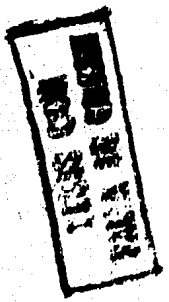
UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE INGENIERIA

HOJA ELECTRONICA DE CALCULO

DESARROLLO DE UNA HOJA ELECTRONICA DE CALCULO PROGRAMADA EN LENGUAJE C.



T E S I S

Que para obtener el Título de
INGENIERO EN COMPUTACION
p r e s e n t a n

Hilario Sánchez Matamoros
Alejandro Espindola Navarrete

Director de Tesis: Fis. Raymundo H. Rangel G.

México, D. F. 1990

UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

INDICE TEMATICO

1. INTRODUCCION

1.1 QUE ES UNA HOJA ELECTRONICA

El 90% de los cálculos que se efectuan en una oficina son relativamente sencillos, emplean fórmulas simples que involucran operaciones aritméticas fundamentales y generalmente se presentan en forma de tablas.

Utilizando un programa de hoja electrónica, se puede calcular una página entera de cifras y recalcularla si alteramos alguno de sus elementos. Además cuentan con otras características que complementan la eficacia y hacen verdaderamente útiles a estos programas.

LA HOJA ELECTRONICA DE CALCULO

Esta herramienta es la transposición de dos hojas de papel, una visible y otra invisible de un tamaño prácticamente ilimitado en la pantalla de la computadora. Los elementos que conforman la hoja se manipulan por su posición en la tabla, es decir, por su columna y su renglón y en algunos casos a través de nombres simbólicos (variables). En la hoja visible, el usuario puede escribir etiquetas, comentarios, líneas y cifras, tal y como lo haría, por ejemplo, para tabular algún problema, asociando un elemento a cada celda. En la hoja invisible se encuentran las fórmulas que ligan los diferentes datos de la hoja visible. Así al cambiar alguna información en la hoja visible, gracias a las fórmulas de la hoja invisible, se calculan automáticamente los otros datos de la hoja visible; por eso se le llama también hoja para modelos de tipo, **si-entonces**, ya que permite, sin programar, hacer un ensayo en los datos o fórmulas (**si**) y observar el resultado (**entonces**).

El mercado del **software** para microordenadores reacciona con mucha rapidez ante el éxito obtenido por uno de sus productos y las hojas no son la excepción. En la actualidad ya se encuentran a la venta, programas de "Hojas Electrónicas" para cualquier clase de máquina, sean apropiados o nó.

Por lo anterior dicho,no hay que sorprenderse de que los paquetes de hojas electrónicas sean el tipo de programas que más se vendan en todo el mundo. En la siguiente tabla se listan algunas de las aplicaciones con hojas electrónicas:

- Revisión de ingresos
- Gestión de una cartera de valores
- Facturación
- Simulación
- Análisis de adquisición y consolidaciones
- Coste del trabajo
- Relaciones de impuestos y depreciaciones
- Previsión de ventas
- Análisis de estado real y estimación de precios

1.2 LA INDUSTRIA DEL SOFTWARE EN MÉXICO

No cabe duda que al abordar el tema de la industria del software en México nos encontramos ante serias dificultades, ya que según numerosas opiniones de un grupo de destacados especialistas mexicanos en el área del software, un alto porcentaje afirmaron "La industria del software en México prácticamente no existe".

No nos centraremos en discutir si la mencionada industria existe o no, nos centraremos en un punto mucho más importante aún, el tratar de demostrar no solo la conveniencia, sino la urgente necesidad y la enorme posibilidad de que la industria del software en México se desarrolle ampliamente a corto plazo.

Cabe mencionar que en un tema tan novel y tan importante, no existen todavía reglas de aplicación general que permitan estructurar un plan de acción para alcanzar la meta sino más bien opiniones y puntos de vista para reflexionar, de cuyo estudio y análisis emanarán criterios y reglas tan necesarios para el desarrollo exitoso de la empresa que se inicia. Por esto, solo nos limitaremos a citar los principales puntos alrededor de los cuales se debe reflexionar, de manera que puedan utilizarse como guía para descartar algunos y profundizar en otros y necesariamente sacar a luz nuevos hechos y circunstancias que enriquezcan el presente trabajo.

IMPORTANCIA ACTUAL DEL SOFTWARE EN EL MUNDO

El **software** ha adquirido una importancia especial en los últimos años debido a factores como:

a) Al papel que jugará en los próximos años en el ámbito de la informática.

b) Debido a que dentro de un sistema informático, es el **software** quien proporciona principalmente el valor agregado a los datos, convirtiendolos en información, la cual puede adquirir entonces el valor estratégico para la toma de decisiones. Es también el software quien proporciona la inteligencia agregada a un sistema informático. (Estrictamente, el hardware únicamente proporciona la oportunidad, a través de las velocidades de proceso y acceso a los datos, el cual, evidentemente, puede también tomar un alto valor estratégico).

IMPORTANCIA DEL SOFTWARE EN MEXICO

En forma especial, el software en México tiene una importancia adicional debido a que se han dado algunas otras circunstancias:

a) Debido a que el software solamente requiere, estrictamente, de lápiz, papel y talento. No hay inversión de capital. El hardware para las pruebas e implementación, es una necesidad inferior en varias órdenes de magnitud. Además con el hardware instalado actualmente en México, se permite contar con un acceso prácticamente ilimitado a hardware altamente representativo, del más comúnmente instalado en el resto del mundo.

b) Debido a que el software es, proporcionalmente, cada vez más costoso que el hardware.

El desarrollo de la tecnología en materia de electrónica permite generar productos más confiables y a menor costo.

Por el contrario, la confiabilidad del software con frecuencia deja mucho que desear y es cada vez más costoso.

Esta situación es la que dió origen a la difusión de la programación estructurada, desarrollo descendente y en general toda la revolución de las técnicas estructuradas de la década de los setentas, es decir a los antecedentes inmediatos de la Ingeniería del Software.

Siendo en buena medida producto de importación, y contemplado desde el punto de vista de posibles desarrollos nacionales, es cada día más atractivo exportarlo. Para el caso del hardware no sucede necesariamente lo mismo, debido a la competitividad existente en el mercado del hardware, es más difícil ofrecer al mercado internacional un desarrollo nacional competitivo tecnológica y económicamente.

c) Debido a la cantidad de Micros ya existentes y a las que se estima existirán en los próximos años. Un software totalmente adecuado es necesario para el aprovechamiento de las microcomputadoras.

d) La ventaja adquirida al contar con técnicos nacionales especialistas en las técnicas de desarrollo de software.

Una de las técnicas para el desarrollo de software más

conocidas actualmente y de mayor actualidad, es la programación estructurada, basada en un teorema que guarda una analogía casi total con el teorema postulado para el diseño de circuitos lógicos.

En la actualidad, se ha logrado que la actividad de diseño de circuitos (hardware) pueda empezar a verse como una actividad de programación (software). La construcción de circuitos puede considerarse como el ensamble de componentes electrónicos sencillos y complejos (Chip's), para conseguir un resultado, tal y como un programa se compone de instrucciones sencillas y complejas (Rutinas).

Por lo anterior, el estudio y dominio de los principios modernos para la construcción de programas y sistemas (modularidad, cohesión, acoplamiento, etc.) son de gran utilidad aún para el área de hardware.

e) Debido a la importancia que tiene el reconocer al software como un integrante natural de una computadora.

En los últimos años se ha impulsado en México a la industria de la computación y en especial la de las microcomputadoras. Se ha obligado a fabricantes nacionales a ciertos porcentajes de integración nacional del equipo que produce, pero toda esta integración se ha enfocado al hardware. Sin embargo prácticamente todo el software básico sigue siendo de importación. (Convendría obligarlos también integrar al menos parcialmente su software con partes nacionales; algunas rutinas del sistema operativo para manejo de los caracteres en español, o rutinas traducidas "help", etc.).

PROBLEMATICA ACTUAL EN EL USO Y DESARROLLO DE SOFTWARE EN MEXICO

En la actualidad se distinguen muchos y muy diversos problemas relacionados con el uso del software de importación y en

el desarrollo de software nacional, los cuales van desde los intracendentes hasta los graves y desde los eventuales hasta los sistemáticos. A continuación se presentan los que se consideran más importantes.

1) Los manuales de usuario y de referencia se encuentran en inglés. Algunos usuarios producen bibliografía complementaria, pero también en Inglés. Cuando la bibliografía incluye ejemplos, éstos estan hechos para circunstancias y situaciones específicas del país de origen. (Sistema Escolar, Comercio, Etc.).

Existe una salida de divisas. La operación del software en la máquina se encuentra en inglés. En caso de que exista una traducción al español, o si se trata de un producto desarrollado en otro país de habla hispana, la terminología técnica que se utiliza es diferente (fichero, octeto, perla, etc.).

No existe en México un soporte de servicio equivalente (capacitación, errores, consulta de dudas por teléfono, etc.). En general los representantes en México no son expertos en el software que venden.

2) No existen técnicos suficientemente capacitados para adaptarlo a las necesidades específicas de la instalación (uso óptimo del hardware, nuevas funciones, traducciones) o para corregirla en caso de algun error.

Existe muy poca investigación en centros de educación superior en este tema. No existe una formación de recursos humanos en las universidades orientada hacia estos temas .

3) Los paquetes no siempre se encuentran adaptados a las necesidades del usuario nacional. En general no se cuenta con los caracteres en español. Algunos paquetes cuentan con acentos y con la letra ñ, sin embargo aparentemente se olvidan de que también es necesario incluir los diéresis, signo de apertura de interrogación y de apertura de admiración.

4) Actualmente resulta prácticamente prohibitivo adquirir software de aplicación, por su altísimo costo. Este costo no sólo

se debe a la relación de paridad del peso, sino que a diferencia de los paquetes en los que el fabricante los puede exportar tal y como se venden en el mercado doméstico, en las aplicaciones el fabricante requiere hacer modificaciones y adaptaciones que en muchos casos implican un gran esfuerzo, lo que aumenta aún más el costo. Adicionalmente, debido al tiempo que le toma al fabricante efectuar estos cambios, y posiblemente publicar una traducción de los instructivos al español, cuando finalmente se tiene la aplicación ésta ya no es la última versión del fabricante, por lo que no siempre se cuenta con el mejor producto existente. En general la compra de aplicaciones de importación está limitada a las grandes empresas transnacionales que por su compatibilidad con su casa matriz requieran mantener el mismo software.

5) En general el software es demasiado dependiente del hardware, el cual no se fabrica en México, lo que hace al mercado demasiado limitado y especializado. No se cuenta con experiencia previa para iniciar este tipo de proyectos. (Se podría iniciar a ganar experiencia colaborando con el fabricante de hardware con los planes de integración nacional).

6) Al no haber un respeto absoluto a los derechos de autor, el esfuerzo necesario no resulta costeable. No hay todo el hardware ni todo el software para desarrollar productos competitivos a nivel internacional (IBM-PC, UNIX).

En los casos que existe, no se está ofreciendo el servicio post-venta tan necesario en la comercialización de software. La comercialización resulta muy costosa. No existe soporte financiero para la comercialización y aún menos para el desarrollo de proyectos de software, (hace falta una difusión en las organizaciones, de los conceptos básicos de la informática). Los paquetes de importación resultan en general más económicos (debido entre otras cosas, al volumen de ventas). No existen recursos humanos especializados en la comercialización de software. En general no hay una cultura informática suficiente entre los

usuarios para valorar el costo del software (copias ilegales es más la regla que la excepción, sin saber, quienes las hacen, del daño tan grande que producen directamente al país). Los compradores de software no tienen muy claro que significa comprar software (se piensa que el costo del software es el costo del medio en el que está grabado y quizá el de los instructivos, y no se piensa en el servicio de respaldo, actualización, capacitación, etc.).

Algunas casas se software extranjeras han iniciado el desarrollo de software de paquetería para exportación con sus propios recursos, lo que implica qué llegará a generalizarse la salida de divisas, la no transferencia de tecnología ni de cultura informática asociada, problemas de terminología técnica, etc.

7) Las aplicaciones se dedican principalmente a las actividades de apoyo administrativo y no a las sustantivas de cada instalación. Además no son suficientemente conocidadas las tecnicas más modernas para el desarrollo de software.

SITUACIONES QUE PREVALECEN EN NUESTRO PAIS

Después de haber presentado la problemática existente, a primera vista parecería que la cantidad de dificultades existentes hacen imposible casi cualquier intento por resolverla, sin embargo, a continuación se presenta un conjunto de factores que demuestran, no sólo la conveniencia y la necesidad, sino la posibilidad real que existe de mejorar la situación que prevalece actualmente.

1. El desarrollo de software requiere, estrictamente sólo de ingenio, no hay inversión de capital, por lo que puede darse en México, aúnante la situación económica actual.

2. Se ha mencionado que la actividad de desarrollo de software en México se asemeja a la etapa del desarrollo humano de

la agricultura para el autoconsumo.

Cada empresa u organización desarrolla el software que requiere con sus propios medios (recursos humanos, materiales, financieros), adaptado a sus propias necesidades para su uso interno exclusivamente.

3. Existe un mercado potencial de microcomputadoras tanto nacional como en el mundo de habla hispana y en el mundo entero, requisito necesario y a la vez prácticamente suficiente para el desarrollo de una industria.

4. Existen en México los recursos humanos capaces de emprender proyectos de software a mediana y gran escala, tanto formados por las universidades del país como con estudios de posgrado en el extranjero.

5. Se sabe de antecedentes de que los mexicanos son capaces de desarrollar software de calidad y competitivo a nivel internacional. Para esto bastan algunos ejemplos como el desarrollo del paquete LCX (quizá el primer lenguaje de cuarta generación que existió en el mundo), el cual hace quince años fue desarrollado en nuestro país por un mexicano, quien actualmente lo comercializa tanto en México como en los Estados Unidos.

Existen casos similares de programas desarrollados por profesores y alumnos de las universidades, así como desarrollos recientes de casas de software mexicanas.

6. La ubicación geográfica de México junto a la primera potencia mundial en fabricación de hardware y software no le permite mantenerse al margen de los avances tecnológicos, y en paricular del fenómeno informático de los Estados Unidos, lo que mantiene a los técnicos interesados en la materia, informados y actualizados a un nivel de alta competitividad.

7. Los paises industrializados no estan interesados en desarrollar el software que requieren los paises en desarrollo ya que: su mercado local es suficientemente grande, el esfuerzo que representa adaptar y traducir su software a otras necesidades y

además, a que no se sienten suficientemente respaldados en sus derechos de autor en las leyes de los paises en desarrollo.

8. La política de descentralización de la Administración Pública Federal abre un gran mercado a la adquisición de Micros y por lo tanto a la necesidad de software estandarizado para el control de recursos materiales, financieros y humanos, el cual debe ser exactactamente igual en cada microcomputadora, abriendo el mercado de paquetería especializada de uso general.

ACCIONES QUE SE SUGIEREN PARA RESOLVER LA REFERIDA PROBLEMÁTICA

El siguiente conjunto de acciones ofrece soluciones al problema actual del software en México, aprovechando las circunstancias prevalecientes en este momento.

1. Se requiere definir una estrategia mexicana de desarrollo informático, en la cual se contemplen las posibilidades reales de desarrollo en las diferentes áreas de la informática y dentro de la cual se reconozca el valor de promover la industria del software en México tanto para consumo nacional como para producto de exportación.

2. Integrar a los programas de estudio de las universidades, tópicos relacionados con la industria del software tales como: desarrollo de software de paquetes (Hojas de cálculo, Procesadores de textos, Paquetes estadísticos, etc.); software de complemento (Manejadores de bases de datos, telecomunicaciones, etc.), comercialización de software; financiamiento de proyectos de software, etc.

3. Estrechar las relaciones Universidad-Industria con el doble propósito, por una parte incorporar a los egresados de informática y computación de las universidades al desarrollo industrial de software y por otra mantener actualizados a los

productores de software con las técnicas más eficaces para el desarrollo y comercialización de sus proyectos.

4. Promover el uso de la informática en las actividades sustantivas de la empresa u organismos, lo que llevaría a: i) Ampliar el mercado de demandantes y ii) Aprovechar más integralmente la capacidad de cómputo instalada.

5. Promover la difusión de la cultura informática donde se reconozca el valor real del software.

6. Difundir los beneficios últimos del uso del software como herramienta para la toma de decisiones y no solamente la industria del software por la industria misma.

7. Promover ampliamente el desarrollo de la "Industria del Software" aún antes que la del hardware, ya que aquí es donde podemos ser más competitivos.

8. Seguir importando software para crear una industria actualizada y a la vez competente. Un cierre en las fronteras para la entrada de software crearía un mercado cautivo que definitivamente restaría calidad y competitividad a los productos mencionados.

9. Permitir y promover la entrada de tecnología de punta tanto en Hardware como en Software.

10. Fomentar el desarrollo de casas de software nacionales mediante estímulos fiscales y similares, pero nunca con el cierre de fronteras.

11. Promover el comercio intra-industrial en materia de software con el resto de paises de habla hispana.

1.3 EL LENGUAJE C

Cuando se definió el proyecto, se tomaron en cuenta tres aspectos importantes para ello. El primero fue el uso de un conjunto de instrucciones del lenguaje **C**, para reducir al mínimo las incompatibilidades. El segundo aspecto fue utilizar el más rápido y eficiente compilador de **C** disposible y el aspecto final fue promover un integrado ambiente de programación **C**, para hacer más dinámico el desarrollo de programas. Por todas estas características se llegó a la conclusión que lo ideal sería utilizar TURBO-C.

C ES UN LENGUAJE DE MEDIANO NIVEL

C a menudo se le conoce como un lenguaje de mediano nivel de computadoras. Esto no significa que el lenguaje **C** sea de menor poder, de difícil uso o menos desarrollado que un lenguaje de alto nivel como BASIC o PASCAL. Ni esto implica que **C** sea similar al lenguaje ensamblador, sino que esta visto como un lenguaje de mediano nivel por la combinación de elementos de lenguajes de alto nivel con funciones de ensamblador.

La siguiente tabla muestra como el lenguaje **C** queda en el espectro de los lenguajes de programación.

Alto nivel	Ada Modula-2 Pascal Cobol Fortran Basic
Mediano nivel	C Forth
Bajo nivel	Macro-ensambladores Ensambladores.

El código del lenguaje **C** es muy transportable. Lo que significa que se puede adaptar el software escrito para un tipo de computador a cualquier otro tipo de computador. Por ejemplo, si se puede llevar fácilmente un programa escrito para una APPLE II+ a una IBM-PC, entonces ese programa es transportable. Además el soporte de TURBO-C, asegura la transportabilidad de sus códigos a otros ambientes.

Todos los lenguajes de alto nivel tienen el concepto de "Tipo de dato" (Un tipo de dato define un conjunto de valores que una variable puede almacenar y que la computadora por medio de un conjunto de operaciones puede ejecutar bajo el nombre de esa variable), tipos de datos comunes que son; enteros, caracter y real. y aunque **C** tiene cinco construcciones básicas para tipos de datos, no son comparables a las de los lenguajes como PASCAL o ADA, sin embargo **C** permite la conversión de casi todos los tipos de datos de cualquiera de los lenguajes de alto nivel. Por ejemplo, se pueden mezclar libremente caracteres y enteros en la mayor parte de las expresiones. En general, los compiladores **C**, ejecutan pequeñas corridas con chequeo de errores, checando el límite de arreglos o verificando la compatibilidad de los tipos. Esto es válido también en TURBO-C, estas verificaciones estan bajo la responsabilidad de los programadores, aunque en algunos compiladores no existe el la verificación dado que durante la ejecución de un programa, la verificación hace a esta que sea lenta, por lo tanto, el programador debe decidir si es necesario o no ejecutar su programa con algún verificador.

Como un lenguaje de mediano nivel, el lenguaje **C** permite la manipulación de bits, bytes y direcciones, los cuales son los elementos básicos para hacer funciones que manejen el computador. Estan habilidades hacen que **C** sea apropiado para la programación de sistemas, donde estas operaciones son comunes.

Otro aspecto importante del lenguaje **C** de TURBO-C es que tiene solamente 38 comandos (32 estan definidas conforme al

acuerdo de estandarización de ANSI y cinco fueron adicionadas por su diseñador para poder hacer mejor uso de algunos aspectos especiales del medio ambiente de la PC). Esta cantidad de comandos son los que distinguen al lenguaje **C**, para hacer una comparación, el BASIC para la IBM-PC tiene 159 comandos.

C ES UN LENGUAJE ESTRUCTURADO

Aunque el término lenguaje estructurado no es aplicable estrictamente al lenguaje **C** en un sentido académico,el lenguaje C es comúnmente referido como un lenguaje estructurado porque tiene estructuras similares a las de ALGOL, PASCAL y MODULA-2.

Técnicamente, un lenguaje estructurado permite declarar subrutinas dentro de otras subrutinas y ya que el lenguaje **C** no permite ésto, no puede estrictamente ser llamado lenguaje estructurado.

Aquí hay algunos ejemplos de lenguajes estructurados y no estructurados:

No estructurados	**Estructurados**
FORTRAN	PASCAL
BASIC	ADA
COBOL	C
	MODULA-2

UN REEMPLAZO DE LENGUAJE ENSAMBLADOR

El lenguaje **C** tiene la habilidad de operar directamente sobre los bits y los bytes de la memoria, esto contribuye a la popularidad de C entre los programadores. Aunque el lenguaje ensamblador da a los programadores potencial para realizar tareas con la máxima flexibilidad y eficiencia, el lenguaje ensamblador es notoriamente difícil de trabajar, cuando se desarrollan y depuran programas. Además ya que el ensamblador es no estructurado

por naturaleza, el programa final tiende a ser un spaghetti de código, un lío de saltos, llamadas e indexaciones.

Esta característica hace de los programas en lenguaje esnsamblador, difíciles de leer, de implementar modificar. Quizá lo más importante, las rutinas del lenguaje ensamblador no son transportables entre máquinas que tienen diferentes CPUs.

Todo esto implica que a pesar de todos los defectos del lenguaje ensamblador, frecuentemente es usado, porque ofrece el único medio para producir programas que corren demasiado rápido.

PARA QUE PUEDE SER USADO C

Inicialemente le lenguaje **C** fue usado para la programación de sistemas. Los programas de sistemas son parte de una larga clasificación de programas que forman parte del sistema operativo de una computadora o son utilería de soporte.

Por ejemplo los siguientes son comúnmente llamados programas de sistemas:

- **Sistemas operativos**
- **Ensambladores**
- **Interpretes**
- **Compiladores**
- **Editores**
- **Manejadores de bases de datos**

Como el lenguaje **C** creció en popularidad, cualquier programador puede empezar a usarlo en cualquier tipo de tareas, porque tiene gran portabilidad y eficiencia. Y sin duda llegará a ser el mejor lenguaje de programación de propósito general.

2 REVISION DE UNA HOJA ELECTRONICA ACTUAL

2.1 Características y funcionamiento del 1-2-3 de LOTUS

Lotus **1-2-3** se ha convertido en uno de los programas más importantes y populares en el mundo de las microcomputadoras . Integra en un solo programa, Análisis de Hojas Electrónicas, Tratamiento de Información y Gráficos. A pesar de contar con más de 110 órdenes y más de 40 funciones, Lotus **1-2-3** no es un programa complicado de utilizar.

La primera función del **1-2-3**, análisis de hojas electrónicas, aplica la memoria y rapidez de la computadora a problemas que se resolverían manualmente con: papel, lápiz, calculadora y numerosas fórmulas. La hoja de trabajo proporciona 2048 renglones y 256 columnas, muchas más de las proporcionadas por sus predecesoras.

Un aspecto importante que se requiere para la utilización del programa es la cantidad de memoria instalada en su computadora. Ya que para el funcionamiento del **1-2-3** es necesario un mínimo de 192K RAM. El resto de memoria se utiliza para almacenar los datos y fórmulas de su hoja de trabajo.

La hoja de trabajo del **1-2-3** contiene datos en forma de palabras, números y fórmulas. Incluye más de 40 funciones que realizan una variedad de cálculos financieros, estadísticos, con fechas, lógicos y matemáticos. Además, el **1-2-3** tiene una gran capacidad de formatos para controlar la presentación de la hoja y sus opciones de impresión proporcionan un control total sobre los informes.

La estructura de la hoja de trabajo nos lleva, por sí sola, a una segunda función de toma de decisiones: la gestión de información, ya que **1-2-3** organiza una base de datos considerando a los datos contenidos en una fila como registros de la base y a las columnas como los campos.

Para Lotus **1-2-3** ya no hay una delimitacion clara entre hoja electrónica y base de datos. Todo el mundo quiere organizar la

información contenida en la hoja de trabajo. **1-2-3** puede clasificar una hoja de trabajo alfabéticamente o numéricamente en sentido ascendente o descendente (**1-2-3** le permite especificar dos campos para una única operación de clasificación). También puede hacer que **1-2-3** localice o extraiga datos de la hoja de trabajo; y a pesar de que su capacidad como base de datos no es comparable a aquellas como las que posee DBASE estas son muy útiles en conjunción con las otras dos.

La tercera función importante del **1-2-3** son los gráficos. Los sofisticados comandos de graficación de **1-2-3** permiten crear gráficas de hasta cuatro variables usando la información contenida en la hoja de cálculo.

1-2-3 nos da la posibilidad de elegir entre cinco tipos de gráficas, incluyendo las gráficas de barra y lineal, de pastel (de una sola variable), de barras adosadas y las gráficas X-Y (con dos listas de variables usadas como pares de coordenadas X-Y).

Es más, una vez que usted haya hecho una gráfica, solo es necesario pulsar tres teclas para desplegarla en otra forma; y si cambia los datos ya representados, solamente necesita pulsar una tecla para ver el gráfico modificado en la pantalla.

