BEGIN
      REPEAT
        ClrScr;                                  ( Presenta el encabezado del procedimiento. )
        GotoXY(30,1); Write('SALIDAS-> N U E V A S');
        GotoXY(70,1); Write(fecha);
        Posibilidad(28, 5,'V E N T A S');
        Posibilidad(28, 7,'P R O D U C C I O N');
        Posibilidad(28, 9,'M A Q U I L A S');
        GotoXY(55,23); Write('¿Qué elección? (Esc) ');
        PregOpcion(opcion,'VPM' + ESC,'');        ( Pide que se tose una opción. )
        CASE opcion[1] OF                         ( Ejecuta el procedimiento seleccionado. )
          'V' : Ventas;
          'P' : Produccion;
          'M' : Maquilas;
        END;
      UNTIL opcion[1] = ESC;                      ( Se repite hasta que se presione la tecla de escape. )
    END;

  !Overlay PROCEDURE YaExistentes;                ( Procedimiento para el control de las salidas ya existentes. )
  !
  !  VAR
  !    cont_sal: Byte;                            ( Contador de salida. )
  !    loc_pro: Integer;                          ( Localización del proveedor. )
  !    opcion: TTecla;                            ( Opción seleccionada. )
  !    salida_aux: TSal;                          ( Salida memorizada. )
  !    des_pro: TDesPro;                          ( Descripción del proveedor. )
  !

    !PROCEDURE CargaSalida                ( Insertar dicho procedimiento. )
    !

    !PROCEDURE CargaProveedor             ( Insertar dicho procedimiento. )
    !

    !PROCEDURE MostrarSalida              ( Insertar dicho procedimiento. )
    !

    !FUNCTION CadenaExterna(acceso: Cad4): Cad78;      ( Función para introducción de cadenas en un formato )
    !                                                  ( de impresión en base a una clave de acceso. )
    !  VAR
    !    cont: Integer;                               ( Contador. )
    !    suma: Real;                                  ( Suma de los importes de las partidas. )
    !
    !  BEGIN
    !    CadenaExterna := '';                         ( Por defecto será la cadena vacía. )
    !    WITH salida DO
    !    WITH des_pro DO
    !      CASE acceso[1] OF                          ( Para cada caso del primer carácter de 'acceso' será: )
```

```
        'e' : CadenaExterna := JusFolio(num);              { El folio de la salida. }
        'f' : CadenaExterna := origen;                     { El origen de la salida. }
        'g' : CadenaExterna := JusEnte(razon,3);           { El código de la salida. }
        'h' : CadenaExterna := salida.clave;               { Clave del que recibió la mercancía. }
        'i' : CadenaExterna := JusFolio(pedido);           { Folio del pedido o referencia. }
        'j' : CadenaExterna := fech;                       { Fecha de la salida. }
        'k' : CadenaExterna := JusReal(importe,14,2);      { Importe de la salida. }
        'l' : CadenaExterna := JusEnte(n_p,2);             { Número de partidas de la salida. }
        'm' : BEGIN                                        { El importe de las partidas. }
                suma := 0;
                FOR cont := 1 TO n_p DO
                  WITH part[cont] DO
                    suma := suma + precio * cant;
                CadenaExterna := JusReal(suma,14,2);
              END;
        'n' : CadenaExterna := nombre;                     { Nombre del proveedor. }
        'o' : CadenaExterna := direc;                      { Dirección del proveedor. }
        'p' : CadenaExterna := telef;                      { Teléfono del proveedor. }
        'q' : CadenaExterna := ciudad;                     { Nombre de la ciudad. }
        'r' : CadenaExterna := estado;                     { Nombre del estado. }
        's' : CadenaExterna := c_p;                        { Número de partidas. }
        't' : IF cont_sal > 0 THEN                         { Cuando se hacen varias salidas a maquila o }
                CadenaExterna := '(' + Chr(96 + cont_sal) + ')';      { producción, será una letra de a..z. }
        'P' : IF Ord(acceso[2]) - 47 <= n_p THEN           { Para la interpretación de las partidas. }
                WITH part[Ord(acceso[2]) - 47] DO          { Identifica el número de partida. }
                  CASE acceso[3] OF                        { Para cada caso del tercer carácter de 'acceso' será: }
                    '1' : CadenaExterna := JusFolio(num);  { El folio de la salida asociado a la partida. }
                    '2' : CadenaExterna := artic;          { La clave del artículo. }
                    '3' : CadenaExterna := JusEnte(cant,6);          { La cantidad. }
                    '4' : CadenaExterna := JusReal(precio,14,2);     { El precio unitario del artículo. }
                    '5' : CadenaExterna := JusReal(precio * cant,14,2); { El importe de la partida. }
                    ELSE MensajeErr('El formato tiene especificado un campo de partidas no definido.');
                  END;
        ELSE                                               { Si no se pudo interpretar, regresa la cadena de }
          BEGIN                                            { acceso. }
            MensajeErr('El formato tiene especificado un campo no definido.');
            CadenaExterna := '@' + acceso;
          END;
      END;
  END;
```

```
FUNCTION OkImprimirForma                { Insertar dicha función. }
```

```
PROCEDURE ImprimirSalida;                { Procedimiento que imprime el comprobante de la salida. }

  BEGIN
    ok := OkImprimirForma('FORMATOS\F_S.TXT',0);
  END;
```

```
{PROCEDURE ImprimirSalidaAlmacen;                    { Procedimiento que imprime la salida del almacén. }
{
{  BEGIN
{    ok := OkImprimirForma('FORMATOS\F_SA.TXT',0);
{  END;
{
-------------------------------------------------------------------------------------------------------
{PROCEDURE SalidaAlmacenProduccion;                  { Procedimiento que imprime la salida del almacén para }
{                                                    { producción. }
{  VAR
{    ok: Boolean;                                    { Indica que está correcto. }
{    cont,                                           { Contador. }
{    cont_part,                                      { Contador de partida. }
{    loc_artic,                                      { Localización del artículo. }
{    loc_prod: Integer;                              { Localización de la descripción de partes para producción. }
{    part_usad: TPartUsad;                           { Partes usadas por el artículo. }
{    part_nece: Array[1..MAX_M_P] OF Integer;        { Acumulador de partes necesarias para la orden de producción. }
{
-------------------------------------------------------------------------------------------------------
   {FUNCTION OkSubensamble                 { Insertar dicha función. }
   {
   -------------------------------------------------------------------------------------------------------
   {FUNCTION OkDesProduccion               { Insertar dicha función. }
   {
   -------------------------------------------------------------------------------------------------------
   {FUNCTION OkPartesProduccion            { Insertar dicha función. }
   {
-------------------------------------------------------------------------------------------------------
{  BEGIN
{    FillChar(part_nece,SizeOf(part_nece),0);        { Borra totalmente la variable 'part_nece'. }
{    salida_nee := salida;                           { Memoriza los datos originales de la salida. }
{    WITH salida DO
{      BEGIN
{        FOR cont := 1 TO n_p DO                     { Inicia un contador desde uno hasta el número de partidas. }
{          WITH part[cont] DO
{            BEGIN
{              ok := FALSE;                        { Por defecto no está correcto. }
{              IF artic[1] = CLA_SUBE
{                THEN                              { Si contiene la clave para subensambles: }
{                  IF OkSubensamble(artic,loc_artic)
{                    THEN ok := TRUE               { Si está registrado el subensamble, estará correcto. }
{                    ELSE
{                ELSE                              { Si no contiene la clave para subensambles: }
{                  IF OkProducto(artic,loc_artic)
{                    THEN ok := TRUE               { Si está registrado el producto terminado, estará correcto. }
{                    ELSE MensajeErr('No está registrado el producto ' + artic);
{              IF ok THEN                          { Si está correcto: }
{                IF OkPartesProduccion(artic,loc_prod,part_usad) THEN   { Si está registrada la descripción para }
{                  FOR cont_part := 1 TO part_usad[0].cant DO           { producción, acumula las partes necesarias. }
{                    part_nece[part_usad[cont_part].numero] := part_nece[part_usad[cont_part].numero] +
{                                                              part_usad[cont_part].cant * cant;
{            END;
```

```
      cont_part := 0;
      cont := 1;
      REPEAT
        WHILE (part_nece[cont] = 0) AND (cont <= num_ep) DO     { Localiza el material utilizado siguiente. }
          cont := Succ(cont);
        IF cont <= num_ep THEN                  { Si se localizó algún material: }
          BEGIN
            cont_part := Succ(cont_part);       { Pasa a la partida siguiente. }
            WITH part[cont_part] DO
              BEGIN                             { Coloca los datos en la variable 'salida'. }
                artic := pnt_ep[cont]^.artic;
                cant := part_nece[cont];
                precio := pnt_ep[cont]^.precio_ult;
                cont := Succ(cont);
              END;
          END;
        IF (cont_part = MAX_N_P) OR             { Si ya no se pueden agregar más partidas a la salida }
           ((cont > num_ep) AND (cont_part > 0)) THEN   { o, ya no se requieren más partidas y la salida }
          BEGIN                                 { contiene al menos una de ellas: }
            n_p := cont_part;                   { Establece el número de partidas de la salida. }
            ImprimirSalidaAlmacen;              { Imprime la salida del almacén. }
            cont_sal := Succ(cont_sal);         { Avanza el contador de salida. }
            cont_part := 0;                     { Reinicializa el contador de partidas a cero. }
          END;
      UNTIL cont > num_ep;                      { Lo repite hasta que el contador rebasa el núm. de materiales. }
    END;
  salida := salida_ees;                         { Restaura los datos originales de la variable 'salida'. }
  cont_sal := 0;
END;

{-------------------------------------------------------------------------------}
{PROCEDURE SalidaAlmacenMaquila;                { Procedimiento que imprime la salida del almacén para maquila. }

  VAR
    cont,                                       { Contador. }
    cont_part,                                  { Contador de partida. }
    loc_artic,                                  { Localización del artículo. }
    loc_eaq: Integer;                           { Localización de la descripción de partes para maquila. }
    part_usad: TPartUsad;                       { Partes usadas por el artículo. }
    part_nece: Array[1..MAX_N_P] OF Integer;    { Acumulador de partes necesarias para la orden de maquila. }

{-------------------------------------------------------------------------------}
  {FUNCTION OkMaquila                   { Insertar dicha función. }

{-------------------------------------------------------------------------------}
  {FUNCTION OkDesMaquila                { Insertar dicha función. }

{-------------------------------------------------------------------------------}
  {FUNCTION OkPartesMaquila             { Insertar dicha función. }

{-------------------------------------------------------------------------------}
  BEGIN
```

```
  FillChar(part_nece,SizeOf(part_nece),0);        { Borra totalmente la variable 'part_nece'. }
  salida_mem := salida;                           { Memoriza los datos originales de la salida. }
  WITH salida DO
    BEGIN
      FOR cont := 1 TO n_p DO                     { Inicia un contador desde uno hasta el número de partidas. }
        WITH part[cont] DO
          BEGIN
            IF OkMaquila(artic,loc_artic) THEN    { Si está registrada la clave del producto de maquila: }
              IF OkPartesMaquila(artic,loc_maq,part_usad) THEN   { Si está registrada la descripción para }
                FOR cont_part := 1 TO part_usad[0].cant DO       { producción, acumula las partes necesarias. }
                  part_nece[part_usad[cont_part].numero] := part_nece[part_usad[cont_part].numero] +
                                                            part_usad[cont_part].cant * cant;
          END;
      cont_part := 0;
      cont := 1;
      REPEAT
        WHILE (part_nece[cont] = 0) AND (cont <= num_ap) DO          { Localiza el material utilizado siguiente. }
          cont := Succ(cont);
        IF cont <= num_ap THEN                    { Si se localizó algún material: }
          BEGIN
            cont_part := Succ(cont_part);         { Pasa a la partida siguiente. }
            WITH part[cont_part] DO
              BEGIN                               { Coloca los datos en la variable 'salida'. }
                artic := pnt_ap[cont]^.artic;
                cant := part_nece[cont];
                precio := pnt_ap[cont]^.precio_ult;
                cont := Succ(cont);
              END;
          END;
        IF (cont_part = MAX_N_P) OR                       { Si ya no se pueden egregar más partidas a la salida }
           ((cont > num_ap) AND (cont_part > 0)) THEN     { o, ya no se requieren más partidas y la salida }
          BEGIN                                           { contiene al menos una de ellas: }
            n_p := cont_part;                             { Establece el número de partidas de la salida. }
            ImprimirSalidaAlmacen;                        { Imprime la salida del almacén. }
            cont_sal := Succ(cont_sal);                   { Avanza el contador de salida. }
            cont_part := 0;                               { Reinicializa el contador de partidas a cero. }
          END;
      UNTIL cont > num_ap;                        { Lo repite hasta que el contador rebasa el núm. de materiales. }
    END;
  salida := salida_mem;                           { Restaura los datos originales de la variable 'salida'. }
  cont_sal := 0;
END;
```

```
BEGIN
  IF num_sal = 0 THEN                             { Si no hay salidas registradas, sale del procedimiento. }
    BEGIN
      MensajeErr('No hay salidas registradas.');
      Exit;
    END;
  na := 4;                                        { Abre el archivo de registro de salidas. }
  bas[na].nom_arch := nom_arch_sal;
```

```
  AbrirArchivo;
  REPEAT
    ClrScr;                                  { Presenta el encabezado del procedimiento. }
    GotoXY(23,1); Write('SALIDAS-) Y A  E X I S T E N T E');
    GotoXY(70,1); Write(fecha);
    FillChar(salida,SizeOf(salida),0);       { Borra totalmente la variable 'salida'. }
    WITH salida DO
      BEGIN
        nua := num_sal_i;                    { Propondrá el folio de la primera salida registrada. }
        REPEAT
          GotoXY(1,2); ClrEol; Write('Salida: (',JusFolio(num_sal_i),'..',JusFolio(num_sal_f),') ');
          PregEnte(nua,5,FALSE);             { Pide el folio de la salida a utilizar. }
          GotoXY(22,2); Write(JusFolio(nua),' ');
        UNTIL OkFolio(nua,num_sal_i,num_sal_f);   { Se repite hasta que el folio esté correcto. }
        na := 5;                             { Abre el archivo de registro de salidas por partida. }
        bas[na].noa_arch := noa_arch_sal_p;
        AbrirArchivo;
        CargaSalida(nua,loc_sal,salida);     { Lee del disco los datos de la salida. }
        na := 5;
        CerrarArchivo;                       { Cierra el archivo de registro de salidas por partida. }
        IF origen = 'M' THEN                 { Si el origen de la salida es por maquila: }
          IF OkProveedor(clave,loc_pro) THEN { Si el proveedor está registrado. }
            BEGIN
              bas[na].noa_arch := noa_arch_des_pro; { Abre el archivo de descripción de proveedores. }
              AbrirArchivo;
              CargaProveedor(loc_pro,des_pro);      { Lee del disco los datos del proveedor. }
              CerrarArchivo;                        { Cierra el archivo de descripción de proveedores. }
            END;
        cont_sal := 0;
        MostrarSalida;                       { Muestra los datos de la salida. }
        Posibilidad(64,18,'Salida almacén'); { Presenta las posibilidades de elección. }
        Posibilidad(64,20,'Comprobante');
        REPEAT
          CASE origen OF                     { Para cada caso del origen de la salida: }
            'V' : BEGIN                      { Para salida por ventas: }
                    GotoXY(29,23); Write('¿Qué elección? (Home) (Esc) ');
                    PregOpcion(opcion,'SC' + ESC,#71);      { Pide que se tome una opción. }
                    GotoXY(29,23); ClrEol;
                    CASE opcion[1] OF                       { Para cada caso de la opción seleccionada: }
                      'S' : ImprimirSalidaAlmacen;          { Imprime la salida de almacén. }
                      'C' : ImprimirSalida;                 { Imprime el comprobante de la salida. }
                    END;
                  END;
            'P' : BEGIN
                    Posibilidad(64,16,'Orden producción');  { Presenta otra posibilidad. }
                    GotoXY(29,23); Write('¿Qué elección? (Home) (Esc) ');
                    PregOpcion(opcion,'OSC' + ESC,#71);     { Pide que se tome una opción. }
                    GotoXY(29,23); ClrEol;
                    CASE opcion[1] OF                       { Para cada caso de la opción seleccionada: }
                      'O' : ok := OkImprimirForma('FORMATOS\F_OP.TXT',1);   { Imprime la orden de producción. }
                      'S' : SalidaAlmacenProduccion;        { Imprime la salida de partes del almacén. }
                      'C' : ImprimirSalida;                 { Imprime el comprobante de la salida. }
```

```
                    END;
                  END;
            'M' : BEGIN
                    Posibilidad(64,16,'Orden maquila');        { Presenta otra posibilidad. }
                    GotoXY(29,23); Write('¿Qué elección? (Home) (Esc) ');
                    PregOpcion(opcion,'OSC' + ESC,#71);        { Pide que se tome una opción. }
                    GotoXY(29,23); ClrEol;
                    CASE opcion[1] OF                          { Para cada caso de la opción seleccionada: }
                      'O' : IF loc_pro > 0                     { Si se localizó el proveedor, imprime la orden de }
                              THEN ok := OkImprimirFormat('FORMATOS\F_OM.TXT',1)      { maquila. }
                              ELSE MensajeErr('Ya no está registrado este proveedor.');
                      'S' : SalidaAlmacenMaquila;              { Imprime la salida de partes del almacén. }
                      'C' : ImprimirSalida;                    { Imprime el comprobante de la salida. }
                    END;
                  END;
          END;
        UNTIL (opcion[1] = ESC) OR                   { Se repite hasta que se apriete la tecla "Esc" o "Home". }
              (opcion[2] = #71);
      END;
    UNTIL opcion[1] = ESC;                           { Se repite hasta que se apriete la tecla de escape. }
    na := 4;                                         { Cierra el archivo de registro de salidas. }
    CerrarArchivo;
  END;
```

```
BEGIN
  REPEAT
    ClrScr;                                          { Presenta el encabezado del procedimiento. }
    GotoXY(33,1); Write('S A L I D A S');
    GotoXY(70,1); Write(fecha);
    Posibilidad(28, 5,'Y A   E X I S T E N T E S');  { Presenta las posibilidades de elección. }
    Posibilidad(28, 7,'N U E V A S');
    GotoXY(55,23); Write('¿Qué elección? (Esc) ');
    PregOpcion(opcion,'YN' + ESC,'');                { Pide que se tome una opción. }
    CASE opcion[1] OF                                { Ejecuta el procedimiento seleccionado. }
      'Y' : YaExistentes;
      'N' : Nuevas;
    END;
  UNTIL opcion[1] = ESC;                             { Se repite hasta que se presiona la tecla de escape. }
END;
```

Por medio de este procedimiento se pueden registrar las salidas e imprimirlas en diferentes formatos.

4.3.10 SUBPROGRAMA PARA CONTROL DEL DINERO.

Este subprograma está formado por un procedimiento de solapamiento cuyo fichero incluido se lista a continuación.

D.OVL { Fichero que contiene el código fuente que se muestra. }

```
Overlay PROCEDURE Dinero;                          { Procedimiento de solapamiento para el control del dinero. }

  VAR
    opcion: TTecla;                                { Opción seleccionada. }

  PROCEDURE Cobros;                                { Registra los cobros hechos por los distribuidores. }

    VAR
      loc,                                         { Localización de algún registro. }
      loc_dis: Integer;                            { Localización del distribuidor. }
      saldo:   Real;                               { Saldo acreedor del distribuidor. }
      nombre:  TNombre;                            { Nombre del distribuidor. }
      cobro:   TCob;                               { Datos del cobro recibido. }
      opcion:  TTecla;                             { Opción seleccionada. }

    FUNCTION OkDistribuidor                        { Insertar dicha función. }

    BEGIN
      na := 4;                                     { Abre el archivo de descripción de distribuidores. }
      bas[na].nom_arch := nom_arch_des_dis;
      AbrirArchivo;
      na := 5;                                     { Abre el archivo de registro de cobros. }
      bas[na].nom_arch := nom_arch_cob;
      AbrirArchivo;
      REPEAT
        ClrScr;                                    { Presenta el encaberado del procedimiento. }
        GotoXY(32,1); Write('DINERO-> C O B R O S');
        GotoXY(70,1); Write(fecha);
        GotoXY(1,5);                               { Presenta la pantalla de registro de cobros. }
        WriteLn('                                                                              ');
        WriteLn('  Clave:          Nombre:                                                     ');
        WriteLn('                                                                              ');
        WriteLn('  Forma de cobro:                  Saldo anterior: $                 MN      ');
        WriteLn('                                   Importe del cobro: $              MN      ');
        WriteLn('                                                      ------------            ');
        WriteLn('                                   Saldo nuevo: $                    MN      ');
        WriteLn('                                                                              ');
        FillChar(cobro,SizeOf(cobro),0);           { Borra totalmente la variable 'cobro'. }
        WITH cobro DO
          BEGIN
```