Exiten además otras varias opciones que nos permiten cambiar la presentación de la gráfica, así, si usted esta utilizando un monitor de color, puede aprovechar su pantalla para que **1-2-3** despliege cada rango de los datos en diferente color.

Es así que la integración es una muy importante característica del **1-2-3**.

Ya que los tres programas de Lotus : hoja de cálculo, base de datos y graficación estan en la memoria de la computadora simultáneamente (**1-2-3** no usa overlays para traer una sección de código cuando es necesitada).

En adición a su poderio y fácil uso, hablemos de otra característica la cual tiene un gran potencial: la utilidad del **MACRO** como la alternativa al tecleo.

Las aplicaciones de las hojas electrónicas a menudo conllevan repeticiones. Por ejemplo, es raro que un pronóstico financiero sea correcto la primera vez. Incluso si las suposiciones con correctas, es deseable probar varios supuestos o escenarios, antes de guardar el pronóstico.

Esto llega a ser aburrido por dos razones. Primero, resulta monótono introducir repetidamente las mismas órdenes. Segundo, si el módelo no se ejecuta a menudo; es fácil olvidar las órdenes involucradas, así como los nombres de las hojas de trabajo que deben ser combinadas. Afortunadamente, **1-2-3** ofrece su "alternativa al tecleo"; la utilidad **MACRO**.

Una macro es una columna de celdas que almacenan los caracteres de las órdenes del **1-2-3** y las entradas de datos como si las hubiera tecleado usted mismo. Asigne a la macro un nombre único; cuando se llama a ese nombre se ejecutan los contenidos de la **MACRO**. En otras palabras, almacenando un grupo de órdenes del **1-2-3** en un rango de celdas, puede ordenar al **1-2-3** que ejecute el grupo, simplemente introduciendo su nombre.

L O S C O M A N D O S

Utilizamos los comandos de **1-2-3** para trabajar con la información almacenada en las celdas, como también para cambiar la representación de la hoja. Por ejemplo, con los comandos de **1-2-3** podemos borrar información o moverla a otra parte de la hoja. Podemos también insertar o eliminar filas, o columnas, o cambiar el ancho de las columnas de la hoja.

Cuando tecleamos el slash (/), los comandos de **1-2-3** aparecen en una lista horizontal en el panel de control. Esta lista de comandos es llamada el menú. En ocasiones aparecen señalamientos de pantalla debajo del comando resaltado con una breve descripción de la función que realiza el comando.

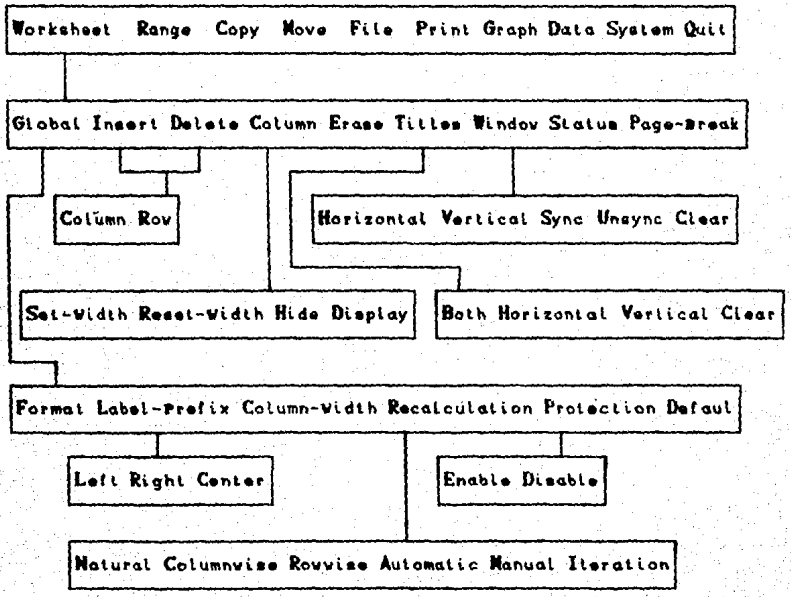
En esta parte, describiremos por secciones los comandos de

acuerdo al orden en el cual aparecen en el menú. Las secciones son: los comandos de **Worksheet**, **Range**, **Copy**, **Move**, **File**, **Print**, **Graph**, **Data**, **Syatem** y **Quit**. Un árbol de menú mostrando el comando y sus submenús aparecen al principio de cada sección.

Cuando creamos una nueva hoja de trabajo, **1-2-3** automáticamente hace ciertas decisiones por nosotros. Estas selecciones son establecidas por definición. Sin embargo, podemos usar los comandos de **1-2-3** para cambiar estas y establecer las nuestras. Los comandos de **Worksheet-Global** cambian estas selecciones para la hoja entera, mientras que los comandos de la orden **Range** los cambian para partes de la hoja.

Y con esta última observación comenzamos la descripción del primer comando: el Comando **Worksheet**.

C O M A N D O W O R K S H E E T



1-2-3 tiene dos tipos de órdenes del comando Worksheet; aquellos que afectan a la hoja entera y aquellos que afectan solo partes de la hoja.

Las órdenes que afectan la hoja entera son las que corresponden a la orden Worksheet-Global. Esta orden nos permite establecer el formato numérico, el alineamiento de etiquetas, el ancho de columna, y el status de protección para la hoja entera. (Podemos invalidar estas suposiciones para partes especificas de la hoja usando ciertas órdenes de los comandos Range o Worksheet).

El comando Worksheet Global Default nos permiten especificar ciertas fijaciones, como son signos monetarios, formatos internacionales de fecha y hora, acceso a Help, despliegue del reloj en la pantalla, el directorio por defecto, y tipos de impresión, que **1-2-3** usa cuando recuperamos hojas de trabajo existentes o las creamos.

Las órdenes del comando Worksheet que afectan partes de la hoja nos permiten insertar o eliminar filas o columnas, cambiar el ancho de columnas especificas, impedir la aparición de columnas en la pantalla, congelar filas o columnas como titulos, poder partes no adyacentes de la hoja simultáneamente, y decirle a **1-2-3** que empieza una nueva página cuando imprimimos.

/**W**orksheet **G**lobal **F**ormat

```
Fixed Scientific Currency , General +/- Percent Date Text Hidden
```

/**WGF** fija la forma en que los valores numéricos aparecen para la hoja entera. Las etiquetas no son afectadas por el comando.

/**W**orksheet **G**lobal **L**abel-Prefix

```
Left  Right  Center
```

/**WGL** fija la alineación de las etiquetas para la hoja entera. Las etiquetas pueden estar alineadas por la izquierda, alineadas por la derecha, o centradas.

/Worksheet Global Column-Width
/WGC fija el ancho para todas las columnas de la hoja excepto para aquellas columnas cuya anchura ha sido individualmente fijada con la orden /Worksheet Column Set-Width.

/Worksheet Global Recalculation

Natural Columnwise Rowwise Automatic Manual Iteration

/WGR controla cuando, en que orden, y cuantas veces son recalculadas las fórmulas en la hoja de trabajo.

/Worksheet Global Protection

Enable Disable

/WGP trabaja en conjunción con las órdenes /Range Protect y /Range Unprotect para evitar cambios hechos a celdas particulares.

/Worksheet Global Default Printer

Interface Auto-LF Left Right Top Bottom Pg-Length Wait Setup

/WGDP especifica la impresora por defecto y el ambiente de la interface.

/Worksheet Global Default Directory

/WGDD especifica el directorio que **1-2-3** automáticamente buscará (cuando recuperemos un archivo) y al cual escribirá (cuando salvemos un archivo) si no se especifica un directorio.

/Worksheet Global Default Status
/WGDS despliega las especificaciones establecidas por las otras órdenes de Worksheet Global Default

/Worksheet Global Default Update
/WGDU salva las especificaciones actuales establecidas por las otras órdenes de Worksheet Global Default en un archivo con extensión CNF

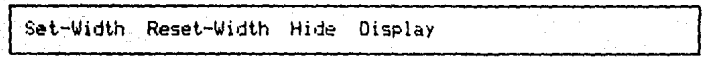
/Worksheet Insert

/WI añade filas o columnas en blanco en la hoja.

/Worksheet Delete

/WD elimina filas o columnas enteras de la hoja.

/Worksheet Column

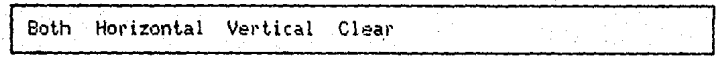


/WC cambia el ancho de una columna, oculta una columna, o vuelve a desplegar una columna oculta.

/Worksheet Erase

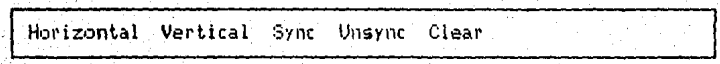
/WE elimina de la pantalla la hoja de trabajo actual y coloca una vacía.

/Worksheet Titles



/WT congela filas o columnas en lo alto o en la orilla izquierda de la pantalla para que así podamos verlas mientras nos desplazamos por la hoja.

/Worksheet Window



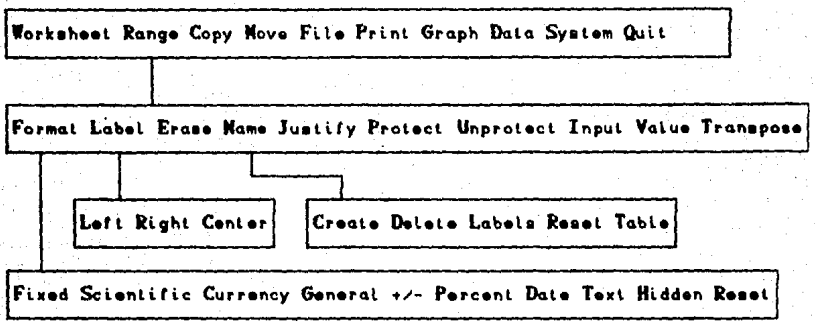
/WW divide la pantalla en dos ventanas horizontales o verticales.

/Worksheet Status

/WS despliega información acerca del uso de memoria, especificaciones globales y opciones de hardware.

/Worksheet Page
/WP inserta un rompimiento de página dentro de la hoja.

C O M A N D O R A N G E



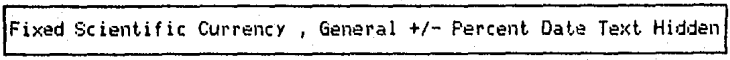
Las órdenes del comando Range manipulan rangos de celdas. Un rango es cualquier bloque rectangular de celdas. Un rango puede ser una sola celda, una fila, una columna, o partes de varias filas y columnas.

Con estos comandos, podemos cambiar el formato numérico de un rango para controlar la presentación de los números, la alineación de etiquetas en un rango, borrar los contenidos de un rango, transponer un rango pasando de una traza vertical (columnas) a una horizontal (filas), y viceversa.

Podemos tambien convertir las fórmulas en un rango a sus valores, proteger o desproteger un rango para prevenir o permitir cambios a entradas, reacomodar "parrafos" de etiquetas largas,

restringuir el movimiento del puntero de celdas a un rango especifico, nombrar un rango, y cambiar o suprimir un nombre de rango.

/Range Format



/RF establece el formato numérico para un rango de celdas, invalidando el formato numérico por defecto.

/Range Label

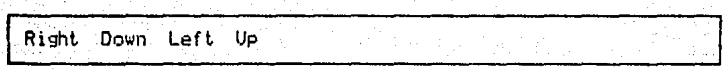
/RL alinea las etiquetas existentes en un rango de celdas. **1-2-3** puede posicionar las etiquetas hacia la orilla izquierda, o derecha, o en el centro de las celdas que componen un rango.

/Range Erase
/RE elimina el contenido de las celdas que forman un rango.

/Range Name Create
/RNC nombra un rango o redefine a que celdas se refiere un rango existente.

/Range Name Delete
/RND elimina el nombre dado a un rango dejando los contenidos del rango sin cambios.

/Range Name Labels



/RNL nombra rangos de una sola celda, usando etiquetas localizadas en celdas adyacentes para los nombres de rango.

/Range Name Reset
/RNR suprime todos los nombres de rango en una hoja de trabajo, pero deja los contenidos de la hoja sin cambios.

/Range Name Table
/RNT lista alfabeticamente todos los nombres de rangos y sus correspondientes direcciones en una tabla de dos columnas en la hoja de trabajo.

/Range Justify
/RJ trata columnas continuas de texto como un parrafo, reacomodando las palabras tal que ninguna de las lineas sea más grande que el de un ancho específico.

/Range Protect
/RP previene cambios y suspensiones a algun rango de celdas cuando la protección global para la hoja de trabajo esta deshabilitada.

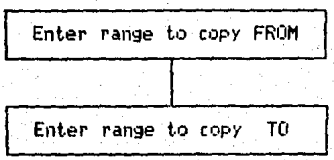
/Range Unprotect
/RU permite cambios a un rango de celdas cuando la protección global para la hoja de trabajo esta habilitada.

/Range Input
/RI limita el movimiento del puntero a celdas hacia celdas desprotegidas dentro de un rango específico.

/Range Value
/RV convierte las fórmulas del rango a sus valores.

/Range Transpose
/RT reordena rangos de columnas a filas o de filas a columnas.

C O M A N D O C O P Y



La orden Copy copia un rango fuente a un rango destino. La tabla siguiente resume los tipos de copias que realiza esta orden.

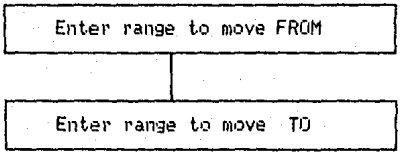
Fuente	Destino
Una celda	Una celda
Una celda	Un rango horizontal(renglón)
Una celda	Un rango vertical(columna)
Una celda	Un rango rectangular
Una columna	Una columna
Una columna	Un bloque rectangular
Un renglón	Un renglón
Un renglón	Un bloque rectangular
Un bloque rectangular	Un bloque rectangular

/C crea copias de celdas con entradas existentes

♦ Cuando copiamos etiquetas y números, **1-2-3** hace duplicados exactos de las entradas originales en otra localidad.

♦ Cuando copiamos formulas, **1-2-3** puede o no ajustar las direcciones de celda(s) en las formulas, dependiendo del tipo de direccionamiento de la celda.(**1-2-3** reconoce tres tipos de direccionamiento de una celda: direccionamiento relativo, absoluto o mezcla).

C O M A N D O M O V E



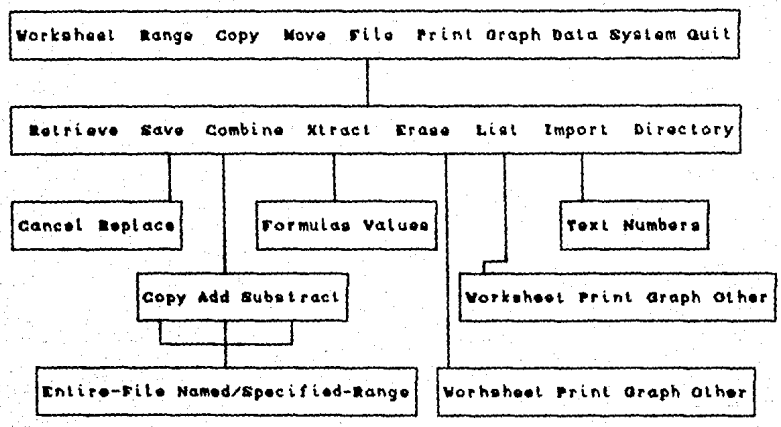
Esta orden traslada un rango de celdas de un área de la hoja de trabajo a otra, por lo que **1-2-3** deja libre el rango fuente de la orden **MOVE** y suprime cualquier entrada que estuviera en el rango destino.

/M transfiere un rango de celdas con entradas desde una parte de la hoja de trabajo a otra.

♦ permite el reacomodo de datos en la hoja mientras mantiene todas las relaciones funcionales entre las celdas conteniendo los datos.**1-2-3** automáticamente ajusta todas las formulas en la hoja explicando los datos movidos.

♦ si se mueve una celda conteniendo una fórmula, la fórmula permanece igual. Si se mueve el contenido de una celda a la que se refiere una fórmula, **1-2-3** cambia la fórmula para reflejar la nueva localización de la celda.

COMANDO FILE



Estas órdenes salvan hojas de trabajo en archivos que son almacenados en disco. Salvando nuestro trabajo en un archivo, podemos recuperarlo despues de haber salido de **1-2-3** o haber apagado la computadora.

Con las órdenes del comando **File**, podemos salvar las hojas de trabajo en archivos, recuperar archivos ya creados, incorporar parte de una hoja de trabajo en otra, salvar parte de una hoja en un archivo separado, borrar archivos de disco, y cargar un archivo de impresión en la hoja de trabajo actual.

/File Retrieve
/FR carga un archivo de disco en la memoria de la computadora, y lo despliega en la pantalla.

/File Save

```
Cancel  Replace
```

/FS salva la hoja de trabajo presente y el ambiente asociado con el en un archivo.

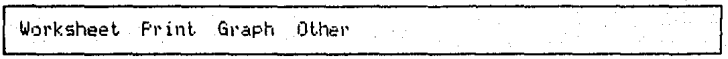
/File Combine

/FC incorpora todo o una parte de un archivo en la hoja de trabajo actual en la localidad del apuntador a celda.

/File Xtract

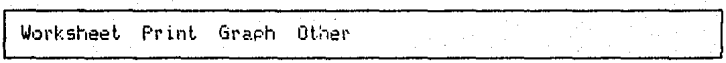
/FX extrae y salva una parte de la hoja de trabajo en un archivo aparte.

/File Erase



/FE elimina uno o más archivos de un tipo de archivo en particular del disco.

/File List



/FL despliega los nombres de todos los archivos de un tipo en particular almacenado en el directorio actual, y el espacio en bytes aún disponible en disco.

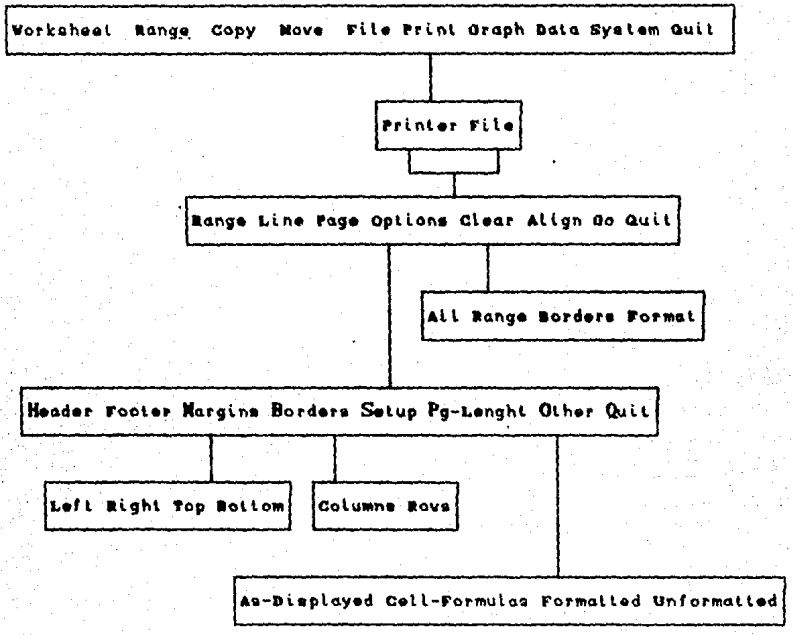
/File Import

/FI copia un archivo de impresión del directorio actual a la hoja de trabajo presente en la localidad del apuntador a celda

/File Directory

/FD reemplaza el directorio presente con uno nuevo, convirtiendolo en el directorio actual para la sesión.

C O M A N D O P R I N T



Estas órdenes nos permiten crear copias impresas de nuestra hoja de trabajo. Pudiendose imprimir la hoja en la impresora, o imprimirla en un archivo salvado en disco. El imprimir hacia un archivo nos permite imprimir el archivo desde DOS, o utilizar el archivo en otro programa , tal como el generado por un procesador de palabras.

Con las órdenes del comando Print, podemos especificar un rango a imprimir, hacer avanzar la impresora por línea o por página, decirle a la impresora que se esta en la parte superior de la página, y especificar opciones de impresión tales como margen, longitud de la página, encabezamientos y pies de página.

/Print Printer o File Range

/PPR y /PFR nos permiten especificar el rango de la hoja que se desea imprimir o almacenar en un archivo.

/Print Printer o File Line

/PPL posiciona la impresora al principio de la siguiente línea.

/Print Printer o File Page

/PPP hace avanzar la página actual al final, e imprime el pie de página.

/Print Printer o File Options

```
Header Footer Margins Borders Setup Pg-Length Other Quit
```

/PPO y /PFO cambia los margenes, y lalongitud de la página de nuestros documentos impresos, tambien para añadir encabezamientos y pies de página, y para indicar tamaño fuente y estilo.

/Print Printer o File Options Other

```
As-Displayed  Cell-Formulas  Formatted  Unformatted
```

/PPOO y /PFOO cambia el formato de impresión y la información que el documento incluye.

/Print Printer o File Clear

```
All  Range  Borders  Format
```

/PPC y /PFC limpia el rango de impresión, encabezamientos, pies de página, y margenes, y otras opciones.

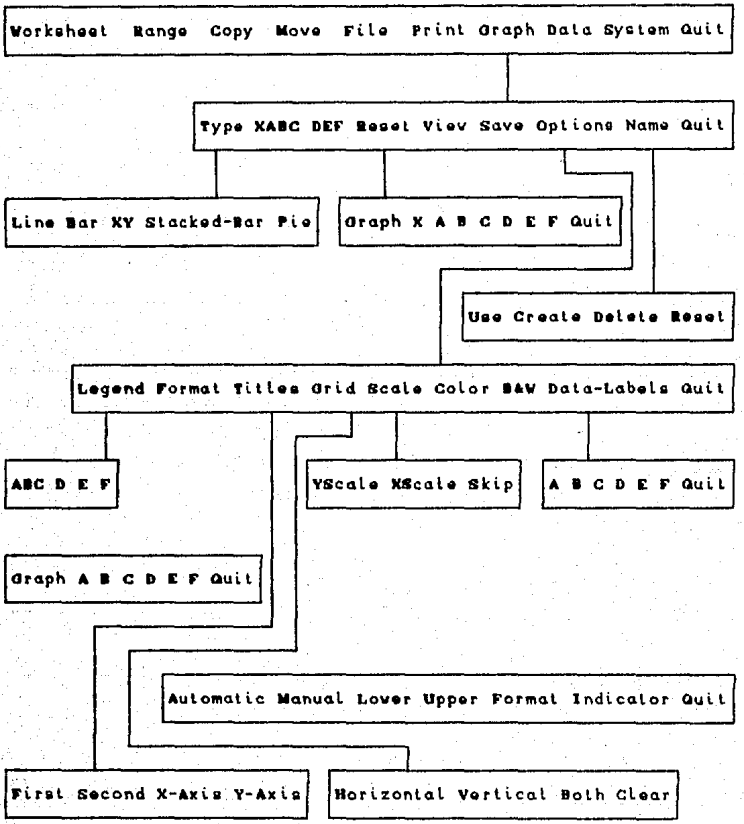
/Print Printer Align

/PPA comunica a la impresora que hemos posicionado el papel en la parte alta de una nueva página.

/Print Printer o File Go

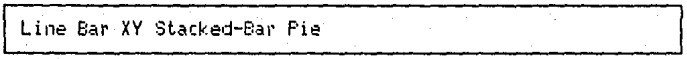
/PPG y /PFG inicia el proceso de impresión del rango que indicamos con /Print Printer Range o /Print File Range.

COMANDO GRAPH



Con este comando podemos representar los datos numéricos de una hoja en forma de una gráfica. Las ordenes del comando Graph pueden crear cinco tipos diferentes de gráficas: lineal, de barras, XY, de barras apiladas, y de pastel.

/Graph Type



/GT usada para seleccionar el tipo de gráfica deseada.

/Graph Reset

/GR cancela gráficos o declaraciones de rangos.

/Graph Save
/GS almacena la gráfica actual en un archivo gráfico.

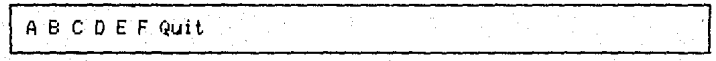
/Graph Options Legends

/GOL añade una leyenda debajo de la gráfica para identificar que representa cada símbolo, color, o sombreado en la gráfica.

/Graph Options Format

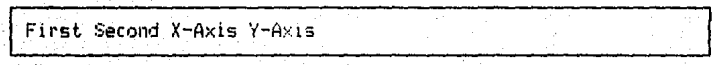
/GOF controla el despliege de los datos para graficas de Linea y XY.

/Graph Options Data-Labels



/GOD etiqueta los puntos con dato en un rango de datos.

/Graph Options Titles



/GOT asigna un título a cada eje o a una gráfica entera.

/Graph Options Grid

Horizontal Vertical Both Clear

/GOG añade o suprime

/Graph Options Scale

Y Scale X Scale Skip

/GOS fija las escalas numéricas para el eje X y el eje Y, y especifica el factor de salto para las etiquetas del eje X.

/Graph Options Color
/GOC despliega las barras, líneas gráficas y símbolos en colores contrastantes.

/Graph Options B&W
/GOB despliega las barras de datos en fondos monocromáticos contrastantes.