```
        REPEAT
          GotoXY(10,6);
          PregCadMay(clave,TAM_CLAVE);          { Pide la clave del distribuidor. }
          IF clave = '' THEN                    { Si no se introdujo clave: }
            BEGIN
              na := 4;                          { Cierra el archivo de descripción de distribuidores. }
              CerrarArchivo;
              na := 5;                          { Cierra el archivo de registro de cobros. }
              CerrarArchivo;
              Exit;                             { Termina la ejecución del procedimiento. }
            END;
        UNTIL OkDistribuidor(clave,loc_dis);    { Se repite hasta que la clave introducida esté correcta. }
        na := 4;
        LeerCad(loc_dis,2,nombre);              { Lee del disco el nombre del distribuidor y lo presenta en }
        GotoXY(29,6); Write(nombre);            { pantalla. }
        LeerReal(loc_dis,12,saldo);             { Lee del disco el saldo del distribuidor y lo presenta en }
        GotoXY(59,8); Write(JusReal(saldo,14,2)); { pantalla. }
        GotoXY(19,8);
        PregCad(forma_pag,TAM_FORM_PAG);        { Pide la forma de pago. }
        REPEAT
          GotoXY(59,9);
          PregReal(importe,14,2,FALSE);         { Pide el importe del cobro. }
        UNTIL importe > 0;                      { Se repite hasta que el importe del cobro sea mayor a cero. }
        GotoXY(59,11); Write(JusReal(saldo + importe,14,2));
        GotoXY(29,13); Write('¿Todo correcto? (S/N) ');
        PregOpcion(opcion,'SN','');             { Pide que se tome una opción. }
        IF opcion[1] = 'S' THEN                 { Si fue "S": }
          BEGIN
            na := 4;
            saldo := saldo + importe;           { Actualiza el saldo del distribuidor. }
            EscribirReal(loc_dis,12,saldo);
            fech := fecha;                      { La fecha del cobro será la actual. }
            na := 5;
            loc := bas[na].num_regs + 1;        { Calcula la localización del registro del cobro a agregar. }
            EscribirCad(loc,1,clave);           { Registra los datos del cobro. }
            EscribirFecha(loc,2,fech);
            EscribirCad(loc,3,forma_pag);
            EscribirReal(loc,4,importe);
          END;
      END;
    UNTIL FALSE;                                { Se repite indefinidamente. }
  END;

{-------------------------------------------------------------------------------}

PROCEDURE Pagos;                                { Registra los pagos hechos a los proveedores. }

  VAR
    loc,                                        { Localización de algún registro. }
    loc_pro: Integer;                           { Localización del proveedor. }
    saldo:   Real;                              { Saldo acreedor con el proveedor. }
    nombre:  TNombre;                           { Nombre del proveedor. }
    pago:    TPag;                              { Datos del pago hecho. }
```

```
    opcion: TTecla;                          { Opción seleccionada. }

  {FUNCTION OkProveedor                      { Insertar dicha función. }

  BEGIN
    na := 4;                                 { Abre el archivo de descripción de proveedores. }
    bas[na].nom_arch := nom_arch_des_pro;
    AbrirArchivo;
    na := 5;                                 { Abre el archivo de registro de pagos. }
    bas[na].nom_arch := nom_arch_pag;
    AbrirArchivo;
    REPEAT
      ClrScr;                                { Presenta el encabezado del procedimiento. }
      GotoXY(31,1); Write('DINERO-> P A G O S');
      GotoXY(70,1); Write(fecha);
      GotoXY(1,5);                           { Presenta la pantalla de registro de pagos. }
      WriteLn('
      WriteLn('  Clave:          Nombre:                                                  ');
      WriteLn('
      WriteLn('  Forma de pago:                  Saldo anterior: $              MN      ');
      WriteLn('                                  Importe del pago: $            MN      ');
      WriteLn('                                                   ----------             ');
      WriteLn('                                  Saldo nuevo: $                 MN      ');
      WriteLn('
      FillChar(pago,SizeOf(pago),0);         { Borra totalmente la variable 'pago'. }
      WITH pago DO
        BEGIN
          REPEAT
            GotoXY(10,6);
            PregCadMay(clave,TAM_CLAVE);     { Pide la clave del proveedor. }
            IF clave = '' THEN               { Si no se introdujo clave: }
              BEGIN
                na := 4;                     { Cierra el archivo de descripción de proveedores. }
                CerrarArchivo;
                na := 5;                     { Cierra el archivo de registro de pagos. }
                CerrarArchivo;
                Exit;                        { Termina la ejecución del procedimiento. }
              END;
          UNTIL OkProveedor(clave,loc_pro);  { Se repite hasta que la clave introducida esté correcta. }
          na := 4;
          LeerCad(loc_pro,2,nombre);         { Lee del disco el nombre del proveedor y lo presenta en }
          GotoXY(29,6); Write(nombre);       { pantalla. }
          LeerReal(loc_pro,9,saldo);         { Lee del disco el saldo con el proveedor y lo presenta en }
          GotoXY(59,8); Write(JustReal(saldo,14,2));  { pantalla. }
          GotoXY(18,8);
          PregCad(forma_pag,TAM_FORM_PAG);   { Pide la forma de pago. }
          REPEAT
            GotoXY(59,9);
            PregReal(importe,14,2,FALSE);    { Pide el importe del pago. }
          UNTIL importe > 0;                 { Se repite hasta que el importe del pago sea mayor a cero. }
```

```
        GotoXY(59,11); Write(JusReal(saldo + importe,14,2));
        GotoXY(29,13); Write('¿Todo correcto? (S/N) ');
        PregOpcion(opcion,'SN',''');                    { Pide que se tome una opción. }
        IF opcion[1] = 'S' THEN                         { Si fue "S": }
          BEGIN
            na := 4;
            saldo := saldo + importe;                   { Actualiza el saldo con el proveedor. }
            EscribirReal(loc_pro,9,saldo);
            fech := fecha;                              { La fecha del pago será la actual. }
            na := 5;
            loc := bas[na].num_regs + 1;                { Calcula la localización del registro del pago a agregar. }
            EscribirCad(loc,1,clave);                   { Registra los datos del pago. }
            EscribirFecha(loc,2,fech);
            EscribirCad(loc,3,forma_pag);
            EscribirReal(loc,4,importe);
          END;
      END;
    UNTIL FALSE;                                        { Se repite indefinidamente. }
  END;
```

```
BEGIN
  REPEAT
    ClrScr;                                             { Presenta el encabezado del procedimiento. }
    GotoXY(35,1); Write('D I N E R O');
    GotoXY(70,1); Write(fecha);
    Posibilidad(26,5,'C O B R O S (Distribuidores)');   { Presenta las posibilidades de elección. }
    Posibilidad(26,7,'P A G O S   (Proveedores)');
    GotoXY(55,23); Write('¿Qué elección? (Esc) ');
    PregOpcion(opcion,'CP' + ESC,''');                  { Pide que se tome una opción. }
    CASE opcion[1] OF                                   { Ejecuta el procedimiento seleccionado. }
      'C' : Cobros;
      'P' : Pagos;
    END;
  UNTIL opcion[1] = ESC;                                { Se repite hasta que se presione la tecla de escape. }
END;
```

Este procedimiento se utiliza para registrar los cobros hechos a distribuidores y los pagos efectuados a proveedores.

4.3.11 SUBPROGRAMA DE AJUSTES AL INVENTARIO.

Este subprograma está formado por un solo procedimiento de solapamiento. A continuación se lista el fichero incluido que contiene su código fuente.

A.OVL (Fichero que contiene el código fuente que se muestra.)

```
Overlay PROCEDURE Ajustes;                          ( Procedimiento de solapamiento para el ajuste de inventarios. )

  VAR
    opcion: TTecla;                                 ( Opción seleccionada. )

  FUNCTION OkMaterial                               ( Insertar dicha función. )

  FUNCTION OkProducto                               ( Insertar dicha función. )

  PROCEDURE Inventarios;                            ( Procedimiento para el ajuste a inventarios en la bodega. )

    VAR
      ok,                                           ( Indica que está correcto. )
      producto:  Boolean;                           ( Indica que se trata de un producto terminado. )
      loc,                                          ( Localización de algún registro. )
      loc_artic,                                    ( Localización del artículo. )
      exist:     Integer;                           ( Existencias del artículo. )
      aju:       TAju;                              ( Datos del ajuste. )
      opcion:    TTecla;                            ( Opción seleccionada. )

    BEGIN
      REPEAT
        ClrScr;                                     ( Presenta el encabezado del procedimiento. )
      GotoXY(25,1); Write('AJUSTES-> I N V E N T A R I O S');
      GotoXY(70,1); Write(fecha);
      GotoXY(1,5);                                  ( Presenta la pantalla de ajuste a inventarios en bodega. )
      WriteLn('                                                                              ');
      WriteLn('  Ajuste:          | Origen: Inventarios | Razón:                             ');
      WriteLn('                                                                              ');
      WriteLn('  Artículo:                              Inventarios en la bodega:           ');
      WriteLn('    Precio: $                                  Cantidad de ajuste:          ');
      WriteLn('                                                                  -------     ');
      WriteLn('                                       Nuevos inventarios en la bodega:       ');
      WriteLn('                                                                              ');
      FillChar(aju,SizeOf(aju),0);                  ( Borra completamente la variable 'aju'. )
      producto := FALSE;                            ( Por defecto indica que no es un producto terminado. )
      WITH aju DO
        BEGIN
          num := SigFolio(num_aju_f);
```