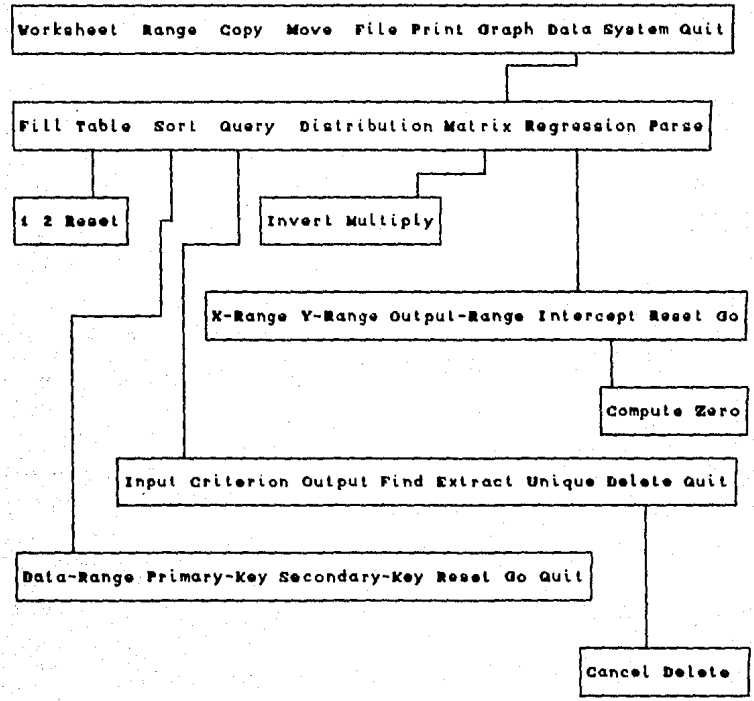
/Graph Name Create
/GNC salva las especificaciones para la gráfica actual bajo un nombre de gráfica.

/Graph Name Use
/GNU nombra un conjunto de especificaciones de una gráfica y dibuja la gráfica.

/Graph Name Delete
/GND borra el conjunto de especificaciones de la gráfica nombrada.

/Graph Name Reset

/GNR borra todos los nombres de graficas.



Las órdenes del comando Data de **1-2-3** nos permiten introducir y analizar datos en una hoja de trabajo.

Varias de las órdenes de Data son usadas con una hoja de trabajo la cual este organizada como una base de datos.

/Data Fill
/DF introduce una secuencia ascendente o descendente de números en un rango especifico de celdas.

/Data Table 1

/DT1 produce una tabla la cual muestra los diferentes valores que una fórmula genera cada vez que cambiamos un valor en esa fórmula.

/Data Table 2

/DT2 produce una tabla la cual muestra los diferentes valores que una fórmula genera cada vez que cambiamos uno o dos valores en esa fórmula.

/Data Sort

```
Data-Range Primary-Key Secondary-Key Reset Go Quit
```

/DS reacomoda los registros en una base de datos en el orden que especifiquemos.

/Data Query Find

/DQF localiza los registros que en una base de datos cumplen con el criterio que especifiquemos.

/Data Query Unique

/DQU trabaja exactamente igual que la orden /Data Query Extract, excepto que elimina cualquier registro duplicado de los registros que **1-2-3** copia al rango de salida.

/Data Query Delete

/DQD borra los registros en el rango de entrada que concuerdan con el criterio y elimina las filas de la base de datos.

/Data Distribution

/DD crea una distribución de frecuencias de los valores en un rango.

/Data Matrix

/DM multiplica e invierte matrices formadas por filas y columnas con entradas.

/Data Regression

X-Range Y-Range Output-Range Intercept Reset Go Quit

/DR computa los valores de los coeficientes y constantes para una fórmula que iguala uno o más rangos de variables independientes con un rango de variables dependientes.

/Data Parse

Format-Line Input-Column Output-Range Reset Go Quit

/DP convierte una columna de etiquetas largas en varias columnas de etiquetas o numeros.

C O M A N D O S Y S T E M

/S permite la utilización del sistema operativo mientras se esta trabajando con **1-2-3**.

C O M A N D O Q U I T

/Q permite la salida de **1-2-3**.

FUNCIONES

Las funciones de 1-2-3 son fórmulas incorporadas las cuales realizan cálculos especializados. En lugar de sumar un rango de números, +A5+A6+A7+A8+A9+A10+A11, podemos utilizar la función @SUM(A5..A11) la cual abrevia nuestro trabajo.

La mayoría de las funciones de 1-2-3 calculan valores numéricos. Otras manipulan secuencias de caracteres, llamadas cadenas.

Tipos de funciones

Funciones matemáticas.- Funciones que realizan cálculos usando valores numéricos.

Esta Función	Devuelve
@ABS(X)	el valor absoluto (positivo) de x.
@ACOS(x)	el arcocoseno del ángulo x.
@ASIN(x)	el arcoseno del ángulo x.
@TAN	el arcotangente del ángulo x (2 cuadrante)
@ATAN2(x, y)	el arcotangente del ángulo y/x (4 cuadrante).
@COS(x)	el coseno del ángulo x.
@EXP(x)	el número e elevado a la xth potencia.
@INT(x)	la parte entera de x.
@LN(x)	el logaritmo natural (base e) de x.
@LOG(x)	el logaritmo (base 10) de x.
@MOD(x, y)	el residuo de x/y.
@PI	el número π (aproximadamente 3.1415926).
@RAND	un número aleatorio entre 0 y 1.
@ROUND(x, n)	x redondeado a n lugares.
@SIN(x)	el seno del ángulo x.
@SQRT(x)	la raíz cuadrada de x.
@TAN(x)	la tangente del ángulo x.

Funciones lógicas.- Funciones que producen valores basados en el resultado de declaraciones condicionales.

Esta función	Devuelve
@FALSE	el valor lógico 0.
@IF(cond, x, y)	x si cond es verdadera, y y si cond es FALSA.
@ISERR(x)	1(VERDADERO) si x contiene el valor ERR;
@ISNA(x)	1(VERDADERO) si x contiene el valor NA; en caso contrario 0(FALSO).
@ISNUMBER(x)	1(VERDADERO) si x contiene un valor numérico en caso contrario 0(FALSO).
@ISSTRING(x)	1(VERDADERO) si x contiene una cadena de caracteres; en caso contrario, 0(FALSO).
@TRUE	el valor lógico 1.

Funciones especiales.- Funciones que ejecutan tareas avanzadas.

Esta función	Devuelve
@@(cell address)	el contenido de la celda referenciada por cel address.
@CELL(atributo, rango)	el código que representa el atributo del rang
@CELLPOINTER (atributo)	el código que representa el atributo de la celda destacada.
@CHOOSE (x, v0, v1,...,vn	el xth valor en la lista v0, v1,...vn.
@COLS(rango)	el número de columnas en rango.
@ERR	el valor ERR(error)
@HLOOKUP(x, rango número de fila)	el contenido de la celda que es especificada por el número de fila de la celda en la fila más alta del rango que es igual a x.
@INDEX(rango, columna, fila)	el valor de la celda localizada en la intersección de columna y fila dentro del rango.
@NA	el valor NA (no disponible).
@ROWS(rango)	el número de filas en el rango.
@VLOOKUP(x, rango, número de columna)	el contenido en la celda que es especificado por el número de columna de la celda en la primera columna del rango que concuerda con x

Funciones que manejan cadenas.-Son funciones que utilizan para sus calculos cadenas de caracteres, las cuales producen valores de cadenas.

Esta función	Devuelve
@CHAR(x)	el caracter ASCII/LICS que corresponde al código del número x.
@CODE(cadena)	el número del código ASCII/LICS para el primer caracter en la cadena.
@EXACT(cadena1, cadena2)	1(VERDADERO) si cadena1 y cadena2 son exactamente iguales; en caso contrario, 0(FALSO).
@FIND(cadena a buscar, cadena, número de inicio)	la posición de la primera ocurrencia de la cadena a busca dentro de cadena.
@LEFT(cadena, n)	los primeros n caracteres en la cadena.
@LENGTH(cadena)	el número de caracteres en la cadena
@LOWER(cadena)	toda la cadena en minúsculas.
@MID(cadena, número inicial,n)	n caracteres de la cadena , comenzando con el caracter que ocupa la posición número inicial
@N(rango)	el valor numérico en la celda superior izquierda del rango.
@PROPER(cadena)	todas las letras en cadena con la primera letra en mayúsculas y el resto en minúsculas.
@REPEAT(cadena, n)	la cadena duplicada n veces
@REPLACE(cadena original, número inicial, n, nueva cadena)	elimina n caracteres de la cadena original, comenzando en la posición número inicial para insertar entonces la nueva cadena a partir de ahí.
@RIGHT(cadena, n)	los últimos n caracteres en la cadena.
@S(rango)	el valor de la cadena que se encuentra en la esquina superior izquierda del rango.
@STRING(x, n)	el valor numérico de x como una cadena, con n decimales.
@TRIM(cadena)	
@UPPER(cadena)	todas las letras de la cadena en mayúsculas.
@VALUE(cadena)	una cadena que se ve como un número en su valor numérico actual.

Funciones de fecha y tiempo.

Esta Función	Devuelve
@DATE(año, mes, día)	el número de día de la fecha año,mes,día
@DATEVALUE (fecha cadena)	el número de día de fecha cadena.
@NOW	el número en serie para la fecha y hora actual.
@TIME(hr, min, seg)	el número de hora de hr, min, seg.
@TIMEVALUE (hora cadena)	el número de hora para hora cadena.
@DAY(número de fecha)	el número de día de número de fecha.
@HOUR(número de hora)	el número de hora de número de hora.
@MINUTE (número de hora)	el número de minutos de número de hora.
@MONTH (número de fecha)	el número de mes de número de fecha.
@SECOND (número de hora)	el número de segundos de número de hora
@YEAR(número de fecha)	el número de año de número de fecha.

Funciones financieras.- Funciones que calculan prestamos, anualidades, y flujos de efectivo durante un periodo de tiempo.

Esta función	Devuelve
@CTERM(int, fv, pv)	el número de periodos compuestos para un inversión de valor presente pv, la cual crecerá a un valor futuro fv, ganando en un periodo fijo a una tasa d interes int
@DDB(costo, salvamento vida, periodo)	descuento por la depreciación de doble declinación de un activo, dado el costo original,el valor de salvamento predicho la vida del activo, y el periodo específico.
@FV(pmt, int, plazo)	el valor futuro de una serie de pagos iguales, cada uno por una cantidad pmt, ganando a una tasa de interes periodico int, sobre el número de periodos de pago durante el plazo.

@IRR(conjetura,rango)	la tasa interna de retorno para las series de flujo de efectivo en un rango, basados en el porcentaje aproximado d la IRR dado por conjetura.
@NPV(int, rango)	el valor presente de las series de flujo de efectivo en un rango, descontandose a una tasa de interes periodica int.
@PMT(prin, int, plazo)	la cantidad del pago periodico necesario para pagar el capital prin,a una tasa de interes periodica int,sobre el número de periodos de pago en un plazo.
@PV(pmt, int, plazo)	el valor presente de una serie de pagos iguales, cada uno por una cantidad pmt, descontandose a una tasa de interes periódico int, sobre el número de periodos de pago en un plazo.
@RATE(fv, pv, plazo)	la tasa de interes periódico necesario para el valor presente pv, para crecer a un valor futuro fv, sobre el número de periodos compuestos durante el plazo.
@SLN(costo, salvamento, vida)	el descuento por medio d la depreciación en línea recta de un activo para un periodo dado, dado el costo, el valor de salvamento predicho, y la vida dl activo
@SYD(costo, salvamento, vida, periodo)	el descuento por medio d la depreciación de la suma de los dígitos de los años de un activo para un periodo, dado el costo el valor de salvamento predicho, y la vida del activo y el periodo específico.
@TERM(pmt, int, fv)	el número de periodos de pago de una inversión, dada la cantidad de cada pago pmt, la tasa de interes periodica int, y el valor futuro de la inversión fv.

Funciones estadísticas.- Funciones que calculan listas de valores.

Esta función	**Devuelve**
@AVG(lista)	el promedio de los valores en la lista.
@COUNT(lista)	el número de entradas no-blancas en la lista
@MAX(lista)	el valor máximo en la lista.
@MIN(lista)	el valor mínimo en la lista.
@STD(lista)	la desviación estandar de los valores en la lista.
@SUM(lista)	la suma de los valores en la lista.
@VAR(lista)	la varianza de los valores en la lista.

Funciones estadísticas de la base de datos.-Estas funciones realizan cálculos estadísticos sobre la base de datos. La base de datos, llamada rango de entrada, consiste de registros, campos y nombres de campo. Un rango de criterio debe ser declarado para seleccionar los registro de la base de datos que cada función usa.

Esta función	Devuelve
@DAVG(entrada, offset, criterio)	el promedio de los valores en la columna offset del rango de entrada que concuerdan con el criterio.
@DCOUNT(entrada, offset, criterio)	el número de celdas con entradas en la columna offset del rango de entrada que concuerdan con el criterio.
@DMAX(entrada, offset, criterio)	el valor máximo en la columna offset del rango de entrada que concuerda con el criterio.
@DMIN(entrada, offset, criterio)	el valor mínimo en la columna offset del rango de entrada que concuerda con el criterio.
@DSTD(entrada, offset, criterio)	la desviación estandar d los valores en la columna offset del rango de entrada que concuerdan con el criterio.
@DSUM(entrada, offset, criterio)	la suma d los valores en la columna offset del rango de entrada que concuerdan con el criterio.
@DVAR(entrada, offset, criterio)	la varianza de los valores en la columna offset del rango de entrada que concuerdan con el criterio.

CONCLUSIONES

1-2-3 sobresale en factores de ingeniería humana, osea, aquellos elementos de un programa que lo hacen fácil de usar. No podemos dejar de hacer énfasis de la importancia de la ingeniería humana en los programas de microcomputadoras. A la fecha, las computadoras han sido difíciles de entender y molesto su uso, lo cual ha desalentado a mucha gente a utilizarlas. 1-2-3 es una de las pocas piezas de software que pueden literalmente ser usadas por cualquier persona. Usted puede comprar 1-2-3 y una computadora personal y utilizar los dos el mismo día.

3 DEFINICION DE LA HOJA DEL PROYECTO

3.1 Características y Funcionamiento de MicroC

MicroC es una hoja electrónica de cálculo similar a SuperCalc, VP-Planner, MUltiplan y muchos otros programas comerciales.

La hoja es una matriz rectangular de celdas organizada en columnas y renglones. La pantalla del monitor actúa como una ventana que muestra una sección de la hoja (la hoja es normalmente demasiado grande para aparecer completa en una pantalla). Órdenes sencillas corren la ventana en cualquier dirección, permitiendo que cualquier parte de la hoja pueda ser vista. Los usuarios del programa escriben en la hoja moviendo el cursor de la pantalla a la celda deseada y escribiendo la información. En cualquier celda el usuario puede escribir un **rótulo** (cualquier cadena de caracteres), un **número** ouna **fórmula** (un cálculo con resultado numérico).

Una fórmula de una celda puede referirse a los valores contenidos en otras celdas; su valor depende entonces de esas otras celdas. Esto hace a los programas sobre hojas especialmente útiles para responder a preguntas del tipo **"que pasa si"**. Por ejemplo, una hoja almacenaría los resultados del cálculo de la amortización de un préstamo basándose en determinados índices de interés. Los resultados para otro interés pueden obtenerse sencillamente cambiando las celdas que contienen el interés a otro valor; el resto de la hoja se calcula entonces bajo esta nueva suposición.

De esta forma las hojas electrónicas permiten utilizar la computadora en problemas como hacer el balance de un libro de cuentas, presupuestos y el cálculo de los impuestos,que de otra forma se resolverían con una calculadora o con lápiz y papel. Por supuesto, la computadora realiza una compleja manipulacion de los posibles datos introducidos (algo no tan fácilmente realizable con lápiz y papel).

MicroC se diferencia de los distintos programas sobre hojas comerciales. Es relativamente lento y no ofrece un gran número de funciones y características. Por otra parte, **MicroC** es más flexible que muchos de los programas comerciales, debido a que es fácilmente modificable; se puede añadir,suprimir o cambiar las características que se deseen.

Funcionamiento de MicroC

A continuación ofrecemos una breve descripción de la forma de operar de **MicroC** ;más adelante al diseñar el programa, se dará una descripción más detallada. Supondremos que el tamaño de la pantalla es de 80 columnas por 25 renglones. La adaptación de **MicroC** a tamaños de pantalla mayores o más pequeños consiste sólo en visualizar de una vez un número mayor o menor de columnas y renglones sobre la pantalla.

Cuando comienza **MicroC** , la pantalla aparece como lomuestra la figura 3.1.1. Se presentan las siete primeras columnas de la hoja y parte de una octava; estas columnas se rotulan y etiquetan en la parte superior de la pantalla. (Hay que tener cuidado de distinguir entre columnas de la hoja y columnas de la pantalla. Una columna de la hoja puede ocupar varias columnas de pantalla. La figura 3.1.1 muestra cómo cada columna de la hoja está formada por diez columnas de pantalla, lo cual es lo más normal. Los renglones van numerados de arriba a abajo, a la izquierda de la pantalla; en primer lugar se ofrecen los primeros 19 renglones. El cursor ocupa la celda de columna 1, renglón 1; estoesta indicado por el **[1,1]** ubicado en la esquina superior izquierda dela pantalla.Además cuenta con una línea que permanece durante toda la sesión, describiendo las principales funciones que puede realizar **MicroC** , y que está colocada en la última línea de la pantalla.

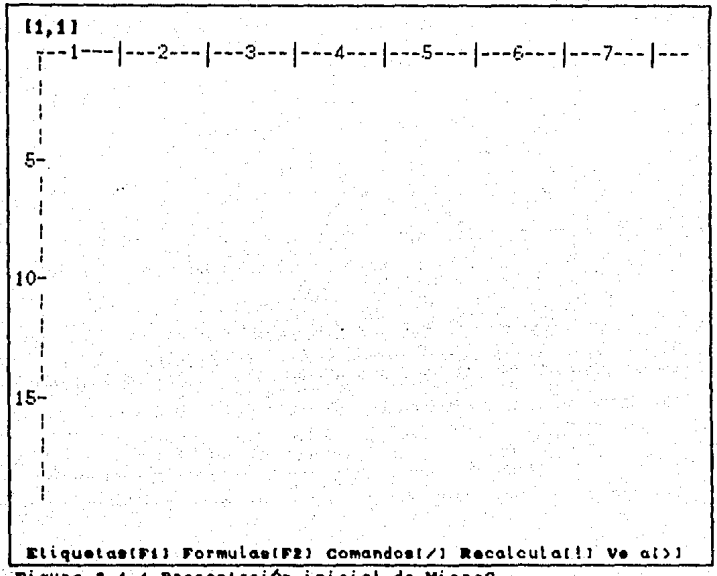


Figura 3.1.1 Presentación inicial de MicroC.

A partir de esta posición inicial, se pueden realizar cualquiera de las cuatro operaciones siguientes, en cualquier orden.

Mover el Cursor

Se puede mover el cursor a una celda adyacente pulsando una de las cuatro teclas de movimiento del cursor (← ↓ → ↑). Se puede acceder a cualquier celda de la hoja desde cualquier otra;por ejemplo para ir a la posición de inicio se presiona la tecla **HOME** o ir a cualquier dirección por medio de la orden " > " (Ir a).

El único error es intentar llevar el cursor fuera de la hoja, en este caso, la orden de movimiento de cursor es ignoraday la computadora suena.

Introducir un Rótulo

Después de mover el cursor a una celda se puede colocar un rótulo en ese punto de la hoja. Se señala el deseo de introducir un rótulo presionando bien una letra o la tecla **F1** . Si se escribe una letra **MicroC** supone que es el primer caracter del rótulo que va a introducirse; elprograma requiere que se introduzca el rótulo, insertando automáticamente la letra que se ha escritocomo el primer caracter del rótulo. Se usa la tecla **F1** cada vez que se quiere introducir un rótulo que no comienza con una letra.

Las reglas comunes de introducción de cadenas también sirven para introducir rótulos. **<BACKSPACE> o <DEL>** borran el último caracter introducido y contienen cualquier caracter imprimible. Cuando se ha introducido el rótulo aparece en la celda actualmente ocupada por el cursor. Se ofrece el rótulo entero aún cuando éste se extienda más allá del espacio normal de la columna. (Estrictamente hablando, únicamente la parte del rótulo que cabe en la ventana de la pantalla normal es la que se presenta; puede que no se muestre el principio o el final de la cadena si sobresale de la ventana).

Introducir un Número o una Fórmula

También se puede introducir un número o una fórmula en la posición actual del cursor. Realmente el programa no distingue entre números y fórmulas, sino que considera a un número como una fórmula simple. Si se desea introducir una fórmula se puede hacerlo escribiendo cualquiera de los siguientes caracteres: **<0> a <9>** , un signo **<+> o <->**, un punto decimal **<.>** , un paréntesis izquierdo **<(>**, un corchete izquierdo **<[>**, o la tecla **<F2>** cuando una fórmula no comienza con ninguno de los caracteres anteriores.

De esta forma se puede evaluar cualquier expresión matemática que contenga operaciones con cantidades en punto flotante,con natación científica utilizando el exponente **<E>**,o con referencias

a celdas de la hoja. Para evaluar las expresiones se usa la precedencia algebraica normal,los paréntesis son los que tienen la mayor prioridad para ser evaluados.

Las fórmulas pueden contener cualquiera de las operaciones siguientes:

Nombre	Símbolo	Operación	Descripción
Producto	*	x * y	Multiplica los números **X** y **Y** .
División	/	x / y	Divide **X** entre **Y** .
Suma	+	x + y	Suma los números **X** y **Y**.
Resta	-	x - y	Resta de **X** el número **Y**.
Exponenciación	^	x ^ y	Eleva la base **X** a la potencia **Y**.
Raiz n	^	x ^ (1/n)	Obtiene la n raiz de **X**

Tabla de operaciones.X,Y son son reales o celdas y n es entero.

Las celdas se usan para conservar resultados previamente calculados. De este modo,en lugar de nombres identificadores,una fórmula puede usar referencias a celdas que indican que,el valor de esa celda va a usarse en la fórmula.

Hay dos tipos de referencias a celdas;absoluta y relativa.Una referencia a celda absoluta es introducida bajo la forma : **[<columna>,<renglón>]**, la columna y renglón de la celda referenciada,se encierran en corchetes y se separan por una coma. Por ejemplo,la referencia a la celda **[3,20]**, hace que se use el valor de la celda de la columna 3, y renglón 20.

La referencia relativa a celda,especifica la celda en términos de su localización con respecto a la celda que contiene la fórmula. La referencia relativa a celda es indicada por signos que preceden a los números de la columna, del renglón o a ambos.Por ejemplo si suponemos que la fórmula de la celda **[19,12]**

contiene la referencia relativa a la celda **[-3,+5]**, quiere decir que se va a usar el valor de la celda,3 columnas a la izquierda y 5 renglones abajo de la celda actual. De este modo, el valor de la celda [16,17] se usa en el punto de la fórmula.

El programa tiene funciones básicas en la utilización de una hoja electrónica, 2 de las funciones que fueron implementadas en **MicroC** son: la función **SUM** que se usa para sumar un número arbitrario de celdas. Por ejemplo **SUM([1,1]:[1,10])** se puede usar en una fórmula para sumar 10 celdas consecutivas de la columna 1 comenzando del renglón 1 hasta la columna 10. La segunda función es **PROM** que se utiliza para obtener el valor promedio de un conjunto de celdas. Por ejemplo **PROM([2,1]:[2,5])** se utilizaría en una fórmulaen lugar de escribir la fórmula extendida **([2,1]+[2,2]+[2,3]+[2,4]+[2,5])/5** .

Introducir una Orden

El programa suministra un conjunto de órdenes que permiten al usuario, borrar celdas de lahoja, recalcular la hoja, guardar la hoja en disco, entre otras.

Cuando el programa esta comenzando, el menú de opciones se presenta presionando la tecla **</>** y escogiendo la letra inicial de cualquiera de los comandos descritos,éste se ejecuta.

En el siguiente cuadro se describen todos los comandos que se emplean en **MicroC**.

ORDEN	SECUENCIA CLAVE	DESCRIPCION
Recalcular	!	Recalcula la hoja.
Ir a	>	Mueve el cursor a la celda especificada por el usuario.
Copiar	/ C	Copia celdas de una parte de la hoja a otra.
Borrar	/ B	Borra una celda,columnas,renglones o toda la hoja.
Insertar	/ I	Inserta renglones o columnas en blanco.
Recuperar	/ R	Recupera una hoja grabada previamente en disco.
Archivar	/ A	Graba la hoja de trabajo en disco.
Salir	/ S	Termina la sesión.
Memoria libre	/ M	Muestra la memoria libre restante.
Imprimir	/ X	Imprime la hoja en papel.

3.2 Estructuras de Datos

La hoja de cálculo **MicroC** está formada por 63 columnas, cada una de las cuales consta de 255 renglones de celdas. La manera "natural" de representar la hoja en memoria sería declarando un arreglo bidimensional, que en lenguaje **C** se declara delsiguiente modo:

struct celdptr celda[MAXCOLS][MAXRENS];

donde el tipo de datos **celdptr** es una estructura o registro que contiene toda la información perteneciente a una sola celda.Pero este arreglo tiene **MAXCOLS*MAXRENS** elementos, un total de casi 16000. Si cada registro **celdptr** toma diez bytes (lo cual es probablemente una estimación excesivamente baja) la matriz requeriría cerca de 160,000 bytes de memoria. Esto en la mayoría de de los compiladores de **C** provocaría un error de exceso de dimensiones del arreglo.

Por supuesto, podríamos hacer la hoja menor decrementando **MAXCOLS** y **MAXRENS** hasta que el arreglo de registros **celda** quepa en la memoria disponible. En su lugar, usaremos un método alternativo para representar las celdas en memoria, una estructura de datos llamada " **Matriz Espaciada** ".

Una matriz espaciada es una matriz en la que la mayoría de los elementos están "vacios", es decir, no contienen información. Cuando muchos elementos están vacios no hay ninguna razón para usar memoria para almacenarlos. En su lugar, seutiliza el siguiente sistema: cada columna de la matriz tiene una **cabecera de columna** , que es un puntero al elemento más alto no vacio de la columna. Si no hay elementos no vacios en la columna,el apuntador es **NULL** , es decir, no apunta a ninguna parte. Cada elemento de la columna contiene también un apuntador al siguiente elemento no vacío que se encuentra por debajo de él en la matriz; este puntero es **NULL** si el elemento es el último no vacío de la columna.

Lo mismo se hace para los renglones.Un arreglo de apuntadores

es la **cabecera de renglón** y apunta a las celdas más a la izquierda de cada renglón y son **NULL** si no hay celdas en el renglón. Cada elemento tiene un apuntador derecho que apunta a la siguiente celda a la derecha del renglón.

La figura 3.2.1 muestra este método con las celdas no vacías, **(1,1),(1,2),(1,5),(3,2),(3,5),(4,1) y (5,2)**.

Los apuntadores de cabecera de columna se muestran como un arreglo de cajas numeradas,en la parte superior de la figura y la cabecera de renglones se encuentra en la parte izquierda. Se usa el símbolo eléctrico de tierra para indicar un apuntador **NULL** .Y donde cada celda tiene dos apuntadores, uno hacia abajo en la columna y otro a la derecha en el renglón.

En lenguaje **C** , podemos realizar esta estructura de datos como sigue. Primero definimos el tipo **celdptr**.

```
struct cedlptr {
        int celdcol;
        int celdren;
        struct formptr *fp;
        char *display
        struct celdptr *rightptr,*downptr;
        };
```

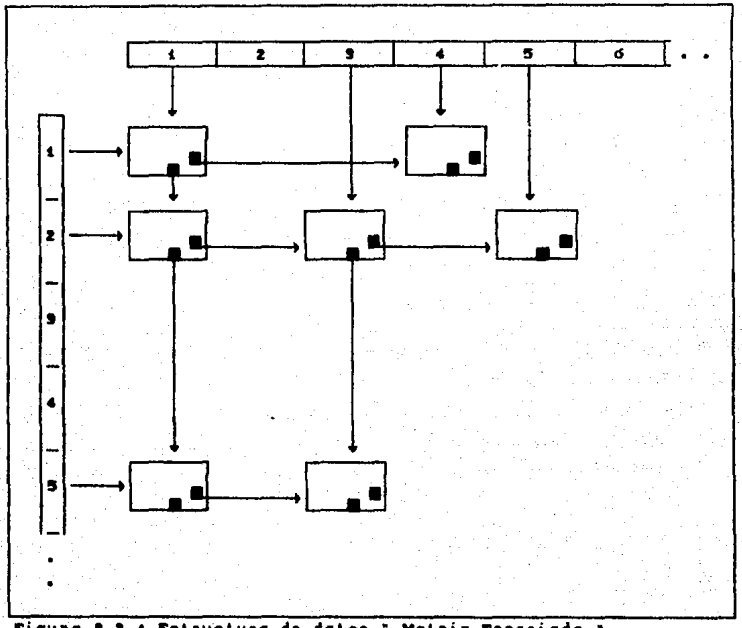


Figura 9.2.1 Estructura de datos " Matriz Espaciada "

Además de los apuntadores derecho e inferior a otras celdas,cada registro de celda contiene campos para las coordenadas de su propia columna y renglón. Estos datos se necesitan ya que si tuviéramos que aproximarnos a una celda desde la izquierda, a través de apuntadores de renglón,no podríamos saber **que columna** es en la que está una celda dada. Similarmente, si nos aproximasemos a una celda desde arriba, a través de los apuntadores de columna, sería difícil descubrir el renglón de la celda. El último paso es definir las cabeceras de las columnas y de los renglones, como sigue:

```
struct celdptr *colptr[MAXCOLS];
struct celdptr *renptr[MAXRENS];
```

Dada esta estructura de datos con sus coordenadas de columna y renglón, podemos escribir una rutina para encontrar una celda en la hoja. La función **busceld** realiza este trabajo:

```
/* Busca la celda en la hoja,devuelve un puntero a ella o a
NULL si no existe */
struct celdptr *busceld(c,r)
int c,r;
{
      struct celdptr *cp;
      char done;

      if( c<1 || c > MAXCOLS || r<1 || r > MAXRENS )
            return NULL; /* Fuera de la hoja */
      else{
            cp = renptr[r];
            done = FALSO;
            while( cp->celdcol < c)
                  if( cp->celdcol < c)
                        cp = cp->rightptr;
                  else{
                        done = TRUE;
                        if( cp->celdcol != c)
                              cp = NULL;
                        }
      return cp;
      }
}
```

Esta rutina devuelve un apuntador a la celda especificada por su columna **c** y el renglón **r** .Si tal celda no está definida en la hoja (o las coordenadas dadas quedan fuera de los límites de la hoja), **busceld** devuelve un apuntador a **NULL** . La estrategiaque usa **busceld** es la de buscar el renglón especificado **r** para una celda con un valor celdcol de **c** ; comienza en la celda señalada por **renptr[r]** y se mueve hacia la derecha a lo largo del renglón hasta que, o bien se encuentra la celda deseada o la búsqueda fracasa.(Prodríamos haber escrito **busceld** para buscar la columna **c** para una celda con un valor **celdren** de **r**).

Contenido de las Celdas

A continuación describiremos los contenidosdel registro de la celda.

A partir de la discusión anterior,sabemos que una celda no vacía contiene o una etiqueta o una fórmula. En cualquiera de los casos,una celda necesita contener un campo que describa **lo que se va a presentar en la pantalla** . En caso de que la celda contenga una etiqueta, este campo es simplemente la propia etiqueta. Cuando la celda contiene una fórmula, el campo contiene una cadena apropiadamente formateada que representa el valor numérico de la fórmula.

Una manera de hacer esto sería, definiendo un campo de la siguiente manera:

```
char display[80];
```

El campo **display** que contendría la cadena que se ofrecería en la posición de la celda. Este enfoque **no** es el que usamos porque realmente necesita demasiada memoria. Ya que para cada elemento utilizaría un tamaño fijo de 80 caracteres, pero nuestro requerimiento para el campo **display** es probablemente mucho menor de 80. Si, por ejemplo, nuestro rótulo medio es de 10 caracteres, desperdiciariamos 70 bytes de memoria cada vez que se creara una nueva celda.

¿ Por qué no declaramos un tamaño de cadena máximo menor para el campo **display**? Esto disminuiría la necesidad de almacenamiento, como se desea. Pero también queremos permitir la posibilidad de usar 80 caracteres o másen campo **display.** Si el usuario desea utilizar un rótulo de 80 o más caracteres en la hoja, el programa no debe prohibirlo.

Este tema surge siempre al diseñarprogramas. Cada variable que definimos tiene una necesidad fija de memoria que no puede cambiarse durante la ejecución del programa.

Nuestra salida a este dilema, es definir un apuntador a una cadena:

```
char *display;
```

Formateado

Si sólo alamcenamos rótulos, la estructura de datos que hemos mostrado podría ser adecuada.Pero una celda también puede contener una fórmula, por lo que necesitamos un campo adicional para este caso. Sin embargo antes de considerar la estructura dedatos para almacenar fórmulas, necesitamos tener en cuenta cual va ser el formateado para los resultados numéricos de una fórmula.

La flexibilidad de variables es uno de nuestros fines, así que hemos puesto un formato general para el manejo de los valores calculados en cada fórmula. El programa automáticamente revizará cuando es necesario usar notación científica o si es suficiente con la representacón de la cantidad en punto flotante.Este es uno de los puntos importantes que puede ser mejorada en versiones futuras del programa **MicroC.**

Almacenamiento de Fórmulas

Ahora podemos considerar el método de almacenamiento de las celdas con fórmula. En el tipo de datos **celdptr**,definimos un apuntador **fp** a un registro o estructura que contiene toda la información sobre la fórmula.

```
struct formptr *fp;
```

Esto permite a **MicroC** distinguir fácilmente si una determinada celda contiene un rótulo o una fórmula .Si es un rótulo, el apuntador **fp** es**NULL**; en otros casos,**fp** apunta a un registro **formptr** válido. Esta asignación tiene el beneficio adicional de que no necesita utilizarse la memoria extra para almacenar información sobre la fórmula si la celda contiene un rótulo.La estructura del registro **formptr** es como sigue:

```
struct formptr {
        double celdval;
        char *formula;
        };
```

El campo **celdval** contiene el valor calculado de la fórmula introducida. Escogimos el tipo de datos **double** para representar cantidades numéricas porque podemos controlar el rango y la precisión del tipo de datos y porque podemos evitar fácilmente los errores de desbordamiento y desbordamiento negativo.

Finalmente, el campo **formula** contiene un apuntador a una cadena de la fórmula producida por el usuario.Usamos un apuntador a cadena en lugar de la propia cadena por las mismas razones explicadas con respecto al campo **display** en el registro **celdptr** ; esto nos va permitir utilizar la memoria con mayor eficacia.

3.3 Presentacion de la Hoja

MicroC utiliza diversas rutinas que controlan la presentación de la hoja sobre la pantalla del monitor.La primera función que **MicroC** llama es **dibujahoja** que presenta un esqueleto de la hoja;es decir, dibuja los ejes vertical y horizontal sin rotular y un indicador de posición del cursor en blanco.

```
/* Dibuja los bordes de la hoja */
dibujahoja()
{
    int i;

    crt(CLEAR);
    posstr(dashes,XORG,YORG-1);
    gotoxy(XORG-1,YORG);
    for(i = 1; i <= nrens; i++){
        if(i % 5 == 0)
            printf("-");
        else
            printf("|");
        crt(LEFT);
        crt(DOWN);
    }
    posstr("   ",0,0);
}
```

Se eligió rotular cada 5 renglones de la hoja;la función **etiquetarens** realiza esta operación.Esta función es llamada con un parámetro, el número de renglón queva a colocarse en la fila superior de la hoja.

```
/* Etiqueta los renglones de la hoja */
etiquetarens(r1)
int r1;
{
      int r,y;

      primeren = r1;
      ultimoren = primeren + nrens - 1;
      r = primeren + 1;
      y = YORG + 1;
      while(r <= ultimoren){
            gotoxy(0,y);
            printf("%3",r);
            r+=5;
            y+=5;
      }
}
```

La llamada a **etiquetaren** actualiza las variables globales **primeren** y **ultimoren**,que controlan los renglones primero y último que actualmente aparecen en la pantalla.

La función **etiquetacols** realiza el trabajo análogo de rotular las columnas de la hoaj. Puesto que cada hoja toma variascolumnas de la pantalla, la tarea es ligeramente más complicada.

```
/* Etiqueta las columnas de la hoja */
etiquetacols(c1)
int c1;
{
      int c,crtcol,vid,i;

      posstr(dashes,XORG,YORG-1);
      firstcol = c1;
      c = firstcol;
      do{
            crtcol = colpos(c+1)-colpos(firstcol)+XORG-1;
            if(crtcol <= MAXCRTCOL){
                  posstr("|",crtcol,YORG-1);
                  ultimacol = c;
```

```
                vid = colpos[c+1] - colpos[c];
                if(vid > 2){
                        gotoxy(crtcol - (vid+1)/2,YORG-1);
                        printf("%2d",c);
                }
        }
        c++;
    }while(crtcol <= MAXCRTCOL && c <= MAXCOLS);
}
```

Este procedimiento destruye cualquier rotulación de columnas anterior poniendo la cadena **dashes** en el lugar correcto de la pantalla.Se ponen separaciones es esta "linea gobernante" para marcar la última columna de la pantalla de cada columna de la hoja.El número de columna se rotula aproximadamente en el centro de la misma. **etiquetacols** actualiza las variables globales **primercol** (que contiene el número de la primera columna de la hoja presentada en la pantalla y **lastcol** (que contiene el número de la última columna de la hoja presentada en la pantalla).

El arreglo **colpos** que se usa en **etiquetacols** contiene la posición de comienzo relativa de cada columna de la hoja. Se utilizó un arreglo para que en versiones posteriores,se puedan utlizar columnas de la hoja de diferentes tamaños y con el uso del arreglo simplificaría la tarea de calcular el comienzo y el final de las columnas de la pantalla de una columna dada de la hoja y de decidir si una determinada celda aparece en la pantalla o no.

El arreglo **colpos** se inicializa por el procedimiento **poncolpos**:

```
/* Pone el peso WIDTH para cada columna */
poncolpos()
{
        int c;

        colpos[1];
        for(c = 1; c < MAXCOLS; c++)
                colpos[c+1] = colpos[c] + WIDTH;
}
```

El elemento **colpos[1]** se pone arbitrariamente a 0. El resto del arreglo **colpos** se calcula sumando simplemente sumando el peso **WIDTH** en cada posición,este peso actualmente es de 10 columnas de pantalla por cada columna de la hoja.

La función **desphoja** presenta la hoja.Se llama con dos parámetros **c1 ,r1** para especificar las coordenadas de la celda que van a presentarse a partir de la esquina superior izquierda de la ventana de la pantalla. Aunque también se llama desde la rutina principal de **MicroC** con propósitos de incialización,es llamada desde muchas otras funciones, bien para presentar lahoja después de que hayaa sido cambiada, o bien para mover la ventana para presentar otra sección de la hoja.

desphoja vuelve a rotular primero los renglones y columnas de la hoja si se ha trasladado la ventana (lo que se refleja en un valor de **c1** diferente a **primercol** , o un valor de **r1** diferente de **primeren** , o ambos). A continuación se presentan los contenidos de la sección de la hoja de la ventana, renglón por renglón.

```
/* Despliega la información de la hoja en la pantalla */
desphoja(c1,r1)
int c1,r1;
{
      register r;

      if(r1 != primeren)
            etiquetarens(r1);
      if(c1 != firstcol)
            etiquetacols(c1);
      for(r = primeren; r <= ultimorem; r++)
            despren(r);
}
```

La función **despren** es el responsable de presentar un renglón simple de la hoja en la pantalla;es donde nuestra analogía "la pantalla es una ventana para la hoja" se implementa. **despren** debe decidir de alguna forma si una cadena de presentación de una determinada celda aparece en la ventana o no.Esta decisión es algo

complicada por el hecho de que la cadena de presentación de la celda puede aparecer parcialmente en la ventana, cortada por la arista de la ventana, bien en la cara derecha, o bien en la cara izquierda, o posiblemente por ambas caras.

La rutina en C es la siguiente:

```
/* Presenta un renglón de información en la pantalla */

despren(r)
int r;
{
      struct celdptr *cp;
      char done,x[15];
      int crtrow,crtcol,c1,c2;

      crtrow = YORG + r - primeren;
      goloxy(XORG,crtrow);
      crt(ERASEOL);
      cp = renptr(r);
      done = FALSO;
      while(cp != NULL && !done){
            crtcol = colpos(cp->celdcol)-colpos(firstcol)+XORG;
            if(crtcol > MAXCRTCOL)
                  done = TRUE;
            else
                  if(cp->display != NULL){
                    c1 = max(1,XORG-crtcol+1);
                    c2 = min(strlen(cp->display),MAXCRTCOL -
                                                          crtcol + 1);
                    if(c2 >= c1){
                        goloxy(crtcol + c1 - 1,crtrow);
                        printf("%s",copy(cp->display,c1,c2-c1+1);
                    }
            }
            else
                  if(cp->fp != NULL){
                    gcvt(cp->fp->celdval,8,x);
                    c1 = max(1,XORG-crtcol+1);
                    c2 = min(strlen(x),MAXCRTCOL-crtcol+1);
                    if(c2 >= c1){
                        goloxy(crtcol+c1-1,crtrow);
                        printf("%s",copy(x,c1,c2-c1+1));
                    }
                  }
            cp = cp->rightptr;
      }
}
```

En este código, la variable **c1** almacena el índice del primer carcter de la cadena presentada y **c2** almacena el índice del último caracter de la cadena presentada.

despren utiliza la función **copy,** una rutina que su implementación se encuentra en el apéndice A.La llamada:

```
substr = copy(s,i,n);
```

copia **n** caracteres de la cadena **s** comenzando en el caracter **i-ésimo**, de la cadena **substr** .

despren también utiliza las funciones **min y max**; son rutinas sencillas y útiles que devuelven el menor y el mayor de sus dos argumentos enteros, respectivamente. He aquí estas dos rutinas:

```
/* Encuentra el mínimo entre i1 e i2 */
min(i1,i2)
int i1,i2;
{
      if(i1 <= i2)
            return i1;
      else
            return i2;
}


/* Encuentra el máximo entre i1 e i2 */
max(i1,i2)
int i1,i2;
{
      if(i1 >= i2)
            return i1;
      else
            return i2;
}
```

Colocación y movimiento del cursor

El siguiente objetivo es diseñar las rutinas que tratan con el cursor: ponerlo en una posición especifica de la hoja y moverlo de un lugar a otro de la hoja.

La función **poncursor** coloca el cursor en una celda específica

por las coordenadas de su renglón y su columna.

```
/* Pone el cursor en el punto especificado */
poncursor(c,r)
int c,r;
{
        int ci,ri;

        if(c < firstcol)
                ci = c;
        else
                if(c > ultimacol){
                        ci = c;
                        while((colpos(c+1)-colpos(ci) <= MAXWIDTH &&
                                                                        (ci > 1))
                                ci = ci - 1;
                        if(colpos(c+1)-colpos(ci) > MAXWIDTH)
                                ci = ci +1;
                }
                else
                        ci = firstcol;
        if(r <primeren)
                ri = r;
        else if(r > ultimorem)
                ri = r - nrens + 1;
        else
                ri = primeren;
        cursor(0);
        if(ri != primeren || ci != firstcol)
                desphoja(ci,ri);
        gotoxy(1,0);
        printf("%2d",c);
        cr(RIGHT);
        printf("%3d",r);
        cursor(1);
        curscol = c;
        cursren = r;
        curcp = busceld(curscol,cursren);
        borralinea(21);
        if(curcp != NULL)
                if((curcp->fp == NULL) && (curcp->display !=
                                                        display != NULL)){
                        posicr("Etiqueta:",0,21);
                        printf("%s",curcp->display);
                }
                else if(curcp->fp != NULL)
                        if(curcp->fp->formula != NULL){
                                gotoxy(0,21);
```

```
                    printf("Fórmula:");
                    printf("%s",curap->fp->formula);
                }
        gotoxy(XORG+colpos[c] - colpos[firstcol],YORG+r - primeren);
}
```

Si la celda no está actualmente representada en la pantalla, **poncursor** corre la ventana para colocar el cursor en la pantalla, bien ajustando la primera columna de la hoja presentada, o bien la primera fila de la hoja presentada o ambas. La única parte algo compleja de este ajuste se plantea cuando el cursor se traslada fuera de la arista derecha de la ventana antigua; entonces hemos de encontrar un nuevo valor de **primercol** de modo que el nuevo cursor a la columna de la hoja es ahora la última columna de la hoja que aparece completa en la pantalla. **poncursor** conoce esto yendo hacia atrás en el arreglo **colpos** desde el nuevo cursor a la columna de la hoja.

Luego **poncursor** actualiza el indicador de posición del cursor de la esquina superior izquierda para que refleje la nueva posición del cursor. Si la celda a la que el cursor apunta no está vacia, **poncursor** tambien presenta el contenido de la celda (o un rótulo o una fórmula) en la línea de estado del programa siguiente a la ventana. (Esto es importante puesto que es la única forma cómoda de encontrar una fórmula de un a celda si se ha olvidado). La rutina descubre si la celda está o no vacía llamando a la función **busceld.**

Finalmente, **poncursor** coloca el cursor de la pantalla en la posición del primer caracter de la celda en la pantalla. El orden es importante: **poncursor** debe dejar el cursor en la posición correcta de la pantalla.

Una vez que se ha diseñado **poncursor,** escribir una rutina para mover el cursor es fácil.

```
/* Mueve el cursor en la dirección especificada por ch */
muevecursor(ch)
char ch;
{
```

```
    if(ch == 72 && curaren > 1) /* Movimiento hacia arriba */
        poncursor(curacol,curaren-1);
    else if(ch == 80 && curaren < MAXRENS) /* HAcia abajo */
        poncursor(curacol,curaren+1);
    else if(vh == 77 && curacol < MAXCOLS) /* A la derecha */
        poncursor(curacol+1,curaren);
    else if(ch == 75 && curacol > 1)    /* A la izquierda */
        poncursor(curacol-1,curaren);
    else                            /* Movimiento fuera de la hoja */
        crt(BEEP);
}
```

Basándose en le carcater escrito por el usuario, **muevecursor** mueve el cursor de la hoja en la dirección especificada llamando a **poncursor**, para poner el cursor en la celda adyacente apropiada.Si el usuario trata de mover fuera de la hoja, **muevecursor** da un pitido de advertencia.

3.4 Etiquetas y Asignación Dinámica

Introducción de Rótulos

Ahora podemos escribir **obtetiqta**, la cual acepta un rótulo del usuario y lo unserta en la hoja. He aquí **obtetiqta**:

```
/* Obtiene del asuario una etiqueta */
obtetiqta(ch)
char ch;
{
        borralinea(inpline);
        posstr("Etiqueta:",0,inpline);
        obtcadena(s,MAXCRTCOL-6,9,inpline,ch);
        borralinea(inpline);
        if(strlen(s) > 0){
                if(curcp != NULL)
                        borraceld(curcp);
                if(!storeceld(curscol,curaren,curcp))
                        memout();
                else if(!storestr(s,curcp->display))
                        memout();
        }
        despren(curaren);
        poncursor(curscol,curaren);
}
```

En general **obtetiqta** acepta un rótulo del usuario y lo almacena en memoria creando una nueva celda en la estructura de matriz espaciada de la hoja. (Si hay ya una celda no vacía en la actual posición del cursor, se borra la vieja celda antes de que se almacenen los nuevos datos para la misma). Luego actualiza la presentación de la hoja (usando **despren**) para mostrar el nuevo rótulo.

Si no hay sitio en memoria para almacenar la nueva celda o la cadena del rótulo, **obtetiqta** llama a **memout** para hacérselo saber al usuario. El diseño de **memout**, **obtcadena** y **borralinea** se encuentran en el apéndice A.

Asignación dinámica de memoria

Nuestro siguiente paso fue resolver el problema de cómo almacenar variables de diferente longitud (como las cadenas) sin desperdiciar memoria. Idealmente, nos gustaría que los datos se almacenaran en memoria mientras se necesitan; cuando ya no se necesitan más, la memoria utilizada por ellos debería devolverse al sistema para otros propósitos. Este proceso se denomina **asignación dinámica de memoria** . El término "dinámica" se utliza porque las necesidades de memoria se determinan mientras se ejecuta el programa.

El lenguaje **C** nos suministra lo que necesitamos, utilizaremos las funciones incorparadas **malloc** y **free** que asigna y desasigna tamaños específicos de memoria. Supongamos que la variable **sp**, es un apuntador a una cadena, entonces la llamada:

```
sp = (char *) malloc (nbytes);
```

encuentra **nbytes** bytes consecutivos de memoria y los asigna para ser utilizados. **malloc** devuelve la dirección del bloque de memoria como resultado de su función . Si **malloc** no puede encontrar un bloque de memoria lo suficientemente grande, devuelve un valor **NULL** para comunicar su fallo.

La memoria asignada por **malloc** puede ser desasignada por **free.** La llamada:

```
free(sp);
```

libera un área de memoria de igual tamaño al que le fue asignada. **malloc** y **free** son compañeras, intentar llamar a **free** con una dirección no asignada por **malloc**, o con una variable que no ha sido asignada por **malloc** ,en cuando menos peligroso.

Gestion de la memoria en lenguaje C

Para tener más claro el proceso de la asiganación dinámica,haremos un breve estudio de cómo se usa la memoria de la

computadora durante la ejecución de un programa, de modo que sepamos como estan organizados los bloques y apuntadores iniciales de la memoria.

Durante la ejecución de un programa, la memoria de la computadora se prepara para diferentes utilizaciones: el almacenamiento del código del programa y los datos, el sistema operativo, buffers E/S, etc.

Por lo general, todo el sistema de utilización de memoria se encuentra en el fondo de la memoria (direcciones bajas) o en la cima (direcciones altas).

La región de la memoria que queda en medio es la memoria libre disponible para ser utilizada por el programa. El tamaño de esta zona de memoria libre depende del tamaño del programa.

En el lenguaje **C**, el sistema anota las posiciones inferiores y superiores de la memoria libre. El fondo de la memoria libre recibe el nombre de **montón** y la cima receibe el nombre de **pila**. La figura 3.4.1 muestra esta sencilla disposición.

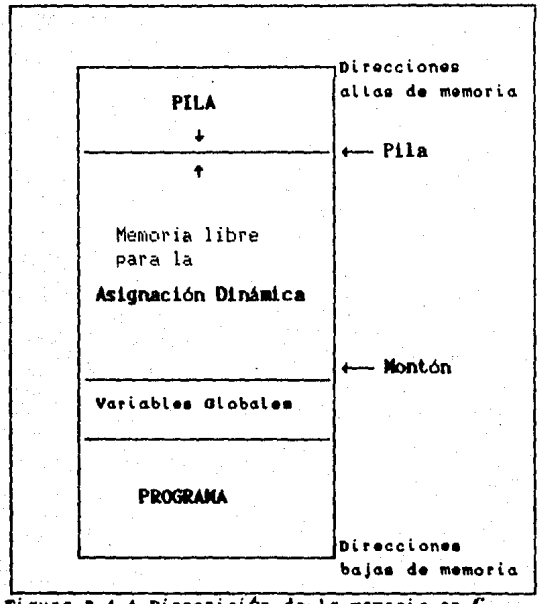


Figura 3.4.1 Disposición de la memoria en C.