```
IF num_aju = 0 THEN                          { Si no hay ajustes registrados, pide que se especifique el }
  BEGIN                                      { folio (mayor a cero) del primer ajuste. }
    MensajeErr('No se tienen ajustes registrados. Especificar el folio inicial.');
    REPEAT
      num := 1;
      GotoXY(11,6); PregEnte(num,5,FALSE);
    UNTIL num > 0;
  END;
GotoXY(11,6); Write(JusFolio(num),' ');      { Presenta el folio del ajuste. }
ok := FALSE;                                 { Por defecto no está correcto. }
REPEAT
  GotoXY(13,8);
  PregCadMay(artic,TAM_ARTIC);               { Pide la clave del artículo. }
  IF artic = '' THEN Exit;                   { Si no se introdujo clave, termina la ejecución. }
  IF OkProducto(artic,loc_artic)
    THEN                                     { Si es un producto terminado: }
      BEGIN
        producto := TRUE;                    { Indica que es un producto terminado. }
        ok := TRUE;                          { Indica que está correcto. }
      END
    ELSE                                     { Si no es un producto terminado: }
      IF OkMaterial(artic,loc_artic)         { Si es un material: }
        THEN ok := TRUE                      { Indica que está correcto. }
        ELSE MensajeErr('No está registrado este artículo.');
UNTIL ok;                                    { Se repite hasta que esté correcto. }
GotoXY(47,6); PregByte(razon,3);             { Pide el código de la razón del ajuste. }
IF producto                                  { El precio del artículo será el último costo del mismo. }
  THEN precio := pnt_pt[loc_artic]^.precio_ult
  ELSE precio := pnt_mp[loc_artic]^.precio_ult;
GotoXY(15,9); Write(JusReal(precio,14,2));   { Presenta el precio en pantalla. }
IF producto
  THEN exist := pnt_pt[loc_artic]^.exist
  ELSE exist := pnt_mp[loc_artic]^.exist;
GotoXY(68,8); Write(JusEnte(exist,6));       { Presenta las existencias del artículo en bodega. }
REPEAT
  GotoXY(67,9);
  IF con_llave
    THEN PregEnte(cant,6,FALSE)              { Si tiene llave, pide una cantidad de ajuste positiva. }
    ELSE PregEnte(cant,7,TRUE);              { Si no tiene llave, pide la cantidad de ajuste. }
UNTIL (exist + cant >= 0) AND (cant <> 0);   { Se repite hasta que la cantidad de ajuste sea válida. }
GotoXY(68,11);
Write(JusEnte(exist + cant,6));              { Presenta las nuevas existencias. }
GotoXY(29,14); Write('¿Todo correcto? (S/N) ');
PregOpcion(opcion,'SN','');                  { Pide que se tecle una opción. }
IF opcion[1] = 'S' THEN                      { Si fue "S": }
  BEGIN
    fech := fecha;                           { La fecha del ajuste será la actual. }
    origen := 'I';                           { El origen del ajuste será por inventarios en bodega. }
    na := 4;
    loc := bas[na].nua_regs + 1;             { Calcula la localización del registro de ajuste a agregar. }
    EscribirEnte(loc,1,num);                 { Registra el ajuste. }
    EscribirCar(loc,2,origen);
```

```
            EscribirByte(loc,3,razon);
            EscribirCad(loc,4,artic);
            EscribirEnte(loc,5,cant);
            EscribirReal(loc,6,precio);
            EscribirFecha(loc,7,fech);
            num_aju := Succ(num_aju);          { Actualiza el contador de ajustes registrados. }
            num_aju_f := num;                  { Actualiza el folio del último ajuste registrado. }
            exist := exist + cant;             { Calcula la nueva existencia del artículo. }
            IF producto
              THEN                             { Si es un producto terminado, actualiza las existencias y }
                BEGIN                          { la cantidad disponible tanto en la memoria como en el disco. }
                  na := 2;
                  pnt_pt[loc_artic]^.exist := exist;
                  EscribirEnte(loc_artic,2,pnt_pt[loc_artic]^.exist);
                  pnt_pt[loc_artic]^.ca_dis := pnt_pt[loc_artic]^.ca_dis + cant;
                  EscribirEnte(loc_artic,3,pnt_pt[loc_artic]^.ca_dis);
                END
              ELSE                             { Si es un material, actualiza las existencias tanto en }
                BEGIN                          { la memoria como en el disco. }
                  na := 1;
                  pnt_mp[loc_artic]^.exist := exist;
                  EscribirEnte(loc_artic,2,pnt_mp[loc_artic]^.exist);
                END;
          END;
      END;
  UNTIL FALSE;                                 { Se repite indefinidamente. }
END;

{-------------------------------------------------------------------------------}
PROCEDURE Compras;                             { Procedimiento para el ajuste de la cantidad pedida a }
                                               { proveedores de materias primas. }
  VAR
    ok:        Boolean;                        { Indica que está correcto. }
    loc_artic: Integer;                        { Localización del artículo. }
    aju:       TAju;                           { Datos del ajuste. }
    opcion:    TTecla;                         { Opción seleccionada. }

  BEGIN
    REPEAT
      ClrScr;                                  { Presenta el encabezado del procedimiento. }
      GotoXY(29,1); Write('AJUSTES-> C O M P R A S');
      GotoXY(70,1); Write(fecha);
      GotoXY(1,5);                             { Presenta la pantalla de ajustes a compras. }
      WriteLn('                                                                ');
      WriteLn('  Ajuste: XXXX  |  Origen: Compras  |  Razón: 0                 ');
      WriteLn('                                                                ');
      WriteLn('  Materia prima:                        Cantidad pedida:        ');
      WriteLn('                                        Cantidad de ajuste:     ');
      WriteLn('                                                        ------  ');
      WriteLn('                                        Nueva cantidad pedida:  ');
      WriteLn('                                                                ');
      FillChar(aju,SizeOf(aju),0);             { Borra totalmente la variable 'aju'. }
```

```
    WITH aju DO
      BEGIN
        ok := FALSE;                          { Por defecto no está correcto. }
        REPEAT
          GotoXY(18,8);
          PregCadMay(artic,TAM_ARTIC);        { Pide la clave de la materia prima. }
          IF artic = '' THEN Exit;            { Si no se introdujo clave, termina la ejecución. }
          IF OkMaterial(artic,loc_artic)
            THEN                              { Si está registrada la clave del material: }
              IF (artic[1] <> CLA_MAQ) AND (artic[1] <> CLA_SUBE)
                THEN ok := TRUE               { Si no es una maquila ni subensamble, indica que está correcto. }
                ELSE MensajeErr('No se pueden ajustar desde compras los subensambles ni las maquilas.')
            ELSE MensajeErr('No está registrada esta materia prima.');
        UNTIL ok;                             { Se repite hasta que esté correcto. }
        GotoXY(68,8);
        Write(JusEnte(pnt_ap[loc_artic]^.ca_ped,6)); { Presenta la cantidad pedida. }
        REPEAT
          GotoXY(67,9);
          IF con_llave
            THEN PregEnte(cant,6,FALSE)       { Si tiene llave, pide una cantidad de ajuste positiva. }
            ELSE PregEnte(cant,7,TRUE);       { Si no tiene llave, pide una cantidad de ajuste. }
        UNTIL (pnt_ap[loc_artic]^.ca_ped + cant >= 0) AND (cant <> 0);  { Se repite hasta que la cantidad sea válida. }
        GotoXY(68,11);
        Write(JusEnte(pnt_ap[loc_artic]^.ca_ped + cant,6));    { Presenta la nueva cantidad pedida. }
        GotoXY(29,14); Write('¿Todo correcto? (S/N) ');
        PregOpcion(opcion,'SN','');           { Pide que se tome una opción. }
        IF opcion[1] = 'S' THEN               { Si fue "S", actualiza la cantidad pedida del artículo tanto en }
          BEGIN                               { la memoria como en el disco. }
            na := 1;
            pnt_ap[loc_artic]^.ca_ped := pnt_ap[loc_artic]^.ca_ped + cant;
            EscribirEnte(loc_artic,3,pnt_ap[loc_artic]^.ca_ped);
          END;
      END;
  UNTIL FALSE;                                { Se repite indefinidamente. }
END;

{-------------------------------------------------------------------------------}
PROCEDURE Produccion;                         { Procedimiento para el ajuste de la cantidad pedida a }
                                              { producción. }
  VAR
    ok,                                       { Indica que está correcto. }
    producto: Boolean;                        { Indica que es un producto terminado. }
    loc,                                      { Localización de algún registro. }
    loc_artic,                                { Localización de un artículo. }
    exist:    Integer;                        { Existencias del artículo en producción. }
    aju:      TAju;                           { Datos del ajuste. }
    opcion:   TTecla;                         { Opción seleccionada. }

  BEGIN
    REPEAT
      ClrScr;                                 { Presenta el encabezado del procedimiento. }
      GotoXY(26,1); Write('AJUSTES-> P R O D U C C I O N');
```

```
    GotoXY(70,1); Write(fecha);
    GotoXY(1,5);                          { Presenta la pantalla de ajustes a materiales en producción. }
    WriteLn('                                                                              ');
    WriteLn('  Ajuste:      Origen: Producción   Razón:                                    ');
    WriteLn('                                                                              ');
    WriteLn('  Artículo:                    Cantidad pedida a producción:                  ');
    WriteLn('    Precio: $                          Cantidad de ajuste:                    ');
    WriteLn('                                                                              ');
    WriteLn('                              Nueva cantidad pedida producción:               ');
    WriteLn('                                                                              ');
    FillChar(aju,SizeOf(aju),0);          { Borra totalmente la variable 'aju'. }
    producto := FALSE;                    { Por defecto es un producto terminado. }
    WITH aju DO
      BEGIN
        num := SigFolio(num_aju_f);
        IF num_aju = 0 THEN               { Si no hay ajustes registrados, pide que se especifique el }
          BEGIN                           { folio (mayor a cero) del primer ajuste. }
            MensajeErr('No se tienen ajustes registrados. Especificar el folio inicial.');
            REPEAT
              num := 1;
              GotoXY(11,6); PregEnte(num,5,FALSE);
            UNTIL num > 0;
          END;
        GotoXY(11,6); Write(JusFolio(num),' ');   { Presenta el folio del ajuste. }
        REPEAT
          ok := FALSE;                    { Por defecto no está correcto. }
          GotoXY(13,8);
          PregCadMay(artic,TAM_ARTIC);    { Pide la clave del artículo. }
          IF artic = '' THEN Exit;        { Si no se introdujo clave, termina la ejecución. }
          IF OkProducto(artic,loc_artic)
            THEN                          { Si está registrada la clave como producto terminado: }
              BEGIN
                producto := TRUE;         { Indica que es un producto terminado. }
                ok := TRUE;               { Indica que está correcto. }
              END
            ELSE                          { Si no está registrada la clave como producto terminado: }
              IF OkMaterial(artic,loc_artic)
                THEN                      { Si está registrada la clave como material: }
                  IF artic[1] = CLA_SUBE
                    THEN ok := TRUE       { Si contiene el carácter para subensambles, está correcto. }
                    ELSE MensajeErr('No se pueden ajustar las compras ni maquilas desde esta sección.')
                ELSE MensajeErr('No está registrado este artículo.');
        UNTIL ok;                         { Se repite hasta que esté correcto. }
        GotoXY(47,6); PregByte(razon,3);  { Pide el código de la razón del ajuste. }
        IF producto                       { El precio será el costo último del artículo. }
          THEN precio := pnt_pt[loc_artic]^.precio_ult
          ELSE precio := pnt_mp[loc_artic]^.precio_ult;
        GotoXY(15,9); Write(JusReal(precio,14,2));   { Presenta el precio del artículo. }
        IF producto                       { Las existencias serán la cantidad pedida a producción. }
          THEN exist := pnt_pt[loc_artic]^.ca_ped
          ELSE exist := pnt_mp[loc_artic]^.ca_ped;
        GotoXY(68,8); Write(JusEnte(exist,6));       { Presenta en pantalla las existencias. }
```