El sistema utiliza la pila para obtener el almacenamiento temporal necesario para llamar a las rutinas. Siempre que se llama a una función, se necesita cierta cantidad de memoria para alamcenar las variables locales de la función, devolver la dirección o cualquier otro tipo de información. Así pues, cuando se llama a una función, la pila crece hacia abajo en memoria y cuando la función termina, la memoria es liberada y la pila se reduce hacia arriba en memoria.

El montón, por otra parte se utiliza para asignar variables dinámicas, siempre que se asigna espacio a una variable dinámica, el montón se desplaza hacia arriba y cuando es liberada se desplaza hacia abajo en memoria.

Cuando el montón y la pila colisionan, es decir, cuando la memoria libre se reduce por completo, el programa sale con un

error de ejecución "desbordamiento de memoria", en otras palabras el sistema está fuera de memoria.

Almacenamiento dinámico de cadenas

Nuestras herramientas para la asignación dinámica de memoria no suelen utilizarse directamente, sino que se utilizan por medio de rutinas que **almacenan** y **borran** variables de diferentes tipos llamando a **mallloc** y **free**. Por ejemplo, **storestr** se utiliza para almacenar una cadena en la memoria dinámica.

```
/* Almacena una cadena en la memoria dinámica */
storestr(s,sp)
char *s, **sp;
{
      sp = (char **) malloc (strlen(s));
      if( !sp )
            return 0;
      else{
            strcpy(sp,s);
            return 1;
      }
}
```

Esta rutina utiliza **malloc** para encontrar y asignar espacio a la cadena S. El número de bytes solicitados por **malloc** es **strlen(s)+1**; el número de caracteres dela cadena. Si no hay sitio para almacenar la cadena, **storestr** devuelve un apuntador a la cadena con **NULL** y 0 como resultado de la función, significando que ha habido un error. En los demás casos, almacena la cadena en la posición asignada, devolviendo un apuntador a la cadena y 1 como resultado dela función significando que ha almacenado la cadena.

Una cadena almacenada se borra dela memoria dinámica con **borracad**.

```
/* Borra una cadena de la memoria */
borracad(sp)
char *sp;
{
```

```
    if(sp != NULL)
        free(sp);
}
```

borracad simplemente llama a **free** para devolver el espacio anteriormente reservado por **malloc** a la lista libre.

Almacenamiento de una celda

Ahora podemos diseñar rutinas para almacenar y borrar celdas en y desde nuestra estructura de matriz espaciada. Nuestro primer paso es almacenar celdas con **storeceld.** La llamada

storeceld(col, ren, cp);

intenta crear una celda en memoria con las coordenadas de columna y renglón **col** y **ren**. El parámetro por variable **cp** devuelve un apuntador a la celda creada; el resultado de la función es un booleano, que refleja si la celda se ha creado o no.

storeceld es algo más complejo que **storestr**. Además de asignar espacio para la celda, **storeceld** debe enlazar la celda con la existente, en la estructura matriz espaciada. He aquí **storeceld**:

```
/* Crea una nueva celda en la posición especificada */
storeceld(curscol,curscren)
int curscol,curscren;
struct celdptr *cp;
{
    struct celdptr *cp1,*cp2;
    char done;

    cp = (struct celdptr *) malloc (sizeof(struct celdptr));
    if( !cp )
        return 0;
    else{
        cp->celdcol = curscol;
        cp->celdren = curscren;
        cp->display = NULL;
        cp->fp = NULL;
        cp1 = NULL;                    /* Enlace con la columna */
```

```
        cp2 = colptr(curacol);
              done = FALSO;
        while(cp2 != NULL && !done)
              if(cp2->celdren <= curaren){
                    cp1 = cp2;
                    cp2 = cp2->dovnptr;
              }
              else
                    done = TRUE;
        cp->dovnptr = cp2;
        if(cp1 == NULL)
              colptr(curacol) = cp;
        else
              cp1->dovnptr = cp;
        cp1 = NULL;                        /* Enlace con el renglón */
        cp2 = renptr(curaren);
        done = FALSO;
        while(cp2 != NULL && !done)
              if(cp2->celdcol <= curacol){
                    cp1 = cp2;
                    cp2 = cp2->rightptr;
              }
              else
                    done = TRUE;
        cp->rightptr = cp2;
        if(cp1 == NULL)
              renptr(curaren) = cp;
        else
              cp1->rightptr = cp;
        return 1;
  }
}
```

storeceld utiliza la función **sizeof**, la cual devuelve la cantidad de memoria en bytes necesitada por el tipo especificado en su argumento.

La rutina **borraceld** quita (desenlaza) una celda específica dela matriz espaciada y libera la memoria anteriormente ocupada por la celda. He aquí **borraceld**:

```
/* Suprime una celda de la hoja */
borraceld(cp)
struct celdptr *cp;
{
      struct celdptr *cp1;
```

```
    if(cp != NULL){
        borraform(cp->fp);
        borracad(cp->display);
        if(colptr(cp->celdcol) == cp)  /*Desliga de la columna*/
            colptr(cp->celdcol) = cp->dovnptr;
        else{
            cpi = colptr(cp->celdcol);
            while(cpi->dovdnptr != cp)
                cpi = cpi->dovdnptr;
            cpi->dovnptr = cp->dovnptr;
        }
        if(renptr(cp->celdren) == cp)  /*Desliga del renglón */
            renptr(cp->celdren) = cp->rightptr;
        else{
            cpi = renptr(cp->celdren);
            while(cpi->rightptr != cp)
                cpi = cpi->rightptr;
            cpi->rightptr = cp->rightptr;
        }
        free(cp);
    }
}
```

Además, **borraceld** llama a **borracad** para borrar la cadena presentada en la celda y a **borraform** (presentada más adelante) para borrar los datos de la fórmula de la celda, si la hubiera.

Almacenamiento de una fórmula

Nuestro último ejemplo ilustra el almacenamiento y supresión de registros de información para las fórmulas. Ambos son relativamente fáciles de realizar. **storeform** asigna espacio para un registro **formptr** y devuelve un apuntador al mismo. He aquí **storeform** :

```
/* Crea un registro de información para una fórmula */
storeform(fp)
struct formptr *fp;
{
    fp = (struct formptr *) malloc sizeof(struct formptr));
    if( !fp )
        return 0;
```

```
    else{
        fp->celdval = zero;
        fp->formula = NULL;
        return 1;
    }
}
```

borraform llama a **borracad** para borrar la cadena de la fórmula, luego llama a **free** para devolver el registro **formptr** a la memoria libre. He aquí **borraform** :

```
/* Suprime un registro de información de fórmula de la memoria */
borraform(fp)
struct formptr *fp;
{
    if(fp != NULL){
        borracad(fp->formula);
        free(fp);
    }
}
```

3.5 LAS FÓRMULAS

Nuestro siguiente objetivo principal en el diseño de **MicroC** es escribir la parte del programa que maneja los números: la aceptación de fórmulas del usuario, el cálculo de los valores y el formateado de los resultados. El sitio natural para empezar es con **obtformula** :

```
/* Obtiene del usuario una fórmula */
obtformula(ch)
char ch;
{
        char success;

        borralinea(inpline);
        poestr("Formula:", 0, inpline);
        getformstr(s,ch);
        borralinea(inpline);
        if(strlen(s) != 0){
                if(curcp != NULL)
                        borraceld(curcp);
                success = storeceld(curacol, curaren, curcp);
                if(success)
                        success = storeform(curcp->fp);
                if(success)
                        success = storestr(s, curcp->fp->formula);
                if(!success)
                        memout();
                else{
                        calceld(curcp);
                        desprer(curaren);
                }
        }
        setcursor(curacol, curaren);
}
```

Gran parte de **obtformula** ya es familiar. después de aceptar la fórmula como la cadena **s**, borra cualquier información anterior de la celda en la posición del cursor llamando a **borraceld. obtformula** crea entonces una nueva celda en la posición actual del cursor llamando a **storeceld**, y crea un registro de información de la fórmula para la celda, llamando a **storeform**. Finalmente,

alamcena la fórmula como una cadena llamando a **storestr**.

Cualquiera de estas asiganciones dinámicas de memoria podría posiblemente causar un error de desbordamiento de memoria. Por tanto, **obtformula** hace un chequeo después de cada asignación para asegurarse de que todo va bien; si falla algún paso,se le informa al usuario de problema.

Si todas las asignaiones se han podido hacer, **obtformula** trabaja sobre la celda introducida, calculando el valor de la expresión introducida y presentándola en pantalla en la posición actual.

Podríamos usar **obtcadena** para aceptar fórmulas,al igual que las usamos para aceptar rótulos en **obtetiqta**. Sin embargo,si queremos añadir una característica importante debemos contar con una función aparte. Por tanto, **obtformula** llama a **obtformcad** para la entrada de la cadena de la fórmula.

```
/* Obtiene del usuario una fórmula */
obtformcad(s,def)
char s[],def;
{
      char ch,e[1];
      int relcol,relren,i;

      strcpy(s,"");
      e[0]=def;
      e[1]='/0';
      gotoxy(INPCOL,inpline);
      if(tec_esp){
            posstr(e,INPCOL,inpline);
            strcpy(s,e);
            gotoxy(INPCOL+1,inpline);
      }
      ch=get_key();
      while(ch!=13){
            if(ch==8 && strlen(s) > 0){
                  crt(LEFT);
                  printf(" ");
                  crt(LEFT);
                  if(strlen(s) > 0) /* Suprime un caracter */
                        s[strlen(s)-1]='';
            }
            else if(ch==27 && strlen(s) > 0){
```

```
                for(i=strlen(s); i <= 1; i--){
                        crt(LEFT);
                        printf(" ");
                        crt(LEFT);
                }
                strcpy(s," ");
        }
        else  if(!tec_esp  &&  ((ch  >=  '0'  &&  ch  <=  '9')  ||     ch
              =='+' || ch == '-' || ch == '.' || ch == '(' ||
              ch == '(' || ch == '@' || ch == ')' || ch == '/'
              || ch == ')' || ch == ',' || ch == ';' || ch ==
              ' ' || (ch >= 'A' && ch <= 'Z' || ch == '^' ||
              ch == 'e') && strlen(s) < (MAXCRTCOL-INPCOL)){
                addchar(s,ch,MAXSTR);
                printf("%c",ch);
        }
        else
                crt(BEEP); /* Caracter ilegal */
    ch = get_key();
    }
}
```

RECALCULO

Llegamos ahora al problema del recálculo: la evaluación de la fórmula de cada celda y la asignación del resultado al valor de la celda. El primer paso es diseñar **recalc**, que recalcula todas las celdas con fórmula de la hoja.

```
/* Recalcula la hoja */
recalc()
{
    int r,c;
    struct celdptr *cp;

    center("Recalculando . . .",statline);
    for(c=1; c <= MAXCOL; c++){
        cp = colptr(c);
        while(cp != NULL){
            calceld(cp);
            cp = cp->downptr;
        }
    }
    borralinea(statline);
}
```

El recálculo se hace por columnas empezando en la celda **[1,1]**, luego **[1,2]**, luego **[1,3]**, y así sucesivamente, seguido de **[2,1]**, **[2,2]**, etc, luego **[3,1]** y así hasta terminar con todas las columnas diferentes de **NULL.**

recalc llama a **calcceld** para evaluar una fórmula de una celda. He aquí **calcceld** :

```
/* Calcula una celda */
calcceld(cp)
struct celdptr *cp;
{
      char i;
      enum xresult result;

      i = 0;
      cpa = cp;
      if( cp != NULL )
            if(cp->fp->formula != NULL){
                  result = evalexpr(cp->fp->formula, &i,&cp->fp->
                                                            celdval);
                  if(result != OK){
                        remark("Funcion indefinida");
                        borraceld(cp);
                  }
            }
}
```

La función **calcceld** a su vez llama a la función **evalexpr** que evalua la expresión matemática de una celda y su método de evaluación es descrito a continuación.

EVALUACIÓN DE EXPRESIONES

El siguiente paso es evaluar expresiones. La figura 3.5.1 muestra el diagrama sintáctico de una expresión válida. El diagrama muestra la secuancia de una expresión de uno o más **términos** separados por signos de más y menos. El primer término puede tambien no llevar signo.

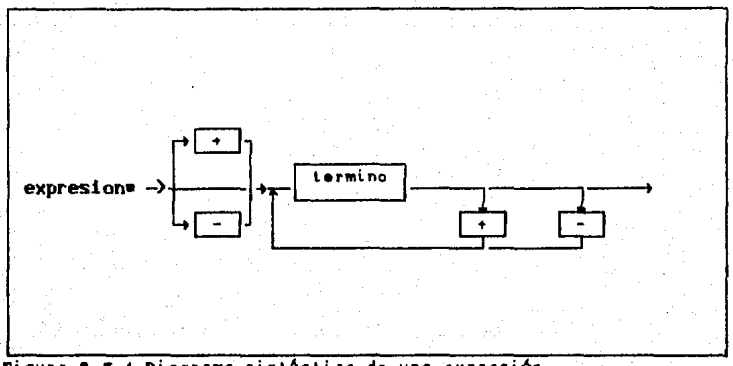


Figura 9.5.1 Diagrama sintáctico de una expresión.

El diagrama sintáctico se traduce a lenguaje C en la siguiente función de una forma natural:

```
/* Evalua una expresión matemática */
evalexpr(s,i,x)
char s[],*i;
double *x;
{
        char c,negterm;
        double xt;
        enum xresult result;

        negterm = FALSO;
        c = gnbchar(s,&*i);
        if( c == '+' || c == '-'){
                *i += 1;
                if(c == '-')
                        negterm = TRUE;
                else
                        negterm = FALSO;
        }
        result = evalterm(s, &*i, &*x);
        if(negterm)
                *x = -*x;
        c = gnbchar(s, &*i);
        while(result == OK && (c == '+' || c == '-')){
                *i += 1;
                result = evalterm(s, &*i, &xt);
                if(result == OK)
```

```
                if(c == '+')
                        result = xadd(&*x, xi, &*x);
                else /* c == '-' */
                        result = xsub(&*x, xi, &*x);
        c = gnbchar(s, &*i);
    }
    return result;
}
```

evalexpr mira primero si aparece el signo de cabecera opcional; la variable **negterm** recibe el valor **TRUE** si aparece un signo menos al principio. Si el signo de comienzo es un signo más o ninguno, **negterm** es **FALSO. evalexpr** llama entonces a **evalterm** para que se evalúe el primer término; si **negterm** es **TRUE** , el valor del primer termino se invierte de signo. El valor del primer término se convierte en nuestra suma.

A continuación **evalexpr** busca términos adicionales. Si el siguiente caracter no blanco es un < + > o < - >,es anotado y **evalterm** evalúa el siguiente término. Después de que cada término haya sido evaluado, es decir, haya sido sumado o restado de nuestra suma, dependiendo de si el signo previo era < + > o < - >. La aritmética es realizada por las rutinas **xadd** y **xsub**. La evaluación de la expresión continúa hasta que no se encuentran más signos, un error señalado por un resultado no-**OK** de **evalterm, xadd o xsub** termina la evaluación.

EVALUACIÓN DE TERMINOS

Evaluar un término es aun más sencillo que evaluar una expresión. La figura 3.5.2 muestra el diagrama sintáctico de un término. Un término consta de uno o más factores separados por asteriscos, slashes o un triángulo. La traducción de un término a lenguaje C es directa.

```
/* Evalua términos de una expresión */
evalterm(s, i, x)
char s[], *i;
double *x;
```

```
{
        double xi;
        enum xresult result;
        char c;

        result = evalfact(s, &*i, &*x);
        c = gnbchar(s, &*i);
        while((result == OK) && (c == '*' || c == '/' || c== '^'))
                *i += 1;
                result = evalfact(s, &*i, *xi);
                if(result == OK)
                        if(c == '*')
                                result = xmult(&*x, xi, &*x);
                        else if(c == '/')
                                result = xdiv(&*x, xi, &*x);
                        else        /* c es '^' */
                                *x = pow(*x,xi);
                c = gnbchar(s, &*i);
        }
        return result;
}
```

De nuevo **evalterm** delega la responsabilidad de evaluar un elemento específico del lenguaje, **factor** ,en otro módulo **evalfact**; **evalterm** no sabe siquiera lo que es un factor. Tambien aislamos el trabajo de realizar las multiplicaciones, las divisiones y las operaciones de exponenciación en los módulos **xmul, xdiv** y la función incorporada del lenguaje **C pow**.

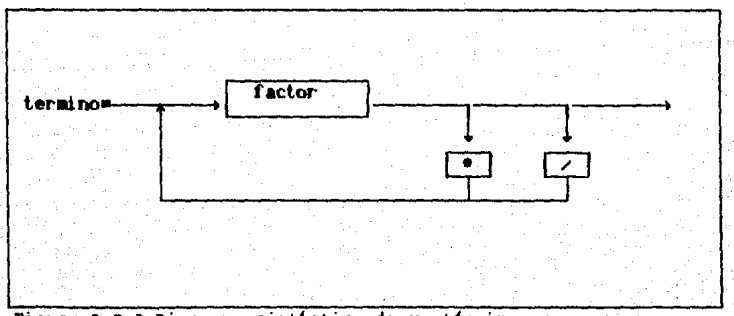


Figura 3.5.2 Diagrama sintáctico de un término.

EVALUACIÓN DE UN FACTOR

El siguiente paso es avaluar los factores. La figura 3.5.2 muestra el diagrama sintáctico. Un factor es, o bien una sentencia entre paréntesis, un identificador o un número. Puede tener un signo cabecera opcional. He aquí la correspondiente rutina en lenguaje **C** :

```
/* Evalua factores */

evalfact(s, i, x)
char s[], *i;
double *x;
{
      char negterm, id[8], ch;
      enum xresult val;
      char c;

      negfact = FALSO;
      c = gnbchar(s, &*i);
      if(c == '+' || c == '-'){
            *i += 1;
            if(c == '-')
                  negterm = TRUE;
            else
                  negterm = FALSO;
      }
      if(gnbchar(s, &*i) == '('){
            *i += 1;
            val = evalexpr(s, &*i, &*x);
            if(gnbchar(s, &*i) == ')')
                  *i += 1;
            else
                  remark("Faltan parentesis derecho");
      }
      else if(c == 'f')
            val = evalcelda(s, &*i, &*x);
      else if(findida(s, &*i, id))
            val = evalida(s, &*i, id, &*x);
      else
            val = stox(s, &*i, &*x);
      if(negfact)
            *x -= *x;
      return val;
}
```

Esta es una ramificación de tres caminos: después de anotar y saltar sobre el signo opcional **evalfact** busca el siguiente caracter no blanco. Si ve un paréntesis izquierdo, la rutina lo salta y llama a **evalexpr** para que evalue la siguiente sentencia, después de evaluar la sentencia, **evalfact** espera ver el paréntesis derecho que lo equilibra como el siguiente caracter no blanco. Si lo hay, pasa al siguiente; si no lo hay, se envía un mensaje de error.

Si no hay paréntesis izquierdo, comprueba si existe un corchete izquierdo indicando una referencia a celda; si encuentra uno, se llama a **evalceld** para extraer desde la cadena de la fórmula y devolver el valor de la celda referenciada:

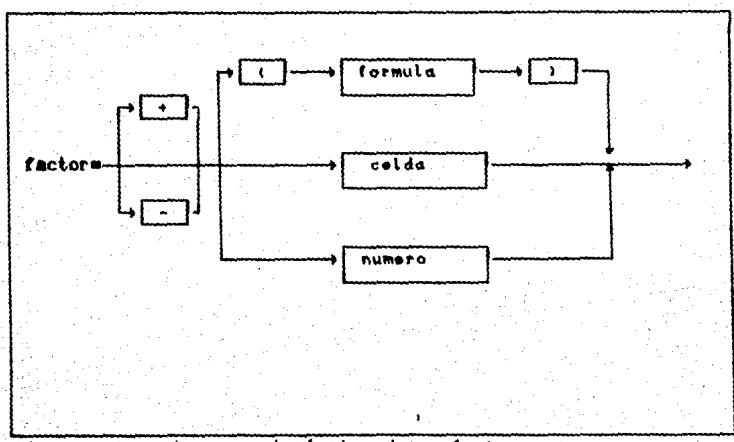


Figura 9.5.9 Diagrama sintáctico de un factor.

```
/* Evalua la referencia a la celda */
evalcelda, i, x)
char s[];
char *i;
double *x;
{
```

```
    struct celdptr *cp1;
    int col, ren;
    enum xresult evalcel;

    *x = zero;
    getcoords(a, &*i, &col, &ren);
    cp1 = busceld(col, ren);
    if(cp1 != NULL)
        if(cp1->fp != NULL){
            *x = cp1->fp->celdval;
            evalcel = OK;
        }
        else
            evalcel = INVALID;
    return evalcel;
}
```

Si la celda referenciada no esta en la hoja (lo que se indicará por un apuntador a **NULL** devuelto por **busceld**) o si la celda contiene un rótulo en vez de una fórmula (indicado por un apuntador de información de fórmula a **NULL**), **evalceld** devuleve un 0 (cero) como valor de la celda, si la celda referenciada tiene un estado de error (desbordamiento negativo, o división por cero), **evalceld** pasa esa información a **evalfact.**

La función **evalceld** llama a **obtcoord** para extraer la referencia de celda de la cadena de la fórmula y traducirla en coordenadas del renglón y la columna. **obtcoord** debe tratar con referencias absolutas y relativas, lo cual es algo delicado. He aquí **obtcoord** :

```
/* Obtiene el renglón y la columna de una referencia a celda */
obtcoords(a, i, col, ren)
char a[],*i;
int *col, *ren;
{
    char relcol,relren;

    stocs(a, &*i, &*col, &*ren, &relcol, &relren);
    if(relcol)
        *col = *col + cpa-> celdcol;
    if(relren)
        *ren = *ren + cpa->celdren;
}
```

La parte delicada es el manejo de referencias relativas y a renglones. Para calcular las coordenadas de una referencia relativa adecuadamente, **getcoord** debe conocer las coordednadas de la celda quese esta evaluando, las cantidades **cp->cledren** y **cp->celdcol.** El valor de **cp** se pasó a **caldceld** como un parámetro, y puesto que **getcoord** se encuentra dentro de **calceld,** puede (y lo hace) acceder a **cp** . Esta práctica, aunque lícita, es muy peligrosa y poco clara.

La alternativa, sin embargo, es pasar **cp** como parámetro a través de **evalexpr, evalterm, evalfact** y **evalceld,** lo cual resulta bastante pesado. El mejor procedimiento cuando se utilizan técnicas de programación un tanto cuentionables es avisar a cualquier lector del programa con un comentario prominente.

getcoord llama a **stoc** para traducir la referencia a celda en la fórmula en un par de coordenadas.

```
/* Extrae de la cadena un par de coordenadas */
stoc(s, i, col, ren, relcol, relren)
char s[], *i, *relcol, *relren;
int *col, *ren;
{
        char c;

        if(gnbchar(s, &*i) == '[')
                *i += 1;
        c = gnbchar(s, &*i);
        if(c == '+' || c == '-')
                *relcol = TRUE;
        else
                *relcol = FALSO;
        *col = ponparam(s, &*i, -MAXCOLS+1, MAXCOLS-1, 0);
        if(gnbchar(s, &*i) == '-')
                *i += 1;
        c = gnbchar(s, &*i, &c);
        if(c == '+' || c == '-')
                *relren = TRUE;
        else
                *relren = FALSO;
        *ren = ponparam(s, &*i, -MAXRENS+1, MAXRENS-1, 0);
        if(gnbchar(s, &*i) == ']')
                *i += 1;
}
```

stoc comienza examinando la referencia a celda en el caracter i-ésimo de la cadena **s**. Devuelve los dos números encontrados como los enteros **col** y **ren**. Los parámetros booleanos **relcol** y **relrow** dicen a la rutina que los llamó si las coordenadas del renglón o de la columna extraidas son relativas o no.