```
        REPEAT
          GotoXY(67,9);
          IF con_llave
            THEN PregEnte(cant,6,FALSE)          { Si tiene llave, pide una cantidad de ajuste positiva. }
            ELSE PregEnte(cant,7,TRUE);          { Si no tiene llave, pide una cantidad de ajuste. }
        UNTIL (exist + cant )= 0) AND (cant <> 0); { Se repite hasta que la cantidad de ajuste sea válida. }
        GotoXY(68,11);
        Write(JusEnte(exist + cant,6));          { Presenta las nuevas existencias. }
        GotoXY(29,14); Write('¿Todo correcto? (S/N) ');
        PregOpcion(opcion,'SN',' ');             { Pide que se tome una opción. }
        IF opcion[1] = 'S' THEN                  { Si fue "S": }
          BEGIN
            fech := fecha;                       { La fecha del ajuste será la actual. }
            origen := 'P';                       { El origen del ajuste será producción. }
            na := 4;
            loc := bas[na].num_regs + 1;         { Calcula la localización del registro de ajuste a agregar. }
            EscribirEnte(loc,1,num);             { Registra el ajuste. }
            EscribirCar(loc,2,origen);
            EscribirByte(loc,3,razon);
            EscribirCad(loc,4,artic);
            EscribirEnte(loc,5,cant);
            EscribirReal(loc,6,precio);
            EscribirFecha(loc,7,fech);
            num_aju := Succ(num_aju);            { Actualiza el contador de ajustes registrados. }
            num_aju_f := num;                    { Actualiza el folio del último ajuste registrado. }
            exist := exist + cant;
            IF producto                          { Actualiza la cantidad pedida en la memoria y en el disco. }
              THEN
                BEGIN
                  na := 2;
                  pnt_pt[loc_artic]^.ca_ped := exist;
                  EscribirEnte(loc_artic,4,pnt_pt[loc_artic]^.ca_ped);
                END
              ELSE
                BEGIN
                  na := 1;
                  pnt_mp[loc_artic]^.ca_ped := exist;
                  EscribirEnte(loc_artic,3,pnt_mp[loc_artic]^.ca_ped);
                END;
          END;
      END;
    UNTIL FALSE;                                 { Se repite indefinidamente. }
  END;

{------------------------------------------------------------------------------}

PROCEDURE Maquila;                               { Procedimiento para el ajuste a la cantidad pedida a maquila. }

  VAR
    ok:       Boolean;                           { Indica que está correcto. }
    loc,                                         { Localización de algún registro. }
    loc_artic,                                   { Localización del artículo. }
    exist:    Integer;                           { Existencias del artículo con el proveedor de maquila. }
```

```
  aju:      TAju;                          { Datos del ajuste. }
  opcion:   TTecla;                        { Opción seleccionada. }

BEGIN
  REPEAT
    ClrScr;                                { Presenta el encabezado del procedimiento. }
    GotoXY(28,1); Write('AJUSTES-> M A Q U I L A S');
    GotoXY(70,1); Write(fecha);
    GotoXY(1,5);                           { Presenta la pantalla de ajuste a la cantidad pedida a maquila. }
    WriteLn('                                                                        ');
    WriteLn('  Ajuste:      | Origen: Maquilas | Razón:                             ');
    WriteLn('                                                                        ');
    WriteLn('  Artículo:                        Cantidad pedida a maquila:          ');
    WriteLn('    Precio: $                        Cantidad de ajuste:               ');
    WriteLn('                                                        ------          ');
    WriteLn('  Clave del proveedor:           Nueva cantidad pedida a maquila:       ');
    WriteLn('                                                                        ');
    FillChar(aju,SizeOf(aju),0);           { Borra totalmente la variable 'aju'. }
    WITH aju DO
      BEGIN
        num := SigFolio(num_aju_f);
        IF num_aju = 0 THEN                { Si no hay ajustes registrados, pide que se especifique el }
          BEGIN                            { folio (mayor a cero) del primer ajuste. }
            MensajeErr('No se tienen ajustes registrados. Especificar el folio inicial.' );
            REPEAT
              nue := 1;
              GotoXY(11,6); PregEnte(num,5,FALSE);
            UNTIL num > 0;
          END;
        GotoXY(11,6); Write(JusFolio(num),' ');     { Presenta el folio del ajuste. }
        REPEAT
          ok := FALSE;                     { Por defecto no está correcto. }
          GotoXY(13,8);
          PregCadMay(artic,TAM_ARTIC);     { Pide la clave del producto de maquila. }
          IF artic = '' THEN Exit;         { Si no se introdujo clave, termina la ejecución. }
          IF artic[1] = CLA_MAQ
            THEN                           { Si contiene la clave para maquilas. }
              IF OkMaterial(artic,loc_artic)
                THEN ok := TRUE            { Si está registrado el material, indica que está correcto. }
                ELSE MensajeErr('No está registrado este producto de maquila.')
            ELSE MensajeErr('En esta sección sólo se pueden ajustar los productos de maquila.');
        UNTIL ok;                          { Se repite hasta que esté correcto. }
        GotoXY(47,6); PregByte(razon,3);   { Pide el código de la razón del ajuste. }
        precio := pnt_mp[loc_artic]^.precio_ult;    { El precio será el último costo del producto de maquila. }
        GotoXY(15,9);
        Write(JusReal(precio,14,2));       { Presenta el precio. }
        exist := pnt_mp[loc_artic]^.ca_ped;         { Las existencias serán la cantidad pedida a maquila. }
        GotoXY(68,8); Write(JusEnte(exist,6));      { Presenta las existencias. }
        REPEAT
          GotoXY(67,9);
          IF con_llave
            THEN PregEnte(cant,6,FALSE)    { Si tiene llave, pide una cantidad de ajuste positiva. }
```

```
            ELSE PregEnte(cant,7,TRUE);          { Si no tiene llave, pide una cantidad de ajuste. }
          UNTIL (exist + cant >= 0) AND (cant <> 0); { Se repite hasta que la cantidad de ajuste sea válida. }
          GotoXY(68,11);
          Write(JusEnte(exist + cant,6));          { Presenta las nuevas existencias en maquila. }
          GotoXY(24,11);
          Write(pnt_ap[loc_artic]^.clave);         { Presenta la clave del último proveedor de maquila. }
          GotoXY(29,14); Write('¿Todo correcto? (S/N) ');
          PregOpcion(opcion,'SN',' ');             { Pide que se tome una opción. }
          IF opcion[1] = 'S' THEN                  { Si fue "S": }
            BEGIN
              fech := fecha;                       { La fecha del ajuste será la actual. }
              origen := 'M';                       { El origen del ajuste será maquila. }
              na := 4;
              loc := bas[na].num_regs + 1;         { Calcula la localización del registro de ajuste a agregar. }
              EscribirEnte(loc,1,num);             { Registra el ajuste. }
              EscribirCar(loc,2,origen);
              EscribirByte(loc,3,razon);
              EscribirCad(loc,4,artic);
              EscribirEnte(loc,5,cant);
              EscribirReal(loc,6,precio);
              EscribirFecha(loc,7,fech);
              num_aju := Succ(num_aju);            { Actualiza el contador de ajustes registrados. }
              num_aju_f := num;                    { Actualiza el folio del último ajuste registrado. }
              exist := exist + cant;
              na := 1;                             { Actualiza la cantidad pedida en la memoria y en el disco. }
              pnt_ap[loc_artic]^.ca_ped := exist;
              EscribirEnte(loc_artic,3,pnt_ap[loc_artic]^.ca_ped);
            END;
        END;
    UNTIL FALSE;                                   { Se repite indefinidamente. }
  END;
```

```
  BEGIN
    na := 4;                                       { Abre el archivo de registro de ajustes. }
    bas[na].nom_arch := nom_arch_aju;
    AbrirArchivo;
    REPEAT
      ClrScr;                                      { Presenta el encabezado del procedimiento. }
      GotoXY(33,1); Write('A J U S T E S');
      GotoXY(70,1); Write(fecha);
      Posibilidad(30,5,'I N V E N T A R I O S');   { Presenta las posibilidades de elección. }
      Posibilidad(30,7,'C O M P R A S');
      Posibilidad(30,9,'P R O D U C C I O N');
      Posibilidad(30,11,'M A Q U I L A');
      GotoXY(55,22); Write('¿Qué elección? (Esc) ');
      PregOpcion(opcion,'ICPM' + ESC,' ');         { Pide que se tome una opción. }
      CASE opcion[1] OF                            { Ejecuta el procedimiento seleccionado. }
        'I' : Inventarios;
        'C' : Compras;
        'P' : Produccion;
        'M' : Maquila;
```

```
    END;
  UNTIL opcion[1] = ESC;                    { Se repite hasta que se presione la tecla de escape. }
  na := 4;                                  { Cierra el archivo de registro de ajustes. }
  CerrarArchivo;
END;
```

Este procedimiento lleva a cabo el registro de los ajustes al inventario de existencias en la bodega y a la cantidad pedida a proveedores o a producción.

4.3.12 SUBPROGRAMA DE RESULTADOS Y UTILERIAS.

Cada uno de estos dos subprogramas está formado por un procedimiento de solapamiento. Estos procedimientos se han dejado en blanco para que en ellos se desarrollen las implementaciones que se consideren necesarías. A continuación se muestran los ficheros incluidos que contienen el código fuente de ambos procedimientos.

R.OVL (Fichero que contiene el código fuente que se muestra.)

```
Overlay PROCEDURE Resultados;          ( Procedimiento de solapamiento para obtención de resultados. )

  BEGIN                                ( Este procedimiento se deja en blanco para su posterior )
  END;                                 ( realización. )
```

U.OVL (Fichero que contiene el código fuente que se muestra.)

```
Overlay PROCEDURE Utilerias;           ( Procedimiento de solapamiento para utilerías. )

  BEGIN                                ( Este procedimiento se deja en blanco para su posterior )
  END;                                 ( realización. )
```

En el procedimiento de resultados se aconseja colocar procedimientos que den respuestas en base a la información contenida en las bases de datos. Por ejemplo: las ventas totales del mes, la cartera de distribuidores, etc.

En el procedimiento de utilerías se pueden colocar procedimientos para el mantenimiento en general. Por ejemplo: procedimientos para cambiar la hora del sistema, para sacar respaldo de los archivos a un disco flexible, para actualizar los costos de las materias primas y procesos en forma manual y los costos restantes en forma automática, etc..

Con esto concluye la etapa de programación y en el siguiente capítulo, se presentan los pasos que se siguieron para instalar el programa.

Para que el programa opere correctamente, es necesario que se instale en la computadora siguiendo los pasos que se citan a continuación.

5.1 PUESTA EN MARCHA.

En primer lugar, es importante mencionar que computadora con que se cuenta en la empresa en cuestión es una computadora compatible con la IBM/PC XT, que cuenta con: un reloj de 8 MHz, memoria RAM de 640 *KBytes*, un disco en RAM de 380 *KBytes*, un disco duro de 20 *MBytes*, una lectora de discos flexibles de 5¼" con capacidad para 360 *KBytes*, un monitor a color RGB (*Red, Green & Blue*) y una impresora para hojas de 8½" de ancho totalmente compatible.

La configuración mínima para que el programa funcione es:

+ Una computadora compatible con la IBM/PC.

+ Memoria RAM de 256 *KBytes*, tomando en cuenta que, con dicha memoria, el programa funcionará correctamente pero con una limitación muy grande en la cantidad de materiales, productos terminados y descripciones para producción y maquila que se pueden manejar.

+ La capacidad de la memoria permanente (disco) está directamente relacionada con la capacidad de archivo, por lo que no se recomienda su utilización en disco flexible. En caso necesario, es posible adaptar el programa para que pueda operar en dos discos flexibles, con lo que se puede llegar a tener una capacidad moderada de almacenamiento.

+ Un monitor y una impresora paralela conectados a la computadora. Si se trata de un monitor monocromático, se debe cambiar en el fichero "SIAC.PAS" la dirección de memoria de la variable 'pantalla' de "$B800" a "$B000".

Considerando que la computadora con que cuenta la empresa posee un disco duro, se muestra a continuación la etapa de instalación del programa en dicho disco. Los pasos que se siguieron son:

PASOS PARA COMPILAR EL PROGRAMA:

Para poder compilar el programa se requiere, como mínimo, de 1 *MByte* de capacidad de almacenamiento en disco, y se necesita:

1 - Crear, en el directorio raíz, un directorio llamado, por ejemplo, "SIAC". Colocar en él el fichero llamado "SIAC.PAS". Indicar, en el código fuente que contiene el programa mencionado, la secuencia de seis caracteres que componen la llave del sistema (variable llave).

2.- Crear, dentro del directorio "SIAC", un directorio llamado "INCLUIDO", y poner dentro de éste todos los procedimientos con extensión "INC" y "OVL". Los ficheros con extensión "BBL" también se pueden colocar en dicho directorio, tomando en cuenta que estos ficheros no forman parte del programa, sino que se utilizan para copiar partes de ellos dentro de los ficheros que forman el programa.

3.- Correr el lenguaje Pascal, señalar el directorio y camino en donde se localiza el fichero "SIAC.PAS", indicar que éste es el fichero de trabajo, pedir la compilación a disco e iniciar la compilación del programa.

4.- Salir del lenguaje pascal.

INSTALACION EN EL DISCO.

Con la compilación del programa se obtiene el código objeto, que se localiza dentro del directorio de nombre "SIAC". A continuación se deben preparar las bases de datos y los formatos de impresión, mediante los siguientes pasos:

1.- Crear dentro del directorio "SIAC" los directorios "BASES" y "FORMATOS".

2.- Correr el dBASE III y crear las bases de datos (que se describen al principio del Capítulo IV) dentro del directorio "BASES".

3.- En la base de datos "CONT.dbf" se deben introducir los folios iniciales respectivos para: las devoluciones, facturas, notas de crédito, órdenes de maquila y órdenes de producción. El valor que se introduzca representa el siguiente folio a utilizar. El orden de los registros no se debe cambiar para lograr que los folios se asignen correctamente. Una vez que el sistema ya esté operando, esta base de datos no debe de ser alterada. Por ejemplo:

Regist#	DESC	CONT
1	Devolución	1
2	Factura	1
3	Nota de crédito	1
4	Orden de maquila	1
5	Orden de producción	1

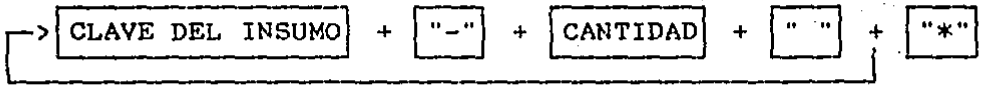
4.- Colocar en la base de datos "INV_MP.dbf" la información referente a materias primas, subensambles (cuyas claves deben iniciar con "[") y productos o subensambles de maquila (con claves que inician con "^"). Al finalizar, se debe ordenar la base de datos por clave del artículo en orden ascendente. Por ejemplo:

Regist#	ARTICULO	EXIST	CA_PED	MIN	MAX	RED	DESCR	CLAVE	PRECIO_V	PRECIO_ULT	FECHA
1	ARO1	1400	0	1000	10000	3	ARO DE 5 cm. DIAM.	SIDEC	600.00	300.00	01/01/88
2	ARO2	3000	0	1000	8000	3	ARO DE 8 cm. DIAM.	SIDEC	600.00	300.00	01/01/88
3	ARO3	500	0	1000	7000	3	ARO DE 12 cm. DIAM.	SIDEC	800.00	400.00	01/01/88
4	BAFLE1	1300	0	1000	10000	3	BAFLE DE 5 cm. DIAM.	SIDEC	1400.00	700.00	01/01/88
5	BAFLE2	1800	0	1000	8000	3	BAFLE DE 8 cm. DIAM.	SIDEC	1800.00	900.00	01/01/88
6	BAFLE3	800	0	1000	5000	3	BAFLE DE 12 cm. DIAM	SIDEC	2000.00	1000.00	01/01/88
7	BRAZO	3000	0	5000	30000	3	BRAZO PARA SUS.	COXA	200.00	100.00	01/01/88
8	PANT1	800	0	1500	8000	2	PANTALLA CHICA	SIDEC	1400.00	700.00	01/01/88
9	PANT2	1350	0	1000	5000	2	PANTALLA GRANDE	SIDEC	1800.00	900.00	01/01/88
10	PUENTE	7000	0	10000	30000	3	PUENTE PARA SOQUET	SIDEC	600.00	150.00	01/01/88
11	RESORTE	3000	0	5000	20000	3	RESORTE PARA BRAZOS	COXA	300.00	150.00	01/01/88
12	RONDANA	5000	0	5000	30000	3	RONDANA UNIVERSAL	IMPTOR	100.00	55.00	01/01/88
13	SOQUET	150	0	5000	30000	3	SOQUET CERAMICO	ARA	1200.00	600.00	01/01/88
14	TOR1	3000	0	1000	30000	3	TORNILLO CHICO	IMPTOR	300.00	150.00	01/01/88
15	TOR2	5000	0	1000	30000	3	TORNILLO MEDIANO	IMPTOR	400.00	200.00	01/01/88
16	TOR3	500	0	1000	30000	3	TORNILLO GRANDE	IMPTOR	250.00	200.00	01/01/88
17	TUERCA	5000	0	5000	30000	3	TUERCA UNIVERSAL	IMPTOR	200.00	100.00	12/16/87
18	[UNI1	400	0	200	1000	2	UNIDAD PEQUEÑA	RIMA	3500.00	1250.00	01/01/88
19	[UNI2	30	0	400	2000	2	UNIDAD MEDIANA	RIMA	4000.00	2000.00	01/01/88
20	[UNI3	500	0	500	3000	2	UNIDAD GRANDE	RIMA	5000.00	2500.00	01/01/88
21	^UNI1B	0	0	100	1000	2	UNIDAD CH. BLANCA	PMP	3000.00	1500.00	01/01/88
22	^UNI1N	0	0	300	1000	2	UNIDAD CH. NEGRA	PMP	3000.00	1500.00	01/01/88
23	^UNI2B	0	0	400	1000	2	UNIDAD MED. BLANCA	PMP	5000.00	2500.00	01/01/88
24	^UNI2N	0	0	300	1500	2	UNIDAD MED. NEGRA	PMP	5000.00	2500.00	01/01/88
25	^UNI3B	0	0	500	3000	2	UNIDAD GR. BLANCA	PMP	5600.00	2800.00	01/01/88
26	^UNI3N	0	0	400	1500	2	UNIDAD GR. NEGRA	PMP	5600.00	2800.00	01/01/88

5.- Poner en la base de datos "INV_PT.dbf" la información de los productos terminados. Al finalizar, habrá que ordenar la base de datos por el campo denominado "ARTICULO". Por ejemplo:

Regist#	ARTICULO	EXIST	CA_DIS	CA_PED	MIN	MAX	RED	DESCR	PRECIO_V	PRECIO_ULT	FECHA
1	E15B-B	50	50	0	200	1000	2	EMPOTRADO 75W BLANCO	20000.00	7000.00	01/01/88
2	E15B-N	40	40	0	100	500	2	EMPOTRADO 75W NEGRO	20000.00	7000.00	01/01/88
3	E15BP-B	30	30	0	100	700	2	EMP/PANT 75W BLANCO	28000.00	11000.00	01/01/88
4	E15BP-N	54	54	0	100	1000	2	EMP/PANT 75W NEGRO	28000.00	11000.00	01/01/88
5	EG40-B	56	56	0	500	1000	2	EMP. 150W BLANCO	25000.00	10000.00	01/01/88
6	EG40-N	160	160	0	200	1000	2	EMP. 150W NEGRO	25000.00	10000.00	01/01/88
7	EGP-B	20	20	0	50	500	1	EMP/PANT 150W BLANCO	35000.00	13000.00	01/01/88
8	EGP-N	36	36	0	100	300	2	EMP/PANT 150W NEGRO	35000.00	13000.00	01/01/88
9	ES01-B	0	0	0	300	1000	2	EMPOTRADO 50W BLANCO	15000.00	5000.00	01/01/88
10	ES01-N	500	500	0	400	1000	2	EMPOTRADO 50W NEGRO	15000.00	5000.00	01/01/88

6.- En la base de datos "PROD.dbf" se deben colocar las descripciones para producción. En el campo "ARTICULO" se pone la clave del artículo que se va a producir (puede ser un subensamble de producción o un producto terminado) y en el campo "PARTES" se introducen las partes (pueden ser materias primas o subensambles de cualquier tipo) y las cantidades requeridas para su fabricación. AL finalizar, se debe ordenar la base de datos por el campo "ARTICULO". El formato que se debe seguir para usar el campo "PARTES" es:



Por ejemplo:

Regist#	ARTICULO	PARTES
1	E15B-B	^UNI1B-1 BRAZO-2 RESORTE-2 SOQUET-1 TOR1-2 RONDANA-2 TUERCA-2 *
2	E15B-N	^UNI1N-1 BRAZO-2 RESORTE-2 SOQUET-1 TOR1-2 RONDANA-2 TUERCA-2 *
3	E15BP-B	^UNI1B-1 PANT1-1 BRAZO-2 RESORTE-2 SOQUET-1 TOR1-2 RONDANA-2 TUERCA-2 *
4	E15BP-N	^UNI1N-1 PANT1-1 BRAZO-2 RESORTE-2 SOQUET-1 TOR1-2 RONDANA-2 TUERCA-2 *
5	EG40-B	^UNI2B-1 BRAZO-2 RESORTE-2 SOQUET-1 TOR2-2 RONDANA-2 TUERCA-2 *
6	EG40-N	^UNI2N-1 BRAZO-2 RESORTE-2 SOQUET-1 TOR2-2 RONDANA-2 TUERCA-2 *
7	EGP-B	^UNI2B-1 PANT2-1 BRAZO-2 RESORTE-2 SOQUET-1 TOR2-2 RONDANA-2 TUERCA-2 *
8	EGP-N	^UNI2N-1 PANT2-1 BRAZO-2 RESORTE-2 SOQUET-1 TOR2-2 RONDANA-2 TUERCA-2 *
9	ES01-B	^UNI3B-1 BRAZO-2 RESORTE-2 SOQUET-1 TOR3-2 RONDANA-2 TUERCA-2 *
10	ES01-N	^UNI3N-1 BRAZO-2 RESORTE-2 SOQUET-1 TOR3-2 RONDANA-2 TUERCA-2 *
11	[UNI1	PUENTE-1 ARO1-1 BAFLE1-1 *
12	[UNI2	PUENTE-1 ARO2-1 BAFLE2-1 *
13	[UNI3	PUENTE-1 ARO3-1 BAFLE3-1 *

7.- En forma análoga, se debe utilizar la base de datos "MAQ.dbf" para definir las partes para maquila. La diferencia con el paso anterior es que éste sólo puede contener en el campo "ARTICULO" claves de subensambles de maquila. La información se introduce en los diversos campos según su definición. Ejemplificación:

Regist#	ARTICULO	PARTES	PRECIO_ULT	FECHA
1	^UNI1B	[UNI1-1 *	500.00	01/01/88
2	^UNI1N	[UNI1-1 *	500.00	01/01/88
3	^UNI2B	[UNI2-1 *	500.00	01/01/88
4	^UNI2N	[UNI2-1 *	500.00	01/01/88
5	^UNI3B	[UNI3-1 *	500.00	01/01/88
6	^UNI3N	[UNI3-1 *	500.00	01/01/88

8.- Introducir en las bases de datos "DES_PRO.dbf" y "DES_DIS.dbf" la información referente a las descripciones de proveedores y distribuidores respectivamente.

9.- Dentro del directorio "FORMATOS" se deben definir los formatos de impresión. Esto se puede hacer por medio del editor de textos del Pascal u otro editor de textos o procesador de palabras, con la condición de que el fichero de disco esté en código ASCII. A continuación se listan dichos formatos:

```
F_P.TXT   - Formato para pedidos.
F_F.TXT   -      "       "   facturas.
F_E.TXT   -      "       "   comprobante de entrada.
F_EA.TXT  -      "       "   entrada al almacén.
F_S.TXT   -      "       "   comprobante de salida.
F_SA.TXT  -      "       "   salida del almacén.
F_OM.TXT  -      "       "   ordenes de maquila.
F_OP.TXT  -      "       "   ordenes de producción.
F_NC.TXT  -      "       "   notas de crédito.
```

En general, las claves utilizadas son distintas para cada uno de estos formatos, y se pueden encontrar en los listados del código fuente de las funciones CadenaExterna; éstas se localizan en los procedimientos que utilizan los formatos. A continuación se muestra un ejemplo de cada uno de ellos:

F_P.TXT

PEDIO	FOLIO: @e__
	FECHA: @g_____

NOMBRE: @m_____ TELEFONO: @o_____
DIRECCION: @n_____ C.P.: @r_____
CIUDAD: @p_____ ESTADO: @q_____
EMBARQUESE A: @s_____ FECHA DE ENTREGA: @j_____

NP	ARTIC	CANT	DESCRIPCION	PRECIO	IMPORTE
1	@P02__	@P03__	@P04_____	@P05_____	@P06_____
2	@P12__	@P13__	@P14_____	@P15_____	@P16_____
3	@P22__	@P23__	@P24_____	@P25_____	@P26_____
4	@P32__	@P33__	@P34_____	@P35_____	@P36_____
5	@P42__	@P43__	@P44_____	@P45_____	@P46_____
6	@P52__	@P53__	@P54_____	@P55_____	@P56_____
7	@P62__	@P63__	@P64_____	@P65_____	@P66_____
8	@P72__	@P73__	@P74_____	@P75_____	@P76_____
9	@P82__	@P83__	@P84_____	@P85_____	@P86_____
10	@P92__	@P93__	@P94_____	@P95_____	@P96_____

NOTAS:
@C_____
@C_____
@C_____

SUMA	@4_____
DESCUENTO @u %	@9_____
SUB-TOTAL	@a_____
I.V.A.	@3_____
TOTAL	@c_____

F_F.TXT

RIMA, S.A.
FABRICANTE DE LUMINARIOS

FACTURA @E_____	FOLIO: @^__
	FECHA: @ _____

FACTURADO A: @a_____ TELEFONO: @o_____
DIRECCION: @n_____ C.P.: @r_____
CIUDAD: @p_____ ESTADO: @q_____
EMBARQUESE A: @s_____ FECHA DE ENTREGA: @j_____

CONDICIONES DE PAGO @t_____	SU PEDIDO: FOLIO: @C_____ FECHA: @C_____	NUESTRO PEDIDO: FOLIO: @e__ FECHA: @g_____	OBSERVACIONES: @C_____ @C_____

@P	ARTIC	CANT	DESCRIPCION	PRECIO	IMPORTE
1	@P02__	@P03__	@P04_____	@P05_____	@P06_____
2	@P12__	@P13__	@P14_____	@P15_____	@P16_____
3	@P22__	@P23__	@P24_____	@P25_____	@P26_____
4	@P32__	@P33__	@P34_____	@P35_____	@P36_____
5	@P42__	@P43__	@P44_____	@P45_____	@P46_____
6	@P52__	@P53__	@P54_____	@P55_____	@P56_____
7	@P62__	@P63__	@P64_____	@P65_____	@P66_____
8	@P72__	@P73__	@P74_____	@P75_____	@P76_____
9	@P82__	@P83__	@P84_____	@P85_____	@P86_____
10	@P92__	@P93__	@P94_____	@P95_____	@P96_____

Debe(emos) y pagare(emos) incondicional-- mente el valor de ésta factura a la orden- de RIMA, S.A.

_____
FIRMA

SUMA	@0_____
DESCUENTO @u %	@9_____
SUB-TOTAL	@5_____
@I % I.V.A.	@U_____
TOTAL	@<_____

F_E.TXT

COMPROBANTE DE ENTRADA FOLIO: @e__

CLAVE: @h_____ FECHA: @j_____
ORIGEN: @f_ PEDIDO: @i__
RAZON: @g_

FECHA PARA PAGO: @k_____ FECHA DE PAGO: @l_____ PAGAR: $@m_____

PARTIDA	ARTICULO	CANTIDAD	PRECIO	IMPORTE
1	@P02___	@P03__	@P04_____	@P05_____
2	@P12___	@P13__	@P14_____	@P15_____
3	@P22___	@P23__	@P24_____	@P25_____
4	@P32___	@P33__	@P34_____	@P35_____
5	@P42___	@P43__	@P44_____	@P45_____
6	@P52___	@P53__	@P54_____	@P55_____
7	@P62___	@P63__	@P64_____	@P65_____
8	@P72___	@P73__	@P74_____	@P75_____
9	@P82___	@P83__	@P84_____	@P85_____
10	@P92___	@P93__	@P94_____	@P95_____
				_____
			SUMA:	@o_____

F_EA.TXT

ENTRADA AL ALMACEN FOLIO: @e__

FECHA: @j_____ PEDIDO: @i__

PARTIDA	ARTICULO	CANTIDAD
1	@P02___	@P03__
2	@P12___	@P13__
3	@P22___	@P23__
4	@P32___	@P33__
5	@P42___	@P43__
6	@P52___	@P53__
7	@P62___	@P63__
8	@P72___	@P73__
9	@P82___	@P83__
10	@P92___	@P93__

F_S.TXT

COMPROBANTE DE SALIDA FOLIO: @e__ @t_
FECHA: @j_____

CLAVE: @h_____
ORIGEN: @f_ PEDIDO: @i__
RAZON: @g_

PARTIDA	ARTICULO	CANTIDAD	PRECIO	IMPORTE
1	@P02___	@P03__	@P04_____	@P05_____
2	@P12___	@P13__	@P14_____	@P15_____
3	@P22___	@P23__	@P24_____	@P25_____
4	@P32___	@P33__	@P34_____	@P35_____
5	@P42___	@P43__	@P44_____	@P45_____
6	@P52___	@P53__	@P54_____	@P55_____
7	@P62___	@P63__	@P64_____	@P65_____
8	@P72___	@P73__	@P74_____	@P75_____
9	@P82___	@P83__	@P84_____	@P85_____
10	@P92___	@P93__	@P94_____	@P95_____
				_____
			SUMA:	@x_____

F_SA.TXT

SALIDA DEL ALMACEN FOLIO: @e__ @t_

FECHA: @j_____ PEDIDO: @i___

PARTIDA	ARTICULO	CANTIDAD
1	@P02___	@P03__
2	@P12___	@P13__
3	@P22___	@P23__
4	@P32___	@P33__
5	@P42___	@P43__
6	@P52___	@P53__
7	@P62___	@P63__
8	@P72___	@P73__
9	@P82___	@P83__
10	@P92___	@P93__

F_OM.TXT

ORDEN DE MAQUILA FOLIO: @e___
@E_____ FECHA: @j_____

PROVEEDOR: @n_____ TEL: @p_____
DIRECCION: @o_____ C.P.: @s____
CIUDAD: @q_____ ESTADO: @r_____

PARTIDA	ARTICULO	CANTIDAD
1	@P02__	@P03__
2	@P12__	@P13__
3	@P22__	@P23__
4	@P32__	@P33__
5	@P42__	@P43__
6	@P52__	@P53__
7	@P62__	@P63__
8	@P72__	@P73__
9	@P82__	@P83__
10	@P92__	@P93__

F_OP.TXT

ORDEN DE PRODUCCION FOLIO: @e___
@E_____ FECHA: @j_____

PARTIDA	ARTICULO	CANTIDAD
1	@P02__	@P03__
2	@P12__	@P13__
3	@P22__	@P23__
4	@P32__	@P33__
5	@P42__	@P43__
6	@P52__	@P53__
7	@P62__	@P63__
8	@P72__	@P73__
9	@P82__	@P83__
10	@P92__	@P93__

F_NC.TXT

NOTA DE CREDITO @E_____	FOLIO: @e__ FECHA: @f_____
Expedida por: R I M A, S. A. Registro federal de causantes: R I M-454345407 Cédula de Empadronamiento: 1144353 Dirección: Tel:	PEDIDO: @g__ FACTURA: @h__
A favor de: @i_____ Tel: @k_____ Dirección: @j_____ Ciudad: @l_____ Estado: @m_____ C.P.: @n__	
CON ESTA FECHA HEMOS ACREDITADO EN SU APRECIABLE CUENTA LO SIGUIENTE:	
@t_____ Importe @p % I.V.A. Total	@o_____ @q_____ @r_____
Concepto del crédito: @s_____	

Estos formatos se pueden diseñar libremente y modificar durante la operación del programa sin alterar su funcionamiento. Es indispensable que existan en el disco, por lo menos, todos los ficheros antes mencionados.

9.- El último paso consiste en utilizar el programa para lo cual, estando en el directorio "SIAC", se corre el programa de este mismo nombre. El programa se cargará en la memoria e iniciará la lectura de los datos del disco, lo cual tomará algún tiempo, según la cantidad de información contenida en las bases de datos. Es muy importante que se tome la opción "T" para salir del programa, ya que, de lo contrario, es probable que se pierda información valiosa.

ALGUNOS DATOS IMPORTANTES PARA EL MANEJO DEL PROGRAMA:

El programa presenta en la parte superior de la pantalla la sección específica en que se encuentra. Las posibilidades de elección se pueden ver en la pantalla de tres maneras:

1). Cuando la primera letra de cada posibilidad está

recalcada, indica que se debe oprimir la tecla correspondiente para tomar una opción.

2). El letrero "(S/N)" indica que se pueden tomar las opciones "S" (sí) o "N" (no).

3). El nombre de una tecla especial contenida entre los signos de menor que y mayor que indicará que opción se puede tomar. Por ejemplo: el letrero "<Esc>" indicará que es posible cancelar la orden con la tecla de escape.

Cuando se desea salir de alguna sección para regresar a la sección anterior, normalmente se presiona la tecla de escape. Algunas secciones no permiten el uso de dicha tecla porque en ellas se solicita la introducción de una clave, que, si se deja en blanco, dará el efecto de dicha tecla.

La llave del sistema puede ser puesta o removida desde cualquier sección del programa cuando se pide la elección de una opción. Para ello se deben presionar simultáneamente las teclas "Ctrl" y "L". Si el sistema no tenía puesta la llave, se pone la llave. Si el sistema tenía llave, se tienen dos oportunidades para introducir correctamente los caracteres que la componen, y así lograr quitar la llave hasta que se ponga de nuevo. De lo contrario, sonará una alarma para notificar el intento de violación al sistema.

5.2 CONCLUSIONES.

La computadora representa una herramienta de suma importancia para la administración eficiente de una empresa siempre y cuando se justifique su utilización. Aplicar dicha herramienta es una tarea laboriosa, pero una vez que se han logrado los objetivos planteados, su utilización permite llevar a cabo la administración de una manera mucho más sencilla y eficaz.

ETAPA DE DESARROLLO DEL PROGRAMA.