La rutina **ponparam** es llamada por **stoc** para realizar una función frecuente: extraer un parámetro entero de un cadena y comprobarlo para asegurarse de que se encuentra entre los límites máximo y mínimo, si no, se asigne un valor por defecto.

```
/* Extrae un entero de una cadena */
ponparam(s, i, min, max, def);
char s[], *i;
int min, max, def;
{
      int param;

      if(stoi(s, &*i, &param))
            param = def;
      else if((param < min) || (param > max))
            param = def;
      return param;
}
```

ponparam hace uso de la función **stoi** para convertir una cadena en un entero:

```
/* Convierte una cadena en un entero */
#define MDIV10  3276      /* MAXINT div 10 */
#define MMOD10  7         /* MAXINT mod 10 */
stoi(s, i, ent)
char s[];
int *i, *ent;
{
      char c,ok, negnum;
      int digit;

      *ent = 0;
      ok = TRUE;
      negnum = FALSO;
      c = gnbchar(s, &*i);
      if(c == '+' || c == '-'){
```

```
        if(c == '-')
                negnum = TRUE;
    *i += 1;
    }
    c = gnbchar(s , &*i);
    while(c >= '0' && c <= '9'){
        if(ok){
            digit = c - 48;
            if(*ent > MDIV10 || (*ent == MMOD10 && digit
                                                    > MMOD10))
                ok = FALSO;
            else
                *ent = 10 * (*ent) + digit;
        }
    *i += 1;
    c = gnbchar(s, &*i);
    }
    if(negnum)
        *ent = -(*ent);
    return ok;
}
```

Volviendo a la función **evalfact** si en lugar de un paréntesis o corchete izquierdo **evalfact** ve un identificador, llama a **evalid** para evaluar el identificador.

```
/* Obtiene el identificador de la cadena */
buscaid(s, i, id)
char s[], id[], *i;
{
    char c;
    strcpy(id, "");
    c = gnbchar(s, &*i);
    if((c >= 'A' && c <= 'Z') || (c >= 'a' && c <= 'z')){
        addchar(id, c,IDSIZE);
        *i += 1;
        c = gnbchar(s ,&*i);
        while((c >= 'A' && c <= 'Z' || (c >= 'a' && c <='z')
                                || ( c >= '0' && c <= '9')){
            addchar(id, c, IDSIZE);
            *i += 1;
            c = gnbchar(s, &*i);
        }
    return TRUE;
    }
```

```
        else
                return FALSO;
}
```

En otros casos, **evalfact** llama a **stox,** quien intenta interpretar el factor como un número. He aquí **stox** :

```
/* Convierte una cadena en un número real */
#define RESIZE 15    /* Tamaño máximo de un número en la fórmula */
stox(s, i, x)
char s[], *i;
double *x;
{
        int nsig, expo, i;
        char negepart,negnum;
        char c;
        double base, nfrac,frac;

        nsig = base = frac = expo = 0;
        negepart = negnum = FALSO;
        nfrac = 1;
        c = gnbchar(s, &*i);
        if(c =='+' || c == '-'){
                if(c == '-')
                        negnum = TRUE;
                *i += 1;
        }
        while(gnbchar(s, &*i) == '0')
                *i += 1;
        c = gnbchar(s, &*i);
        while(c >= '0' && c <='9'){
                if(nsig < RESIZE)
                        base = 10 * base + (c-48);
                        nsig += 1;
                        *i += 1;
        c = gnbchar(s, &*i);
        }
        if( == '.'){
                *i += 1;
                nsig = 0;
                c = gnbchar(s, &*i);
                while(c >= '0' && c <= '9'){
                        if(nsig < RESIZE)
                                frac = 10 * frac + (c - 48);
                                nfrac = nfrac * 10.0;
                                nsig += 1;
                *i += 1;
```

```
            c = gnbchar(s, &*i);
            }
        base = base + (frac / nfrac);
        }
        c = gnbchar(s, &*i);
        if(c == 'E' || c == 'e'){
            *i += 1;
            c = gnbchar(s, &*i);
            if(c == '+' || c == '-'){
                if(c == '-')
                    negepart = TRUE;
            *i += 1;
            }
            c = gnbchar(s, &*i);
            while(c >='0' && c <='9'){
                if(maig < RESIZE)
                    expo = 10 * expo + ( c - 48);
            *i += 1;
            c = gnbchar(s, &*i);
            }
            if(negepart){
                for(i = 1; i <= expo; i++)
                base = base /10.0;
            }
            else{
                for( i = 1; i <= expo; i++)
                    base = base * 10.0;
            }
        *x = base;
        if(negnum)
            base = - base;
        return ok;
}
```

ARITMETICA

En esta sección describiremos las rutinas que realizan las operaciones aritméticas. De alguna manera estas rutinas deben estar diseñadas para detectar los errores que se cometen al efectuar una operación como lo son desbordamientos de cantidades y divisiones entre cero:

Cabe aclarar que se ha utilizado la función **pow** proporcionada por librería de funciones del sistema TURBO-C. Su diseño de las

demás es la siguiente:

```
/* Suma dos numeros reales */
xadd(arg1, arg2, x)
double *arg1, arg2, *x;
{
      *x = *arg1 + arg2;
      return OK;
}



/* Resta dos numeros reales */
xsub(arg1, arg2 x)
double *arg1, arg2, *x;
{
      arg2 = - arg2;
      return   xadd(*arg1, arg2, &*x);
}

/* Multiplica don nmeros reales */
xmul(arg1, arg2, x)
double *arg1, arg2, *x;
{
      *x = *arg1 * arg2;
      return Ok;
}



/* Divide dos numeros reales */
xdiv(arg1, arg2, x)
double *arg1, arg2, *x;
{
      if(arg2 == 0)
            return ZERODIVIDE;
      else{
            *x = *arg1 / arg2;
            return OK;
      }
}
```

LAS FUNCIONES SUM Y PROM

MicroCtiene únicamente dos funciones: **SUM** y **PROM** . La sintaxis de **SUM** y **PROM** es como sigue:

SUM (<arg1>, <arg2>, . . . <argN>)

PROM (<arg1>, <arg2>, . . . <argN>)

SUM acepta un número arbitrario de argumentos separados por comas; el resultado de la función es la suma de los valores de todos estos argumentos. Por su parte **PROM** obtiene el promedio de los argumentos, es decir, suma N argumentos y los divide entre N :

(< arg1 > + < arg2 > + . . . + < argN >) / N

Cada argumento puede ser o una expresión o un rango de celdas. Si el argumento es un rango de celdas, su valor es la suma de todas las celdas dentro del rango.

Un rango de celdas se construye con dos referencias a celdas separadas por dos puntos :

[<col1>,<ren1>]:[<col2>,<ren2>]

El rango define (en general) un área rectangular de 1 a hoja, con **[<col1>,<ren1>]** en la esquina superior izquierda y **[<col2>,<ren2>]** en la esquina inferior derecha. Las sumas sobre un renglón puede obtenerse haciendo **<ren1> = <ren2>** ; las sumas sobre una columna simple puede obtenerse haciendo **<col1> = <col2>** . Las referencias a celda pueden ser absolutas o relativas. Si no existe la segunda referencia a celda, se considera que el rango sólo contiene la celda simple **[<col1>,<ren1>]**.

Para ver cómo funciona esto, consideremos primero **evalid** :

```
/* Evalua un identificador */
evalid(s, i, id, x)
char s[], id[], *i;
double *x;
{
      char c;
      enum xresult evalue;
```

```
    if(!strcmp(id, "SUM"))
        evaluo = evalsum(s, &*i, &*x);
    else if(!strcmp(id, "PROM")){
        evaluo = evalsum(s, &*i, &*x);
        *x = *x / n_arg;
    }
    else
        evaluo = INVALID; /* Funcion desconocida */
    return evaluo;
}
```

evalid comprueba si el nombre de la funcipn extraida por **findid** es **SUM** o **PROM** . Si no es ninguna de las dos, **evalid** considera esto como un intento de unas una función sin definir; presenta una observación sobre lo ocurrido y se salta la lista de argumentos de la función sin definir si la tuviera.

Si se encontró la palabra **SUM** ,**evalid** llama a **evalsum** para evaluar la función; si se encontró la palabra **PROM** tambien llama a **evalsum** para sumar los argumentos y simplemente los divide entre **n_arg** para obtener su promedio:

```
/* Evalua una suma de argumentos */
evalsum(s, i, x)
char s[], *i;
double *x;
{
    enum xresult status;
    char c;
    double x1;
    char i;

    *x = zero;
    status = OK;
    if(gnbchar(s, &*i) == '('){
        while(( status == OK) && findarg(s, &*i, arg)){
            i = 0;
            if(gnbchar(arg, &i) == '(')
                status =summatrix(arg, &i, &x1);
            else
                status = evalexpr(arg, &i, &x1);
            if(status == OK)
                status = xadd(&*x, x1, &*x);
        }
```

```
        if(gnbchar(s, &*i) == ')')
            *i += 1;
        else

            remark("Falta parentesis derecho");
    }
    return status;
}
```

evalsum llama a **buscarg** para extraer los sucesivos argumentos de la lista de argumentos (si existe), para evaluarlos uno a uno. Si el argumento comienza con un corchete izquierdo **[** **i** **]**, **evalsum** lo evalúa como un rango de celdas llamando a **summatrix** ; en los demás casos llama a **evalexpr** para evaluar el argumento como un función.

La función **buscarg** es llamada por **evalid** y **evalsum** ; comienza examinando un argumento de la función en el caracter **(i+1)-ésimo** dela cadena **s** , y lo devuelve como la cadena **arg**. Si no se encuentra ningún argumento en la posición especificada en **s**, **buscarg** devuelve **FALSO** como resultado de su función; en los demás casos devuelve **TRUE.** Ella deja a **i** apuntando al caracter que delimita el argumento, bien sea una coma o un paréntesis derecho:

```
/* Extrae el argumento de una cadena */
buscarg(s, i, arg)
char s[], arg[], *i;
{
    int nparen, nbrak;
    char done,c;

    strcpy(arg, "");
    c = gnbchar(s, &*i);
    if(c == ')' || c == 0)
        return FALSO;
    else{
        nparen = nbrak = 0;
        done = FALSO;
        do {
            *i += 1;
            c = gnbchar(s, &*i);
```

```
            if((c == ',' || c == ')') && nparen <= 0 && nbrak
                                                    <= 0 )
                  done = TRUE;
            else{
                  addchar(arg, c, MAXSTR);
                  if(c == '(')
                        nparen += 1;
                  else if(c == ')')
                        nparen -= 1;
                  else if(c == '[')
                        nbrak += 1;
                  else if(c == ']')
                        nbrak -= 1;
            }
      }while(!done);
      return TRUE;
      }
}
```

Hay que hacer notar que **buscarg** no es tan fácil como podría esperarse en un principio. No puede recorrer la cadena buscando una coma o un paréntesis derecho para terminar el argumento, debido a que un argumento puede contener una coma o un paréntesis derecho. Por ejemplo, en la fórmula:

SUM([1,1]:[1,10], (1+(1+3)), [43,38])

buscarg debe devolver la cadena "**[1,1]:[1,10]**" como primer argumento, no "**[1,**" como devolvería si simplemente buscara hasta la primera coma. Igualmente, el segundo argumento tiene dos paréntesis derechos, pero ninguno de ellos son delimitadores de argumento.

La solución que **buscarg** utiliza es contar el número de paréntesis izquierdos sin equilibrar (**nparen**) y corchetes izquierdos sin equilibrar (**nbrak**) encontrados en el argumento. El argumento sólo termina cuando se encuentra una coma o un paréntesis derecho y los contadores **nparen** y **nbrak** son ambos menores o iguales a cero.

Lo único que nos queda es el procedimiento **summatrix**:

```
/* Suma la matriz de celdas */
summatrix(s, i, total)
char s[], *i;
double total;
{
        enum xresult status;
        int x1,x2,y1,y2,x;
        struct celdptr *cp;
        char done, c;

        n_arg = 0;
        *total = zero;
        status = OK;
        obtcoords(s, &*i, &x1, &y1);
        if(gnbchar(s, &*i) == ':'){
                *i += 1;
                obtcoords(s, &*i, &x2, &y2);
        }
        else{
                x2 = x1;
                y2 = y1;
        }
        if((x1 >= 1) && (x1 <= MAXCOLS) && (y1 >= 1) && (y1 <=
          MAXRENS) && (x2 >=1) && (x2 <= MAXCOLS) && (y2 >=1) && (y2
          <= MAXRENS)){
                x = x1;
                while((x <= x2) && (status == OK)){
                        cp = colptr(x);
                        done = FALSO;
                        while((cp != NULL) && (status == OK) && (!done)){
                                if(cp->celdren >= y1)
                                        if(cp->celdren <= y2){
                                                status=xadd(&*total,cp->fp->celdval
                                                                , &*total);
                                                n_arg += 1;
                                        }
                        }
                        else
                                done = TRUE;
                        cp = cp->downptr;
                        }
                x += 1;
                }
        }
        return status;
}
```

Este procedimiento extrae las coordenadas del rango de celdas de la cadena de argumentos llamando a **obtcoord** dos veces (o una sola vez, si no se han encontrado dos puntos después de la primera referencia a celda). Luego después de comprobar que el rango entero está contenido en la hoja, suma las celdas dentro del rango acumulando las sumas en la variable **total.**

3.6 COMANDOS DE MicroC

El último objetivo importante del diseño de **MicroC** es implementar las órdenes de la tabla 3.1.1. La rutina **docomando** es llamada desde el programa principal :

```
/* Evalua una orden del usuario */
docomando(ch)
char ch;
{
      unsigned long ml;

      if (ch == '/') {
            eraseline(inpline);
            posstr("Comandos:Memoria/Ximprime/Recupera/Archiva/Copia
                  /Borra/Inserta/Salir: 0, inpline);
            posstr("Para      ejecutar      un      comando      teclee      la      letra
                  inicial", 10, 23);
            gotoxy(70, 22)
            ch = getche();
            eraseline(23);
            eraseline(inpline);
            switch (ch) {
                  case 'a':
                  case 'A': archiva();
                              break;
                  case 'r':
                  case 'R': recupera();
                              break;
                  case 'c':
                  case 'C': copias();
                              break;
                  case 'b':
                  case 'B': borrar();
                              break;
                  case 'x':
                  case 'X': imprime();
                              break;
                  case 'i':
                  case 'I': insertar();
                              break;
                  case 's':
                  case 'S': salir();
                              break;
                  case 'm':
                  case 'M': despmem();
```

```
                        break;
                }
        }
        else if (ch == '!') {
                recalc();
                disphoja(firstcol, firstrow);
        }
        else if (ch == '>') {
                posstr("Ve a", 0, inpline);
                posstr("Presione ENTER al introducir cada coordenada",
                        8, 29);
                getacc(6,inpline, 1, MAXCOLS,curscol, 1, MAXRENS,
                        currow, &curscol, &currow);
                eraseline(29);
                eraseline(inpline);
        }
        setcursor(curscol, currow);
}
```

docomando es llamada con el parámetro **ch** ; esto es o una exclamación < ! >, un signo < > >, o un slash < / >. Una señal de exclamación recalcula y vuelve a presentar la hoa entera; esto se realiza llamando a **recalc y disphoja** , las cuales hemos visto anteriormente.

La tecla "mayor que" se utiliza para moverse a una celda específica. **MicroC** solicita que el usuario introduzca las coordenadas del renglón y la columna de la celda, actualiza las variables globales **curscol** y **cursrow,** luego llama a **setcursor** para trasladar el cursor a esa celda, colocando adecuadamente la ventana. (Realmente, todas las órdenes terminan con una llamada a **setcursor** para asegurar que el cursor está en el lugar correcto de la pantalla).

getacc se llama para obtener del usuario las coordenadas de columna y renglón. Los límites inferior y superior de las coordenadas y las coordenadas por defecto se pasan a la rutina. Las coordenadas se devuelven como los parámetros **col** y **row** . He aquí **getacc** :

```
/* Obtiene las coordenadas de una celda */
getacc(crtcol, crtrov, xmin, xmax, xdef, ymin, ymax, ydef, col,
                                                              rov)
int crtcol, crtrov, xmin, xmax, xdef, ymin, ymax, ydef,*col,*rov;
{
      int column, reng;

      posstr("   ,    ", crtcol, crtrov);
      gotoxy(crtcol + 1, inpline);
      scanf("%d", &column);
      if(column < xmin || column > xmax)
            *col = xdef;
      else
            *col = column;
      gotoxy(crtcol + 4, inpline);
      scanf("%d", &reng);
      if(ren < ymin || reng > ymax)
            *rov = ydef;
      else
            *rov = reng;
}
```

Una vez que hemos considerado las órdenes de recálculo y de "ir a", podemos tener las órdenes con "slash". Si el usuario imprime un slash, **docomando** le pregunta por una de las ocho órdenes como sigue:

Comandos: Memoria/Ximprime/Recupera/Archiva/Copia/Borra/Inserta

/Salir:

Después que el usuario escribe la primera letra de cada uno de estos datos, **docomando** llama a la rutina coreespondiente para ejecutar la órden.

ÓRDENES FÁCILES

Siempre que nos enfrentamos con la necesidad de diseñar un gran número de rutinas, todas de similar importancia, una buena regla es **hacer primero las más fáciles** . Siguiendo esta regla, la primera rutina que hay que escribir es **salir**:

```
/* Realiza la orden de salir */
salir()
{
      char ch;

      posstr("Salir ... Ya salvaste tu hoja ?(S/N)",0,22);
      ch = getche();
      if(ch == 'n' || ch == 'N')
            alldone = FALSO;
      eraseline(22);
}
```

Recordemos que el ciclo de control principal **MicroC** se sale cuando **alldone** es **FALSO**.

La rutina **despmem** es casi igual de fácil. Presentando la memoria libre para ser utilizada por la hoja de trabajo.

```
/* Presenta la memoria libre */
despmem()
{
      eraseline(23);
      gotoxy(18,23);
      ml = coreleft();
      printf("%u  bytes libres  (Presiona una tecla)",ml);
      getche();
      eraseline(23);
}
```

ORDEN DE COPIAR

La copia (También llamada replicación en algunos programas comerciales) es probablememte una de las órdenes de mayor utilidad en **MicroC** . En términos generales, la orden se utiliza para hacer una o máscopias de un rango de celdas especifacadas por el usuario. Se pueden copiar celdas con rótulos, celdas con fórmulas o ambas. Nuestro objetivo es hacer esta funcion lo más flexible

posible.

Cuando se da una orden de copia, el programa escribe

Copia de [,]:[,] a [,]:[,]

En palabras, en programa pide al usuario que introduzca primero un rango fuente de celdas del cual copiar, luego un rango destino de celdas enlas que se copia.

Los rangos pueden especificar cualquier sección rectangular de la hoja, como lo vimos en la discusión de la función **SUM**. Las celdas individuales, los renglones simples y las columnas simples, son casos especiales de tals secciones rectangulares. En la siguiente discusión, usaremos el término " **matriz** " para denominar una función rectangular general.

Puesto que un rango puede especificar una celda, una columna, un renglón, o (en el caso más general) una matriz, tenemos un total de 16 (4 veces 4) casos diferentes (celdas copiadas en celdas, celdas copiadas en renglones, celdas copiadas en columnas y así sucesivamente). Consideremos aquí los casos más útiles.

1. Copia de una celda en una celda. En este caso los rangos fuente y destino son celdas simples. Ejemplo:

Copia de [1,15]:[1,15] a [10,4]:[10,4]

simplemente copia la celda [1,15] en [10,4].

2. Copia de una columna en una celda. En este caso el usuario especifica el rango fuente como una columna introduciendo las coordenadas de la primera y la última columnas. El rango de destino se introduce como una celda simple en la que se copiará la primera columna; el resto de la columna se copia en la celda que a continuación del punto. Ejemplo:

Copia de [2,10]:[2,15] a [5,5]:[5,5]

copia las seis celdas desde [2,10] a la [2,15] (algunas o

tadas de las cuales pueden estar vacías) en una columna que comienza en [5,5] y sigue hasta la [5,10].

3. Copia de un renglón en una celda. Este es un caso similar al 2, excepto que estamos copiando un renglón en vez de una columna. El usuario especifica las coordenadas izquierda y edrecha del renglón que va a ser copiado. El rango destino es la celda en la que se copian las celdas más a la izquierda del renglón. El resto se copia en las celdas que se necuentran a la derecha de la celda especificada.

4. Copia de una matriz en una celda. Esta es simplemente una generalización de los tres casos anteriores. En vez de una celda sencilla, una columna sencilla o un renglón, se da para el rango fuente completo de posiciones. He aquí un ejemplo:

Copia de [1,1]:[10,10] a [21,21]:[21,21]

Esto copia las cien celdas de la matriz 10x10 con la esquina superior izquierda [1,1] y la esquina inferior derecha [10,10] (algunas de estas celdas pueden estar vacùas) en la matriz del mismo tamaño que tiene la esquina superior izquierda [21,21].

5. Copia de una celda en una columna. Aquí el rango fuente es, al igual que en caso 1, una sola celda.

El rango destino se introduce como las coordenadas del principio y el final de la columna e la que el usuario quiere copiar la celda.

Copia de [5,10]:[5,10] a [10,1]:[10,10]

copia la celda de la posición [5,10] en las diez celdas [10,1],[10,2], ... ,[10,10].

6. Copia de una celda en un renglón. Este caso es análogo al caso 5. El usuario especifica una celda simple como fuente y las coordenadas izquierda y dereha del renglón en la que sre va a

copiar la celda.

7. Copia de una celda en una matriz. Este caso es una generalización de los casos 5 y 6. La fuente es una única celda y el usuario especifica una matriz rectangular como el rango destino.

8. Copia de una columna en un renglón. Aquí se especifica una columna como rango fuente y un renglón como destino. Es una generalización del caso 2. En lugar de copiar la columna fuente especificada a partir de una celda, con lo que se realiza una copia de la columna, el usuario especifica un reglón horizontal de posiciones de comienzo para las columnas en las que se copia, con lo cual resulta más de una copia de la columna.

9. Copia de un renglón en una coumna. Este caso es análogo al caso 8 y es una generalización del caso 3: varias copias del renglón fuente se realizan a partir de una columna vertical.

El resto de los casos (columna a columna, columna a matriz, renglón a renglón, renglón a matriz, matriz a columna, matriz a renglón y matriz a matriz) son generalizaciones de estos nueve casos. (Dicho de otra forma, todos los casos mencionados anteriormente son casos especiales de la copia de una matriz en una matriz). Francamente, es difícil ver cómo algunos de estos casos podrían utilizarse en aplicaciones prácticas, pero posiblemente el usuario puede encontrar esta flexibilidad útil para algo. Es un defecto reducir artificialmente las carcaterísticas de un programa aunque no podamos imaginasr exactamente cómo podría utilizarse.

He aquí **copias** en lenguaje **C** :

```
/* Realiza la orden de copiar */
copias()
{
      int x1,x2,x3,x4,y1,y2,y3,y4,xdest,ydest;
      char success;

      posstr("Copia de [  ,  ,  ] a [  ,  ,  ]",0,22);
      posstr("Presione ENTER despues de cada coordenada",5,23);
```

```
        getacc(10,22,1,MAXCOLS,curscol,1,MAXRENS,cursrov,&x1,&y1);
        getacc(19,22,x1,MAXCOLS,x1,y1,MAXRENS,y1,&x2,&y2);
        getacc(31,22,1,MAXCOLS,x1,1,MAXRENS,y1,&x3,&y3);
        getacc(40,22,x3,MAXCOLS,x3,y3,MAXRENS,y3,&x4,&y4);
        eraseline(23);
        center("Copiando . . .",24);
        success = TRUE;
        xdest = x3;
        while(xdest <= x4 && success){
              ydest = y3;
              while(ydest <= y4 && success){
                    success = copymatrix(x1,y1,x2,y2,xdest,ydest);
                    ydest += 1;
              }
        xdest += 1;
        }
        eraseline(24);
        if(!success)
              memout();
        desphojalfirstcol,firstrov);
        eraseline(22);
}
```

copias llama a **getacc** cuatro veces para obtener los rangos fuentes y destino de la copia; el rango fuente es [x1,y1]:[x2,y2] y el rango destino es [x3,y3]:[x4,y4].

copias llama a **copymatrix** para copiar el rango fuente en una celda perteneciente al rango destino. **copymatrix** devuelve **FALSO** si el programa se sale de la memoria mientras e hace la copia; esto hace que inmediatamente termine el proceso de la copia.

La rutina **copymatrix** recorre el rango fuente, copiando celda a celda no vacía en el rango destino en la posición espacificada.

```
/* Copia una matriz de celdas en su destino */
copymatrix(x1,y1,x2,y2,xdest,ydest)
int x1,y1,x2,y2,xdest,ydest;
{
      int x,y;
      struct celdptr *cp;
      char success, done;

      x = x1;
      success = TRUE;
      while(x <= x2 && success){
```

```
        cp = colptr(x);
        done = FALSO;
        while(cp != NULL && !done && success){
              y = cp->celdren;
              if(y >= y1)
                    if(y <= y2)
                          success = copyceld(cp,xdest+(x-x1),ydest
                                                          +(y-y1));
                    else
                          done = TRUE;
        cp = cp->downptr;
        }
      x += 1;
      }
      return success;
}
```

copymatrix copia el rango fuente por columna; esta elección del orden sólo interesa si el usuario, por alguna razón específica, copia parámetros que hacen que dos o más elementos fuente se copien en las misma delda destino.

Cada celda no vacía se copia en otra posición con **copyceld**":

```
/* Copia una celda en la posicion especificada */
copyceld(cp, col, row)
struct celdptr *cp;
int col, row;
{
      struct celdptr *newcp;
      char success;

      success = TRUE;
      if(col >= 1 && col <= MAXCOLS && row >=1 && row <= MAXRENS){
            newcp = busceld(col, row);
            if(newcp != NULL)
                  eraseceld(newcp);
            newcp = (struct celdptr *) malloc (sizeof(struct celdptr
                                                                ));
            success = storeceld(col, row, newcp);
            if(success)
                  if(cp->display != NULL){
                    newcp->display = (char *) malloc (strlen(cp->
                                                          display));
                    success = storestr(cp->display, newcp->display);
                  }
                  if(cp->fp != NULL){
```

```
                    newcp->fp = (struct formptr *) malloc(sizeof(
                                                    struct celdptr));
                    success = storeform(newcp->fp);
                    if(success)
                        newcp->fp->celdval = cp->fp->celdval;
                    if(cp->fp->formula != NULL){
                        newcp->fp->formula = (char *) malloc (strlen
                                                (cp->fo->formula));
                        success = storestr(cp->fp->formula,newcp->fp
                                                    ->formula);
                }
            }
        }
        return success;
}
```

Esta rutina comprueba en primer lugar las coordenadas de la celda para asegurarse de que están dentroi de los límites de la hoja. Si una celda ocupa actualmente la posición en que ha de copiarse, se borra. Luego se crea una nueva celda con **storeceld**, y los contenidos de la celda fuente se copian en la celda destino utilizando los procedimientos de asignación de memoria apropiados. Al igual que **copymatrix**, **copyceld** devuelve **TRUE** si la copia se realiza con éxito; en los demás casos, devuelve **FALSO**.