Una de las etapas más importantes en el desarrollo del sistema fue la planeación general del mismo. Una vez que se ésta se realizó correctamente, los pasos siguientes se facilitaron y se hizo posible alcanzar el resultado buscado.

Para el diseño de este sistema, fue necesario estar familiarizado con el manejo administrativo de la empresa. Una vez que se logró esto, se propusieron modificaciones para su posterior análisis. Esta fue una etapa iterativa con que se depuró la proposición de los nuevos seguimientos administrativos.

Esto no implica que siempre habrá que modificar los procedimientos administrativos originales de la empresa, si bien generalmente es necesario cuestionarse si éstos son los adecuados.

Los nuevos procedimientos administrativos se discutieron principalmente con el gerente (por ser éste quien mejor conoce el manejo global de la empresa) para después ser comentados y analizados con las personas que intervienen en cada proceso.

De este estudio conjunto se derivó un nuevo flujo de información a partir del cual se desarrolló la etapa de programación. En ésta se utilizó a fondo el lenguaje "Turbo Pascal", y se crearon e implementaron ciertas rutinas y "trucos" para mejorar la presentación y el funcionamiento del programa. A la vez, se constató que el lenguaje Pascal es sumamente adecuado para el desarrollo de grandes proyectos (o programas) pues, como es un lenguaje estructurado permite dividir las tareas grandes en otras más pequeñas con cual permite una gran claridad y legibilidad que permiten una depuración rápida del mismo. Por otra parte, los procedimientos de solapamiento permitieron aumentar el código objeto del programa en forma prácticamente ilimitada, sin lo cual no se habría podido obtener un programa tan grande, y, a la vez, con tan poco requerimiento de memoria RAM.

El análisis del dBase III mostró, además, que su uso es relativamente sencillo y muy potente, por lo que permitiría una altísima capacidad de manipulación de la información. La aplicación de dicho programa ha resultado bastante satisfactorio y, en general, ha dado solución a todas las nuevas implementaciones del programa.

ETAPA DE INSTALACION Y OPERACION DEL PROGRAMA.

En la etapa de instalación del programa, lo que resultó más extenso y laborioso fue la preparación de las bases de datos, pues se necesitó asignar claves a los artículos y definir las partes necesarias para producción y maquila.

Una vez que el sistema estuvo listo y revisado se procedió a la operación piloto del programa; para ello, se explicó a la persona a cargo de la computadora la forma de utilizarlo, lo cual fue relativamente fácil ya que el programa guía al usuario en todas sus secciones.

Durante la utilización del programa se corroboró que un factor predominante para la implementación exitosa del sistema es el factor humano. En muchas personas existe cierto grado de resistencia a utilizar la computadora. Esta actitud se fue modificando con persuación y capacitación. Como se ha dicho en

varias ocasiones, la computadora es tan solo una herramienta del hombre, por lo que no funcionará si éste no está dispuesto o capacitado para utilizarla correctamente.

El sistema que se desarrolló cumplió totalmente con los objetivos planteados. Durante su aplicación se fueron identificando nuevas necesidades a satisfacer y la importancia de hacer ciertas modificaciones en algunos de los procedimientos ya elaborados.

Los resultados obtenidos fueron muy positivos, pues permitieron a la empresa tener un control bastante preciso de sus principales áreas de manejo administrativo, con una inversión relativamente baja. El gerente se liberó de muchas tareas rutinarias y pudo dedicar más tiempo a la toma de decisiones, ahora basada en información actualizada y veraz. La persona encargada de atender a los clientes, mejoró sustancialmente sus servicios al poder proporcionar rápidamente información de vital importancia para ventas y así logró incrementar su productividad.

Los problemas de cobranza fueron prácticamente erradicados con el manejo adecuado del sistema ya que con éste se pudo reglamentar lo relativo a cobranza.

La sección de órdenes de maquila resultó de mucha utilidad, tanto para los proveedores como para la administración, y el control de los pagos a proveedores se ha podido llevar a cabo con eficiencia y exactitud.

El encargado de producción ha podido planear sus lotes con mayor anticipación, y la falla en el suministro de materiales ha sido menor ya que, con la implementación del programa ésta sólo obedece a problemas del proveedor. El bodeguero ya no tiene que llevar el control del inventario por medio de tarjetas, sino sólamente atender la entrega de lo que indican las hojas para salidas de bodega y verificar que se recibe lo que indican las hojas para entradas a bodega. Dicha simplificación en el manejo de inventarios ha producido una considerable mejora en su control, aun cuando pueden subsistir algunas discrepancias ente las existencias reales y las que indica la computadora, debido principalmente a errores humanos por parte del bodeguero (y algunos por parte del capturista).

Existen aún muchas ventajas que no se exponen, y otras más que se podrían encontrar con nuevas implementaciones al sistema.

NUEVAS IMPLEMENTACIONES.

El sistema planteado en esta tesis no pretende solucionar todo el control administrativo de una empresa manufacturera sino la parte medular del mismo. La solución global de dicha

administración es una meta factible, que se puede alcanzar mediante nuevas implementaciones al sistema y tal vez algunas modificaciones al programa propuesto.

En general, las nuevas rutinas se desarrollaron por medio del dBase III, ya que el mantenimiento de las bases de datos se hace con gran facilidad, los reportes se elaboran en cuestión de minutos y la transferencia de datos hacia la mayoría de los dispositivos existentes y de los principales programas es casi directa.

Las rutinas de mantenimiento de archivos se realizaron por medio de programas muy sencillos en dBase III. Uno de ellos es el de envío de registros del primer mes del que tiene control la computadora a discos flexibles (para archivo), y su eliminación del disco duro. El uso de esta rutina es de suma importancia, ya que la capacidad de almacenamiento del disco y del programa principal es limitada.

También se realizó un programa en dBase III que obtiene una lista de precios condensada utilizando la base de datos de productos terminados. Esta lista de precios se puede imprimir como un reporte o se puede transferir de varias maneras muy sencillas a un procesador de texto.

Un proyecto interesante sería el de desarrollar un programa que obtuviera los datos históricos de las ventas y, a partir de éstos, que calculara las existencias mínimas y máximas recomendables de productos terminados.

Estando consciente de que lo óptimo es una meta a la que nunca se llega y de que, sin embargo, el progreso se finca en el intento reiterado por llegar a ella, puedo concluir que aún queda mucho por hacer en el continuo perfeccionamiento del sistema que en esta tesis se presenta.

BIBLIOGRAFIA

Byres, Robert A. Introducción a las Bases de Datos con dBase III Plus, 1a edición, (España: Osborne/McGraw-Hill, 1987).

Graham, Lyne J. IBM/PC. Guía del Usuario, 1a edición, (México: Osborne/McGraw-Hill, 1985).

Holffman, Paul y Tamara Nicoloff. Sistema Operativo MS-DOS. Guía del Usuario, 1a edición, (México: Osborne/McGraw-Hill, 1985).

Jones, Edward. Aplique el dBASE III, 1a edición, (México: Osborne/McGraw-Hill, 1986).

Jones, Edward. Aplique el dBase III Plus. 1a edición, (México: Osborne/McGraw-Hill, 1987).

Sand, Paul A. Pascal Avanzado. Técnicas de Programación, 1a edición, (México: Osborne/McGraw-Hill, 1986).

Shneider, G. Michael. Introducción a la Programación y solución de Problemas con Pascal, 1a edición (México: LIMUSA, 1986).

V. Farina, Mario. Diagramas de Flujo, 19a impresión, (México: Editorial Diana, 1987).

Wood, Steve. Turbo Pascal. Versión 3.0, 1a edición, (España: Osborne/McGraw-Hill, 1986).