ORDEN DE BORRAR

Al igual que con una hoja real, los usuarios de **MicroC** tendrán que borrar alguna vez información de la hoja electrónica. Esto se realiza con la orden de borrar **< /B >**. Cuando se da la orden, el programa escribe:

Borrar: Una_celda, Columnas, Renglones o la Hoja. (U/C/R/H):

El usuario escribe una **< U >** para borrar la celda sobre la que actualmente se encuentra el cursor, **< C >** para borrar varias columnas, **< R >** para borra varios renglones y **< H >** para borrar

toda la información de la hoja. En los casos de borrar columans o renglonmes, se pregunta al usuario :

Cuantas columnas:

o

Cuantos renglones:

El usuario escribe entonces el número de columnas que ha de borrarse. Si el usuario escribe un 0 la hoja se deja como está. Las columnas que se borran son las que están a la derecha de la posición actual del cursor incluyendo la columna en la que está el cursor. después de que se borra, todas las columnas a la derecha de las columnas suprimidas se mueven hacia la izquierda tantas columnas como se suprimieron.

La supresión de renglones es igual que la de columnas. **MicroC** pide al usuario que escriba el número de renglones que han de suprimirse, y si éste escribe un valor de 0 se deja la hoja como está. Los renglones se borran hacia abajo incluyendo el renglón en la que está el cursor. Los rengloes no suprimidos por debajo del cursor se mueven hacia arriba un número de renglones igual a los suprimidos.

En el caso de borrar la hoja entera, hay que tener cuidado al tomar esta decisión. Ya que borrar accidentalmente una hoja sería uan catástrofe.

La rutina **borrar** maneja el borrado de la celda y la hoja y delega el borrado más complejo de columnas y renglones a **erasecols** y **eraserows**. He aquí **borrar**:

```
/* Orden de borrar */
borrar0
{
      char ch;
      int c;
      struct celdptr *cp;

      posstr("Borrar: Una_celda,Columnas,Renglones,Hoja. (U/C/R/H):
                                                          ",0,22);
```

```
        ch = get_key();
        eraseline(22);
        switch(ch){
                case 'U': erasecold(curcp);
                          desprev(curarov);
                          eraseline(22);
                          break;
                case 'C': erasecols();
                          break;
                case 'R': eraserows();
                          break;
                case 'H': for(c = 1; c <= MAXCLOS; c++){
                                cp = colptr(c);
                                while(cp != NULL){
                                      erasecold(cp);
                                      cp = cp->downptr;
                                }
                          }
                          desphoja(firstcol,firstrov);
                          eraseline(21);
                          break;
                default: erasecold(curcp);
                          desprev(curarov);
                          eraseline(21);
                          break;
        }
}
```

Consideremos primero el borrado de columnas :

```
/* Suprime columnas de la hoja */
erasecols()
{
        int n,c;
        struct coldptr *cp;

        posstr("Cuantas columnas:",0,22);
        gotoxy(19,22);scanf("%d",&n);
        if(n > 0){
              center("Borrando ...",21);
              for(c = curscol; c <= (curscol+n-1); c++){
                    cp = colptr(c);
                    while(cp != NULL){
                          erasecold(cp);
                          cp = cp->downptr;
                    }
              }
              for(c = curscol; c <= MAXCOLS-1; c++){
```

```
                colptr[c] = colptr[c+1];
                cp = colptr[c];
                while(cp != NULL){
                        cp->coldcol = c;
                        cp = cp->dovnptr;
                }
        }
        colptr[MAXCOLS] = NULL;
        ajustcol(n);
        eraseline(21);
        labelcols(firstcols);
        deephoja(firstcol, firstrov);
    }
    eraseline(22);
    eraseline(23);
}
```

Para cada columna que ha de borrarse, **erasecols** llama repetidamente a **eraseceld** para borrar la celda superior de la columna hasta que ésta quede vacía. A continuación **erasecols** mueve todas las columnas que están a la derecha de las columnas suprimidas, **n** columnas a la izquierda. Esto puede hacerse con gran sencillez reasignando los punteros de columna y el resto de las variables de información de columna. Debemos tambien ir por cada columna movida para reajustar el campo **celdcol** de las celdas. El borrado de **n** columnas deja **n** columnas vacías en la parte derecha de la hoja, lo que se indicará porque sus punteros estarán a **NULL**.

La tarea que le queda a **erasecols** es ajustar las referencias a celdas en todas las fórmulas de las celdas para que reflejen la nueva colocación de las columnas. Esto lo realiza la función **ajustcol**; recorre odas las celdas con formulas de la hoja y cambia sus referencias a celda, si es necesario, para que se refieran a las mismas celdas que antes.

He aquí **ajustcols:**

```
/* Ajusta las referencias a columnas en las formulas de la hoja */
ajustcols(n)
int n;
{
```

```
int c, col, row;
struct celdptr *cp;
char s[80], st[80];
char cambio,relcol,relrow,ch. c1,i;

for(c = 1; c <= MAXCOLS; c++){
      cp = colptr(c);
      while(cp != NULL){
            if(cp->fp != NULL)
                  if(cp->fp->formula != NULL){
                        strcpy(s,"");
                        i = 0;
                        cambio = FALSO;
                        c1 = gnbchar(cp->fp->formula, &i);
                        while(c1 != 0){
                              if(c1 != 'T'){
                                    addchar(s, c1, MAXSTR);
                                    i += 1;
                              }
                              else{
                                    atoc(cp->fp->formula,&i,&col,
                                          &row,&relcol,&relrow);
                                    if(relcol){
                                       if((c >= curscol) && ( c+n+
                                                col < curscol)){
                                          cambio = TRUE;
                                          col += 1;
                                       }
                                       else if((c < curscol) && (
                                                c+col >= curscol)){
                                                cambio = TRUE;
                                                col = col - n;
                                       }
                                    }
                                    else if(col >= curscol){
                                          col = col - n;
                                          cambio = TRUE;
                                    }
                                    ctoa(col, row, relcol,relrow,st);
                                    strcat(s, st);
                              }
                        c1 = gnbchar(cp->fp->formula, &i);
                        }
                        if(cambio)
                              strcpy(cp->fp->formula, s);
                  }
            cp = cp->downptr;
            }
      }
}
```

ajustclos examina todas las fórmulas de la hoja, cambiando todas las referencias a columnas que sean necesarias. Si se cambia una fórmula, la antigua versión se modifica copiando en ella misma la nueva fórmula

La función **eraserows** es análoga a **erasecols** ; es un poco más sencillo porque no hay que considerar formatos de columna:

```
/* Suprime renglones de la hoja */
eraserows()
{
        int n,r;
        struct celdptr *cp;
        posstr("Cuantos renglones: ",0,22);
        gotoxy(24,22);
        scanf("%d", &n);
        if(n > 0){
                center("Borrando . . .",24);
                for(r = currow; r <= (currow + n - 1); r++){
                        cp = renptr(r);
                        while(cp != NULL){
                                eraseceld(cp);
                                cp = cp->rightptr;
                        }
                }
                for(r = currow; r <= MAXRENS-1; r++){
                        rowptr[r] = rowptr[r+1];
                        cp = rowptr[r];
                        while(cp != NULL){
                                cp->celdrow = r;
                                cp = rightptr;
                        }
                }
                rowptr[MAXRENS] = NULL;
                ajustrens(n);
                eraseline(24);
                desphoja(firstcol, firstrow);
        }
        eraseline(22);
        eraseline(24);
}
```

También necesitamos una función **ajustrens** exactamente análogo a **ajustcols**:

```
/*  Ajusta las referencias a renglones de las formulas de la hoja
*/
ajustrens(n)
int n;
{
        int r, col, row;
        struct celdptr *cp;
        char s[80], s1[80];
        cambio, relcol, relrow, ch, i, ch1;

        for(r=1; r <= MAXRENS; r++){
              cp = renptr(r);
              while(cp != NULL){
                    if(cp->fp != NULL)
                          if(cp->fp->formula != NULL){
                             strcpy(s, "");
                             i = 0;
                             cambio = FALSO;
                             ch1 = gnbchar(cp->fp->formula,&i);
                             while(ch1 != 0){
                                 if(ch1 != '('){
                                    addchar(s, ch1, MAXSTR);
                                    i +=1;
                                 }
                                 else{
                                    atoc(cp->fp->formula, &i, &col, &row,
                                                          &relcol, &relrow);
                                  if(relrow){
                                       if((r >= currow) && (r+n+row <
                                                          currow)){
                                          cambio = TRUE;
                                          row += n;
                                       }
                                       else if(( r >= currow) && (r+row >=
                                                          currow)){
                                          cambio = TRUE;
                                          row  -= n;
                                       }
                                       else if(row >= currow){
                                        cambio = TRUE;
                                        row -= n;
                                       }
                                       ctoa(col, row, relcol, relrow,s1);
                                       strcat(s,s1);
                                 }
                                ch1 = gnbchar(cp->fp->formula, &i);
```

```
                    }
                if(cambio)
                    strcpy(cp->fp->formula, s);
                }
            cp = cp->rightptr;
            }
        }
    }
}
```

ORDEN DE INSERTAR

La orden de insertar (</I>) se utiliza para insertar varios renglones o columnas en blanco en la hoja.

Los renglones o columnas existentes se mueven para hacerle sitio a las nuevas. Esto es análogo a "cortar y pegar" en la edición de una hoja de papel, es decir, hacer sitio para la información que se necesita insertar en la hoja para escribir sobre la información que ya se ha introducido. Cuando se ha dado la orden **MicroC** pregunta:

Insertar: Columnas o Renglones (C/R) :

El usuario escribe < **C** > para introducir columnas y < **R** > para los renglones. He aquí **insertar** :

```
/* Insertar renglones o columnas en la hoja */
insertar()
{
    char ch;

    posstr("Inserta:Columnas o Renglones (R/C): ", 0, inpline);
    ch = get_key();
    eraseline(inpline);
    switch (ch) {
        case 'r':
        case 'R': insertrows();
                  break;
        case 'c':
```

```
            case 'C': insertcols();
                      break;
        }
}
```

insertar llama a **insertcols** para insertar columnas, y a **insertrows** para insertar renglones. Examinemos primero **insertcols** :

```
/* Inserta columnas */
insertcols()
{
        int c, n;
        struct celdptr *cp;
        char ok;

        posstr("Insertar, cuantas columnas?:", 0, inpline);
        gotoxy(29, inpline); scanf("%d", &n);
        if (n > 0) {
                ok = TRUE;
                c = MAXCOLS;
                while ( c >= (MAXCOLS - n + 1) && ok) {
                        if (colptr[c] != NULL)
                                ok = FALSO;
                        c -= 1;
                }
                if (!ok) {
                        center("Precaucion: insercion borrando celdas no
                                vacias", statline);
                        ok = TRUE;
                        eraseline(statline);
                }
                if (ok) {
                        center("Insertando. . .", statline);
                        for (c = MAXCOLS; c >= (MAXCOLS - n + 1); c--)
                                while (colptr[c] != NULL)
                                        eraseceld(colptr[c]);
                        for (c = MAXCOLS; c >= (curscol + n); c--) {
                                colptr[c] = colptr[c -1];
                                cp = colptr[c];
                                while (cp != NULL) {
                                        cp->celdcol = c;
                                        cp = cp->downptr;
                                }
                        }
                        for (c = (curscol + n - 1); c >= curscol; c--)
                                colptr[c] = NULL;
                        adjustcols(-n);
```

```
                eraseline(statline);
                labelcols(firstcol);
                desphoja(firstcol, firstrow);
            }
        }
        eraseline(inpline);
        eraseline(23);
}
```

insertcols pregunta primero al usuario el número de columnas en blanco que ha de insertar; si introduce un 0 no se realiza ningún cambio sobre la hoja. Luego, comprueba si la inserción de las columnas podría echar de la hoja a cualquier celda no vacía de las **n** columnas más a la derecha; si es así, pide al usuaio que haga una confirmación antes de continuar.

Luego **insertcols** suprime las **n** columnas más a la derecha de la hoja para hacerle sitio a las nuevas columnas. Si hay alguna celda no vacía en estas columnas, la información de las mismas se pierde. A continuación, todas las columnas desde la posición del cursor hasa la cara derecha de la hoja se mueven **n** columnas a la derecha reasignando los punteros a columnas y las variables asociadas y actualizando el campo **celdcol** de cada celda, igual que en **eraseclos.** Finalmente, los punteros a columna de las columnas nuevamente insertadas se ponen a **NULL.**

Después de que se hayan insertado las **n** columnas en blanco, **insertcols** llama a **ajustcols** para ajustar las referencias a columnas en las fórmulas, también igual que en **erasecols.** En este caso los ajustes son los contrarios a los que se uasn en la supresión de columnas; por tanto, el argumento para **ajustcols** es **-n** en vez de **n.**

insertrows es análogo a **isertcols:**

```
/* Inserta renglones */
insertrows()
{
    int r, n;
```

```
    struct celdptr *cp;
    char ok;

    posstr("Inserta. Cuantos renglones:", 0, inpline);
    gotoxy(28, inpline); scanf("%d", &n);
    if (n >0) {
          ok = TRUE;
          r = MAXRENS;
          while (r >= (MAXRENS - n + 1) && ok) {
                if (renptr[r] != NULL)
                      ok = FALSO;
          r--1;
          }
          if (!ok) {
                center("Precaucion, la insercion sera borrando celdas no
                    vacias", statline);
                ok = TRUE;
                eraseline(statline);
          }
          if (ok) {
                center("Insertando...", statline);
                for (r = MAXRENS; r >= (MAXRENS - n + 1); r--_
                      while(renptr[r] != NULL)
                            erasecell(rovptr[r]);
                for (r = MAXRENS; r >= (curarov + n); r--) {
                      renptr[r] = renptr[r - 1];
                      cp = renptr[r];
                      while (cp != NULL) {
                            cp->celdren = r;
                            cp = cp->rightptr;
                      }
                }
                for (r = (curarov + n - 1); r >= curarov; r--)
                      renptr[r] = NULL;
                ajustren(-n);
                eraseline(statline);
                desphoja(firstcol, firstrov);
          }
    }
    eraseline(inpline);
}
```

CARGA Y ALMACENAMIENTO

Para que sea completamente útil, **MicroC** debe también ser capaz de almacenar la hoja en disco y volverla a cargar en memoria después. Las órdenes de almacenamiento (< **A** >) y carga (< **R** >) realizan esta función; la orden de almacenamiento escribe toda la información sobre la hoja actual en un fichero de disco, de forma que cuando la hoja se recupere con una orden de carga, aparezca (al menos aparentemente) igual a la que se almacenó. Por tanto el fichero de disco debe de alguna forma guardar toda la información referente a la celda.

La función **archivar** pregunta primero el nombre del fichero bajo el que se va a guardar la hoja. Si el programa no es capaz de abrir el fichero de disco,se envía un mensaje de error al usuario y se invalida el comando. En caso contrario se guarda toda la información de la celda en un registro de celda y se escribe en el fichero, continuando así hasta que no haya celdas "no vacías".

He aquí la función **archivar**:

```
/* Archiva la hoja de trabajo en disco */
archivar()
{
      char name[8], status;
      int c;

      posstr("Salvar hoja en el fichero ?",0, inpline);
      gets(nasm);
      if(strlen(name) > 0){
            center(name, 24);puts(" Salvando");
            if((in = fopen(name, "wb")) ==){
                  remark("No pude abrir fichero");
                  eraseline(inpline);
                  return;
            }
            c = 1;
            status = TRUE;
            while(c <= MAXCOLS && status){
                  ap = colptr(c);
                  while(ap != NULL && status){
                        xp.columna = ap->coldcol;
                        xp.renglon = ap->coldren;
                        if(ap->display != NULL){
```

```
                        strcpy(xp.etiqueta, ap->display);
                        strcpy(xp.formula, "");
                        xp.valorf = 0;
                    }
                    else{
                        strcpy(xp.formula, cp->fp->formula);
                        xp.valorf = ap->fp->coldval;
                        strcpy(xp.etiqueta, "");
                    }
                    if(fwrite(&xp,sizeof(struct  mueve), 1, in) !=
                                                                    1){
                        remark("Error en escritura de fichero");
                        status = FALSO;
                        fclose(in);
                    }
                    ap = ap->downptr;
                }
            c += 1;
            }
        fclose(in);
        eraseline(21)
        }
        eraseline(inpline);
}
```

La rutina **recuperar** es paralela a **archivar**; se le pide el nombre del fichero de entrada; y luego la función se encarga de leer registro a registro, escribiendo cada uno de estos en el lugar exacto tal y como fueron escritos.

recuperar tiene la caracterìstica que si en la posición de algún registro de celda leído de disco ya se encuentra información en la hoja previa, ésta se borra escribiendose la información que se esta leyendo del fichero de disco.

```
/* Recupera la informacion de la hoja del disco */
recuperar()
{
        char name[8];
        struct coldptr *cp1;

        postr("Recupero la hoja :",0, inpline);
        gets(name);
        if(strlen(name) > 0){
            center(name,21);
```

```
        puts("Recuperando");
        if((in = fopen(name, "rb")) == NULL){
                remark("Nopudo abrir el archivo");
                eraseline(inpline);
                return;
        }
        while(fread(&xp, sizeof(struct mueve), 1, in) == 1){
                if(feof(in)){
                        remark("Error de lectura");
                        close(in);
                        return;
                }
                else{
                        cpi = buscelda(xp.columna, xp.renglon);
                        if(cpi != NULL)
                                erasecelda(cpi);
                        cpi = (struct celdptr *) malloc (sizeof(struct
                                                        celdptr));
                        if(storecelda(xp.columna, xp.renglon, cpi))
                                if(strlen(xp.etiqueta) > 0){
                                  cpi->display = (char *) malloc (strlen
                                                        xp.etiqueta));
                                  if(storestr(xp.etiqueta,cpi->display))
                                      memout();
                                }
                                else{
                                        cpi->fp = (struct formptr *) malloc
                                                (sizeof(struct formptr *));
                                        if(storeform(cpi->fp)){
                                          cpi->fp->celdval = xp.valor1;
                                          cpi->fp->formula =(char *) malloc
                                                        (strlen(xp.formula));
                                          if(storestr(xp.formula,cpi->fp->
                                                        formula))
                                            memout();
                                        }
                                }
                        else
                                memout();
                }
        }
        fclose(in);
        labelcols(firstcol, firstrow);
        eraseline(23);
    }
    eraseline(22);
}
```

Tal vez desee experimentar con diferentes métodos de almacenamiento de los datos de una hoja. Puede ser una mejora para disminuir el tiempo necesario para leer y escribir la hoja a y desde disco, como tambien lo sería para disminuir la cantidad de espacio que los datos utilizan en el fichero de disco.

IMPRESIÓN DE LA HOJA

La última orden que hemos de considerar es la orden de impresión (</X>), la cual imprime una sección de la hoja especificada por el usuario en hoja de papel. Después de que se dé la orden, **MicroC** pregunta al usuario:

ColumnaInicial__ ColumnaFinal__ RenglonInicial__ RenglonFinal__

El usuario debe teclear un número entero menor o igual a las dimensiones de su hoja y presionar ENTER en cada posición introducida. A continuación mostramos el código de la función **imprimir** a la cual aún se le deben añadir importantes características.

```
/* Imprime la hoja en papel */
imprimir
{
      struct celdptr *cp1;
      int mreng, mcol, co_i, re_i, i, j, k;
      char x[20], x1[20];

      eraseline(22);
      gotoxy(0,22);printf("ColumnaInicial");scanf("%d",&co_i);
      gotoxy(15,22);printf("ColumnaFinal");scanf("%d",&mcol);
      gotoxy(30,22);printf("RenglonInicial");scanf("%d",&re_i);
      gotoxy(45,22);printf("RenglonFinal");scanf("%d",&reng);
      if((mcol - co_i) > 10)
            mcol = co_i + 10;
      printa("\n\r");
      i = re_i;
      while(i <= mreng){
            strcpy(x, "");
            if(renptr(i) == NULL)
                  printa("\n\r");
```

```
        else{
                j = co_i;
                while(j <= mcol){
                        cp = buscold(j, i);
                        if(cp === NULL)
                                strcat(x,"          ");
                        else
                                if(cp->display != NULL)
                                        strcat(x, cp->display);
                                        if(strlen(cp->display < 10)
                                                for(k=strlen(cp->display);
                                                                k<= 9; k++)
                                                        strcat(x," ");
                                }
                                else{
                                        gcvt(cp->(p->coldval, 15,x1);
                                        if(strlen(x1) < 10)
                                                for(k=strlen(x1); k <= 9;
                                                                k++)
                                                        strcat(x1, " ");
                                        strcat(x, x1);
                                }
                        j += 1;
                        }
                strcat(x, "\n\r");
                prints(x);
                }
        i += 1;
        }
        eraseline(22);
}
```

3.7 LA FUNCIÓN PRINCIPAL

Ahora que hemos definido los tipos de datos utilizados y las funciones de **MicroC**, podemos presentar la rutina principal y la declaración de las variables globales:

```
#include "stdio.h"   /* Libreria de funciones de TURBO_C */
#include "alloc.h"   /* Libreria de funciones de TURBO_C */
#include "math.h"    /* Libreria de funciones de TURBO_C */
#include "ctype.h"   /* Libreria de funciones de TURBO_C */
#include "bi_funs.h" /*Libreria de funciones de MicroC  */

/* Constantes de la hoja */

#define MAXRENS 255     /* NUmero maximo de renglones de la hoja */
#define MAXCOLS 63      /* Numero maximo de columnas de la hoja */
#define XORG 4          /* (XORG, YORG) posicion de la primera  */
#define YORG 2          /* celda  de la hoja */
#define IDSIZE 8        /* Tamano maximo de un identificador */
#define MAXVIDTH 76     /* Peso maximo de formato    */
#define INPCOL 8        /* Columna de entrada de informacion */
#define VIDTH 10        /* Ancho de la columna de cada celda */
#define MAXSTR 80       /* Longitud maxima de una cadena */
#define MAXCRTCOL 80    /* Numero de columnas de la pantalla */
#define MAXCRTREN 25    /* Numero de renglones de la pantalla */
#define RESIZE 15     /* Tamano maximo de un numero en la formula */

/* Variables booleanas */

#define TRUE  1      /* TRUE es un valor verdadero */
#define FALSO 0      /* FALSO es un valor falso */

/* Tipos de datos para la informacion de la formula */

struct formptr{
            double celdval; /* Contiene el resultado de la formula*/
            char *formula;  /* Contiene la expresion a evaluar */
            };
enum xresult{ OK, ZERODIVIDE, INVALID}; /*Posibles resultados de
                                                    una operacion */

/* Estructura para la informacion de cada celda */
```

```
struct celdptr{
            int celdcol;          /* Columna de la celda */
            int celdren;          /* Renglon de la celda */
            struct formptr *fp; /* Apuntador a la formula */
            char *display; /*Apuntador a una cadena de caracteres */
            strcut celdptr *rightptr, *downptr; /* Apuntadores para
                                                   enlazar la celda */
            };


/* Variables de la hoja */

int curscol, cursrov, firstrov, lastrov, firstcol, lastcol;
int nrows, impline, statline;
struct celdptr *curcp;
struct celdptr *colptr[80];
struct celdptr *renptr[255];
struct celdptr *cpa;
char dashes[80];
int colpos[65];
double zero;
int n_arg;

/* Variables para el flujo de control */

char alldone;
char ch, tup, tec_esp;

/* Cadenas para la captacion de identificadores */

char s[80];
char arg[80];

/* Tipos para el manejo de la pantalla */

typedef enum{HOME, CLEAR, ERASEOL, ERASEOS, UP, DOWN, LEFT,
                                    RIGHT, BEEP} crtcommand;

/* Tipos para el manejo de interrupciones */

stuct WORDREGS{
            unsigned int ax, bx, cx, dx, si, di,cflag, flags;
      };
struct BYTEREGS{
            unsigned char al, ah, bl,bh, cl, ch,  dl, dh;
      };
```

```
union REGS{
        struct WORDREGS x;
        struct BYTEREGS h;
    };

/* Tipos de datos para archivar y recuperar la hoja de trabajo
                                                        en disco */
struct mueve{
        int columna, renglon;
        char etiqueta[12], formula[25];
        double valorf;
        };
struct mueve xp; /*Estructura buffer para almacenamiento y carga*/
FILE  *in;        /* Apuntador al fichero de datos */
struct celdptr *ap; /* Apuntador a celda auxiliar */



/* Rutina principal */
main()
{
    entrada();
    inicvar();
    dibujahoja();
    setcolpos();
    labelcols(1);
    labelrens(1);
    desphoja(1,1);
    lineacc();
    setcursor(1,1);
    do{
        ch = get_key();
        if(tec_esp && (ch == 72 || ch == 80 || ch == 77 ||
                                                ch == 75))
            movecursor(ch);
        else if(tec_esp && ch == 71)
            setcursor(1,1);
        else if(isalpha(ch) || (tec_esp && ch == 59))
            getlabel(ch);
        else if(isdigit(ch) || ch == '+' || ch == '-' || ch ==
            '.' || ch == '(' || ch == '!' || (tec_esp && ch ==
                                                        60))
            getformula(ch);
        else if(ch == '/' || ch == '!' || ch == ':')
            docomando(ch);
    }while(alldone);
    crt(CLEAR);
}
```

Puesto que **MicroC** es un programa relativamente largo, tiene sentido dividir las declaraciones de constantes, tipos, variables en familias y comentar breves descripciones de las principales variables globales.

Nombre	Tipo	Descripción
curcol currow	entero	Coordenadas del rengón y columna de la posición actual del cursor.
firstrow lastrow	entero	Primero y último renglón presentado actualmente en la pantalla.
firstcol lastcol	entero	Primera y última columna presentada actualmente en la pantalla.
nrows	entero	Numero de renglones de la hoja presentada en pantalla.
inpline	entero	Linea de la pantalla sobre la que se indica la entrada del usuario.
statline	entero	Linea de la pantalla sobre la cual se presenta el estado programa.
curcp	celdptr	Puntero a la celda en la que actualmente se encuentra el cursor.
colptr renptr	arreglos tipo celdptr	Cabeceras para las columnas y renglones de la hoja.

Tras inicializar las variables globales, crear y presentar la hoja inicial, ect., **MicroC** introduce un ciclo que acepta un caracter de entrada escrito por el usuario y ejecuta una acción basada en el tipo de caracter introducido. Las cuatro acciones principales son : la rutina **movecursor** mueve el cursor a una celda adyacente de la hoja, **getlabel** acepta un rótulo del usuario, **getformula** acepta una fórmula y **docomando** desarrolla una de las órdenes de la tabla 3.1.1. El ciclo continua hasta que el discriminador **alldone** se vuelve **TRUE** esto se realiza mediante la orden de salida.

INICIALIZACIÓN DE VARIABLES GLOBALES

MicroC usa una función separada para inicializar las variables cómunes a todo el prigrama. He aquí esta rutina:

```
/* Inicializa las variables globales de la hoja */
inicvar()
{
      register i;

      crt(CLEAR);
      inpline = MAXCRTREN - 2;
      statline = MAXCRTREN -3;
      alldone = TRUE;
      nrow = MAXCRTREN - Yorg - 3;
      for(i =0; i <= MAXCOLS; i++)
            colptr[i] = NULL;
      for(i = 0; i <= MAXRENS; i++)
            renptr[i] = NULL;
      for(i =0; i <= 79; i++)
            dashes[i] = '-';
      zero = 0;
}
```

4- APLICACIONES DE MicroC

4.1 Tres aplicaciones con MicroC

En este capítulo se muestran con un listado, el uso de las principales órdenes y fórmulas que se necesitan introducir para producir un modelo de aplicación.

Una ilustración completa, incluyendo datos de muestra, se presenta en cada modelo.

Nuestra sugerencia es que después de introducir el modelo, utilice los datos de muestra para probar la precisión de lo que ha introducido. Si los resultados obtenidos son diferentes a los mostrados aquí, hay varias cosas que puede hacer para determinar cual es al problema.

Lo primero a hacer es imprimir el listado de su modelo y compararlo con el listado presentado aquí. Recordar que cada caracter y espacio son significativo, excepto los dos puntos (:) que se imprime en el listado par mayor claridad. Una inspección a fondo seguramente revelará algún error de escritura.

Si su problema aún permanece después de eliminar los errores de entrada de datos, se sugiere que revice el diseño el tema de la implementación de la hoja electrónica **MicroC** para detectar algún posible mal uso del programa.

BALANCE, ESTADO DE LOS INGRESOS Y ANALISIS FINANCIERO

Este modelo produce una hoja con el balance y un estado de las rentas y calcula algunas relaciones financieras comunes. Para ver el estado de las rentas y las relaciones financieras después de que se hayan introducido las fórmulas calcule la hoja completa.

El estado de los ingresos comienza en la línea 22, la razón de rentabilidad en la línea 42 y la rzón de liquidez y solvencia en la línea 62. La Figura 1-1 muestra el modelo completo con un ejemplo.

El balance y el estado de los ingresos deben terminarse antes de que se calculen financieras. Después de dar los datos solicitados para el balance y el estado de los ingresos, vuelva a calcular la hoja para ver los resultados.

Los títulos de los elementos del balance y del estado de los ingresos pueden cambiarse, o añadirse otros, con tal de que las fórmulas ajusten lo necesario.

LISTADO

```
[4,1]:  '  HOJA DE BALANCE
[1,1]:  '-------------------------------------------------------
        -----------------
[2,2]:  '---------
[3,2]:  '---------
[4,2]:  '---------
[5,2]:  '---------
[6,2]:  '---------
[7,2]:  '---------
[8,2]:  '---------
[1,3]:  'NOMBRE COMPAÑIA:
[3,3]:  'TECLEAR NOMBRE
[5,3]:  'DE:
[6,3]:  'TECLEAR FECHA
[1,5]:  'VENTAJAS
[5,5]:  'PASIVO
[1,6]:  '  DINERO DISPONIBLE.............
[3,6]:  'TECLEAR NUMERO
[5,6]:  '    ESTADO DE CUENTA PBLE...
[7,6]:  'TECLEAR NUMERO
[1,7]:  '    ESTADO DE CUENTA REC......
[3,7]:  'TECLEAR NUMERO
[5,7]:  '  NOTAS PBLE........
[7,7]:  'TECLEAR NUMERO
[1,8]:  '  NOTAS REC.........
[3,8]:  'TECLEAR NUMERO
[5,8]:  '   INTERESES PBLE....
[7,8]:  'TECLEAR NUMERO
[1,9]:  '  INVENTARIO........
[3,9]:  'TECLEAR NUMERO
[5,9]:  '   IMPUESTOS PBLE.......
[7,9]:  'TECLEAR NUMERO
[1,10]: ' OTROS GASTOS....
[3,10]: 'TECLEAR NUMERO
[5,10]: ' OTROS CORRIENTES PASIVOS.....
[7,10]: 'TECLEAR NUMERO
[1,11]: '     PEQUEÑAS VENTAS..........
```

```
[4,11]: SUM([3,6]:[3,10])
[5,11]: '     TOTAL DISPONIBLE.....
[8,11]: SUM([7,6]:[7,10])
[1,13]: '  LAND.............
[3,13]: 'TECLEAR NUMERO
[5,13]: '  FIAR PBLE.......
[7,13]: 'TECLEAR NUMERO
[1,14]: ' CONSTRUCCION
[3,14]: 'TECLEAR NUMERO
[5,14]: '  PRESTAMO PBLE........
[7,14]: 'TECLEAR NUMERO
[1,15]: '  EQUIPAMIENTO........
[3,15]: 'TECLEAR NUMERO
[5,15]: '  HIPOTECA PBLE....
[7,15]: 'TECLEAR NUMERO
[1,16]: ' MENOS LO ACUMULADO DEP....
[7,16]: 'TECLEAR NUMERO
[1,17]: '  OTRAS VENTAJAS
[3,17]: 'TECLEAR NUMERO
[1,18]: '     TOTALES.....
[4,18]: [3,13]+[3,14]+[3,15]-[3,16]+[3,17]
[5,18]: '     TOTALES.............
[8,18]: SUM([7,13]:[7,16])
[5,19]: '     PROPIETARIOS EQUITATIVOS.....
[8,19]: 'TECLEAR NUMERO
[1,20]: '     TOTALES.................
[4,20]: [4,20]+[4,28]
[5,20]: '     TOTAL PASIVO .......
[8,20]: [8,11]+[8,18]+[8,19]
[3,22]: ' RETENSIONES DE AUDITORES.............
[1,23]: '-------------------------------------------------------------
 ------------------
[2,23]: '----------
[3,23]: '----------
[4,23]: '----------
[5,23]: '----------
[6,23]: '----------
[7,23]: '----------
[8,23]: '----------
[1,24]: 'NOMBRE DE LA COMPANIA:
[3,24]: 'TECLEAR NOMBRE:
[5,24]: 'PARA PERIODO FINAL:
[7,24]: 'TECLEAR FECHA:
[1,26]: 'VENTAS Y RENTAS..............
[4,26]: '  TECLEAR NUMERO
[6,26]: '  IMPUESTOS RETENIDOS (.XX)=
[8,26]: '  TECLEAR NUMERO
[1,27]: 'OTRAS RENTAS..............
[4,27]: '  TECLEAR NUMERO
[2,28]: '  TOTAL RENTAS.............
```

```
[5,28]: [4,26]+[4,27]
[1,30]: 'MENOS GASTOS:
[1,31]: ' COSTOS Y ESTADOS DE CUENTA..
[4,31]: '  TECLEAR NUMERO
[1,32]: ' VENTAS.....
[4,32]: '  TECLEAR NUMERO
[1,33]: ' ADMINISTRATIVO.....
[4,33]: '  TECLEAR NUMERO
[1,34]: ' INTERESES.....
[4,34]: '  TECLEAR NUMERO
[1,35]: ' IMPUESTOS SOBRE VENTAS.....
[4,35]: [5,28]-SUM[4,31][4,31..[4,34]]*[8,26]
[1,36]: '            TOTAL GASTOS.....
[5,36]: SUM([4,31]..[4,35])
[2,37]: '  INGRESO NETO.....
[5,37]: +[5,28]-[5,36]
[1,38]: 'MENOR:
[1,39]: ' DIVIDENDOS.....
[5,39]: '  TECLEAR NUMERO
[2,40]: '  RETENCIONES.....
[5,40]: +[5,37]-[5,39]
[1,42]: 'COCIENTE DE BENEFICIOS
[1,43]: '----------
[1,45]: 'RANGO DE RETORNO
[6,45]: '    =
[7,45]: [16,2] [5,37]+[4,34]*[1-[8,26]]]/[4,20]
[1,46]: '  NETO INC+INTERES EXP(NETO DE IMPUESTO)/PROMEDIO
[1,48]: 'MARGEN DE RADIO
[6,48]: '    =
[6,48]: [16,2] [5,37]+[4,34]*[1-[8,26]]]/[4,26]
[1,49]: '  NETO INC+INT EXP(IMPUESTO )/VENTAS
[1,51]: 'TOTALES
[6,51]: '    =
[7,51]: [16,2]+[4,26]/[4,20]
[1,52]: '  VENTAS/PORCENTAJE TOTAL
[1,54]: 'INVENTARIO
[6,54]: '    =
[7,54]: [16,2]+[4,31]/[3,9]
[1,55]: '  COSTOS/PROMEDIO INVENTARIO
[1,57]: 'PLANTEAMIENTO
[6,57]: '    =
[7,57]: [16,2]+[4,26]/@SUM[3,10]..[3,16]
[1,58]: '  VENTAS/PROMEDIO PLANTEAMIENTO
[1,62]: 'COCIENTE DE LIQUIDACION
[1,63]: '----------
[1,65]: 'COCIENTE ACTUAL
[6,65]: '    =
[7,65]: [16,2] +[4,11]/[8,11]
[1,66]: '   CORRIENTE ACTIVO/CORRIENTE PASIVO
[1,68]: 'RADIO
```

```
[6,68]: '    =
[7,68]: [16,2] [3,6]+[7]+[3,8]/[8,11]
[1,69]: '   LIQUIDACION ACTIVO/CORRIENTE PASIVO
[1,71]: 'COCIENTES SOLVENTES
[1,72]: '---------------
[1,74]: 'LONG-TERMINACION
[6,74]: '    =
[7,74]: [16,2] +[8,18]/[8,18]+[8,19]
[1,75]: '   TOTALES LIAB/TOT NO-LIAB
[1,77]: 'TIEMPO DE CARGA
[6,77]: '    =
[7,77]: [16,2] [5,37]+[4,34]+[4,35]/[4,34]
[1,78]: '   NETO(ANTES INTERES&IMPUESTO)/INTERES
[3,1]:  ' ASIGNACION DE GASTOS A PRODUCTOS
```

ASIGNACION DE GASTOS A PRODUCTOS

Este modelo asigna gastos a una colección de productos como un porcentaje de las ventas. El modelo completo, con un ejemplo introducido, se muestra en la figura 4.2. Simplemente introducir el costo de los gastos por categoría, insertando tantas filas como sean necesarias. Hay que tener cuidado de ajustar la fila **VENTAS TOTALES** al final y copiar la fórmula TOTAL en la última columna. Introducir los datos de ventas para cada producto. Aquí también puede insertar más filas para los productos adicionales, pero asegurándose de copiar todas las fórmulas apropiadamente y de ajustar las fórmulas las fórmulas en la fila **VENTAS TOTALES** . Puede usar un parámetro distinto a las ventas para asignar gastos, simplemente insertando estas filas en l espacio suministrado para los datos de las ventas y cambiando el título de la sección. Después de que haya introducido todos los datos, recalcule la hoja para ver los resultados.

LISTADO

```
[1,2]:  '-------------------------------------------------------------
         ---------------
[4,2]:  '----------------------
[1,3]:  'GASTOS GENERALES
[3,3]:  'ENE
[4,3]:  'FEB
[5,3]:  'MAR
[6,3]:  'ABR
[7,3]:  'MAY
[8,3]:  'JUN
[9,3]:  'JUL
[10,3]: 'AGO
[11,3]: 'SEP
[12,3]: 'OCT
[13,3]: 'NOV
[14,3]: 'DIC
[15,3]: 'TOTAL
[1,4]:  '--------------------------
[3,4]:  '-------
[4,4]:  '-------
[5,4]:  '-------
[6,4]:  '-------
[7,4]:  '-------
[8,4]:  '-------
[9,4]:  '-------
[10,4]: '-------
[15,4]: '-------
[1,5]:  'RENTA
[3,5]:  'TECLEAR NUMERO
[4,5]:  'TECLEAR NUMERO
[5,5]:  'TECLEAR NUMERO
[6.5]:  'TECLEAR NUMERO
[7,5]:  'TECLEAR NUMERO
[8,5]:  'TECLEAR NUMERO
[9,5]:  'TECLEAR NUMERO
[10,5]: 'TECLEAR NUMERO
[11,5]: 'TECLEAR NUMERO
[12,5]: 'TECLEAR NUMERO
[13,5]: 'TECLEAR NUMERO
[14,5]: 'TECLEAR NUMERO
[15,5]:  SUM([3,5]:[14,5])
[1,6]:  ' UTILIDAD
[3,6]:  'TECLEAR NUMERO
[4,6]:  'TECLEAR NUMERO
[5,6]:  'TECLEAR NUMERO
[6,6]:  'TECLEAR NUMERO
[7,6]:  'TECLEAR NUMERO
```

```
[8,6]:  'TECLEAR NUMERO
[9,6]:  'TECLEAR NUMERO
[10,6]: 'TECLEAR NUMERO
[11,6]: 'TECLEAR NUMERO
[12,6]: 'TECLEAR NUMERO
[13,6]: 'TECLEAR NUMERO
[14,6]: 'TECLEAR NUMERO
[15,6]: SUM([3,5]:[14,5])
[1,6]:  'UTILIDAD
[3,6]:  'TECLEAR NUMERO
[4,6]:  'TECLEAR NUMERO
[5,6]:  'TECLEAR NUMERO
[6,6]:  'TECLEAR NUMERO
[7,6]:  'TECLEAR NUMERO
[8,6]:  'TECLEAR NUMERO
[9,6]:  'TECLEAR NUMERO
[10,6]: 'TECLEAR NUMERO
[11,6]: 'TECLEAR NUMERO
[12,6]: 'TECLEAR NUMERO
[13,6]: 'TECLEAR NUMERO
[14,6]: 'TECLEAR NUMERO
[15,6]: SUM([3,6]:[14,6])
[1,7]:  'OTROS
[3,7]:  'TECLEAR NUMERO
[4,7]:  'TECLEAR NUMERO
[5,7]:  'TECLEAR NUMERO
[6,7]:  'TECLEAR NUMERO
[7,7]:  'TECLEAR NUMERO
[8,7]:  'TECLEAR NUMERO
[9,7]:  'TECLEAR NUMERO
[10,7]: 'TECLEAR NUMERO
[11,7]: 'TECLEAR NUMERO
[12,7]: 'TECLEAR NUMERO
[13,7]: 'TECLEAR NUMERO
[14,7]: 'TECLEAR NUMERO
[15,7]: SUM([3,7]:[14,7])
[1,8]:  '----------------------
[3,8]:  '-------
[4,8]:  '-------
[5,8]:  '-------
[6,8]:  '-------
[7,8]:  '-------
[8,8]:  '-------
[9,8]:  '-------
[10,8]: '-------
[11,8]: '-------
[12,8]: '-------
[13,8]: '-------
[14,8]: '-------
[15,8]: '-------
```

```
[1,9]:  'TOTAL GASTOS
[3,9]:  SUM([3,5]..[3,7])
[4,9]:  SUM([4,5]..[4,7])
[5,9]:  SUM([5,5]..[5,7])
[6,9]:  SUM([6,5]..[6,7])
[7,9]:  SUM([7,5]..[7,7])
[8,9]:  SUM([8,5]..[8,7])
[9,9]:  SUM([9,5]..[9,7])
[10,9]: SUM([10,5]..[10,7])
[11,9]: SUM([11,5]..[11,7])
[12,9]: SUM([12,5]..[12,7])
[13,9]: SUM([13,5]..[13,7])
[14,9]: SUM([14,5]..[14,7])
[15,9]: SUM([15,9]..[15,9])
[1,10]: '-------------------------------------------------------
         --------------------
[1,11]: 'DATOS DE PRODUCTOS DE VENTAS
[3,11]: 'ENE
[4,11]: 'FEB
[5,11]: 'MAR
[6,11]: 'ABR
[7,11]: 'MAY
[8,11]: 'JUN
[9,11]: 'JUL
[10,11]: 'AGO
[11,11]: 'SEP
[12,11]: 'OCT
[13,11]: 'NOV
[14,11]: 'DIC
[15,11]: 'TOTAL
[1,11]:  '-------------------
[3,12]:  '-------
[4,12]:  '-------
[5,12]:  '-------
[6,12]:  '-------
[7,12]:  '-------
[8,12]:  '-------
[9,12]:  '-------
[10,12]: '-------
[11,12]: '-------
[12,12]: '-------
[13,12]: '-------
[14,12]: '-------
[15,12]: '-------
[1,13]:  'TECLEAR NOMBRE DEL PRODUCTO
[3,13]:  'TECLEAR NUMERO
[4,13]:  'TECLEAR NUMERO
[5,13]:  'TECLEAR NUMERO
[6,13]:  'TECLEAR NUMERO
[7,13]:  'TECLEAR NUMERO
```

```
[8,13]:  'TECLEAR NUMERO
[9,13]:  'TECLEAR NUMERO
[10,13]: 'TECLEAR NUMERO
[11,13]: 'TECLEAR NUMERO
[12,13]: 'TECLEAR NUMERO
[13,13]: 'TECLEAR NUMERO
[14,13]: 'TECLEAR NUMERO
[15,13]: SUM([3,13]..[14,13])
[2,14]:  'GASTOS
[3,14]:  ([3,13]/[3,20])*[3,9]
[4,14]:  ([4,13]/[4,20])*[4,9]
[5,14]:  ([5,3]/[5,20])*[5,9]
[6,14]:  ([6,13]/[6,20]*[6,9]
[7,14]:  ([7,13]/[7,20]*[7,9]
[8,14]:  ([8,13]/[8,20]*[8,9]
[9,14]:  ([9,13]/9,20]*[9,9]
[10,14]: ([10,13]/[10,20]*[10,9]
[11,14]: ([11,13]/[11,20]*[11,20]
[12,14]: ([12,13]/[12,20]*[12,9]
[13,14]: ([13,13]/[13,20]*[13,9]
[14,14]: ([14,13]/[14,20]*[14,9]
[15,14]: ([15,13]/[15,20]*[15,9]
[1,15]: 'TECLEAR NOMBRE DEL PRODUCTO
[3,15]: 'TECLEAR NUMERO
[4,15]: 'TECLEAR NUMERO
[5,15]: 'TECLEAR NUMERO
[6,15]: 'TECLEAR NUMERO
[7,15]: 'TECLEAR NUMERO
[8,15]: 'TECLEAR NUMERO
[9,15]: 'TECLEAR NUMERO
[10,15]: 'TECLEAR NUMERO
[11,15]: 'TECLEAR NUMERO
[12,15]: 'TECLEAR NUMERO
[13,15]: 'TECLEAR NUMERO
[14,15]: 'TECLEAR NUMERO
[15,15]: SUM([3,15]..[14,15])
[2,16]:  'GASTOS
[3,16]:  ([3,15]/[3,20])*[3,9]
[4,16]:  ([4,15]/[4,20])*[4,9]
[5,16]:  ([5,15]/[5,20])*[5,9]
[6,16]:  ([6,15]/[6,20])*[6,9]
[7,16]:  ([7,15]/[7,20])*[7,9]
[8,16]:  ([8,15]/[8,20])*[8,9]
[9,16]:  ([9,15]/9,20])*[9,9]
[10,16]: ([10,15]/[10,20])*[10,9]
[11,16]: ([11,15]/[11,20])*[11,9]
[12,16]: ([12,15]/[12,20])*12,9]
[13,16]: ([13,15]/[13,20])*[13,9]
[14,16]: ([14,15]/[13,20])*[13,9]
[15,16]: SUM([3,15]..[3,15])
```

```
[2,16]:  'GASTOS
[3,16]:  ([3,15]/[3,20])*[3,9]
[4,16]:  ([4,15]/[4,20])*[4,9]
[5,16]:  ([5,15]/[5,20])*[5,9]
[6,16]:  ([6,15]/[6,20])*[6,9]
[7,16]:  ([7,15]/[7,20])*[7,9]
[8,16]:  ([8,15]/[8,20])*[8,9]
[9,16]:  ([9,15]/[9,20])*[9,9]
[10,16]: ([10,15]/[10,20])*[10,9]
[11,16]: ([11,15]/[11,20])*[11,9]
[12,16]: ([12,15]/[12,20])*[12,9]
[13,16]: ([13,15]/[13,20])*[13,9]
[14,16]: ([14,15]/[14,20])*14,9]
[15,16]: ([15,15]/[15,20])*[15,9]
[1,17]:  'TECLEAR NOMBRE DEL PRODUCTO
[3,17]:  'TECLEAR NUMERO
[4,17]:  'TECLEAR NUMERO
[5,17]:  'TECLEAR NUMERO
[6,17]:  'TECLEAR NUMERO
[7,17]:  'TECLEAR NUMERO
[8,17]:  'TECLEAR NUMERO
[9,17]:  'TECLEAR NUMERO
[10,17]: 'TECLEAR NUMERO
[11,17]: 'TECLEAR NUMERO
[12,17]: 'TECLEAR NUMERO
[13,17]: 'TECLEAR NUMERO
[14,17]: 'TECLEAR NUMERO
[15,17]: SUM([14,17]..[14,17])
[2,18]:  'GASTOS
[3,18]:  ([3,17]/[3,20])*[3,9]
[4,18]:  ([4,17]/[4,20])*[4,9]
[5,18]:  ([5,17]/[5,20])*[5,9]
[6,18]:  (6,17]/[6,20])*[6,9]
[7,18]:  ([7,17]/[7,20])*[7,9]
[8,18]:  ([8,17]/[8,20])*[8,9]
[9,18]:  ([9,17]/[9,20])*[9,9]
[10,18]: ([10,17]/[10,20])*[10,9]
[11,18]: ([11,17]/[11,20])*[11,9]
[12,18]: ([12,17]/[12,20])*[12,9]
[13,18]: ([13,17]/[13,20])*[13,9]
[14,18]: ([14,17]/[14,20])*[14,9]
[15,18]: ([15,17]/[15,20])*[15,9]
[1,19]:  '--------------------------------
[3,19]:  '--------
[4,19]:  '--------
[5,19]:  '--------
[6,19]:  '--------
[7,19]:  '--------
[8,19]:  '--------
[9,19]:  '--------
```

```
[10,19]: '--------
[11,19]: '--------
[12,19]: '--------
[13,19]: '--------
[14,19]: '--------
[15,19]: '--------
[1,20]:  'TOTAL VENTAS
[3,20]:  [3,13]+[3,15]+[3,17]
[4,20]:  [4,13]+[4,15]+[4,17]
[5,20]:  [5,13]+[5,15]+[5,17]
[6,20]:  [6,13]+[6,15]+[6,17]
[7,20]:  [7,13]+[7,15]+[6,17]
[8,20]:  [8,13]+[8,15]+[8,17]
[9,20]:  [9,13]+[9,15]+[9,17]
[10,20]: [10,13]+[10,15]+[10,17]
[11,20]: [11,13]+[11,15]+[11,17]
[12,20]: [12,13]+[12,15]+[12,17]
[13,20]: [13,13]+[13,15]+[13,17]
[14,20]: [14,13]+[14,15]+[14,17]
[15,20]: [15,13]+[15,15]+[15,17]
```

ANALISIS COSTO/BENEFICIO DE LA PUBLICIDAD Y PROMOCION

Este modelo calcula los costos y beneficios asociados con la publicidad en diferentes publicaciones. El modelo completo, con un ejemplo introducido, se muestra en la figura 4.3. Se suministran el costo por inserción y números de inserciones por cada publicación, así como la cantidad y valor en pesos de las ventas generales por la publicidad de cada publicación. El modelo calcula los costos totales de la publicidad en cada publicación y también los costos relativos a la circulación, volumen de ventas y ventas en pesos. Depués de introducir todos los datos recalcule la hoja para ver los resultados.

LISTADO

```
[1,4]: 'CIRCULACION:
[4,4]: 'TECLEAR NUMERO
[5,4]: 'TECLEAR NUMERO
[6,4]: 'TECLEAR NUMERO
[7,4]: 'TECLEAR NUMERO
[8,4]: SUM([4,4]..[7,4])
[1,6]: 'COSTO POR PUBLICACION
[1,7]: 'COSTO/INCERSION
[4,7]: ([6,2]) 0 'TECLEAR NUMERO
```

```
[5,7]: ([6,2]) 0 'TECLEAR NUMERO
[6,7]: ([6,2]) 0 'TECLEAR NUMERO
[7,7]: ([6,2]) 0 'TECLEAR NUMERO
[8,7]: ([6,2]) 0 'TECLEAR NUMERO
[1,8]: 'NUMERO DE INCERSIONES
[4,8]: 'TECLEAR NUMERO
[5,8]: 'TECLEAR NUMERO
[6,8]: 'TECLEAR NUMERO
[7,8]: 'TECLEAR NUMERO
[8,8]: 'TECLEAR NUMERO
[9,8]:  SUM([4,8]..[7,8])
[4,9]: '--------
[5,9]: '--------
[6,9]: '--------
[7,9]: '--------
[8,9]: '--------
[1,10]: 'TOTAL DE COSTOS DE PUBLICIDAD
[4,10]: ([6,2]) ([4,7]*[4,8])
[5,10]: ([6,2]) ([5,7]*[5,8])
[6,10]: ([6,2]) ([6,7]*[6,8])
[7,10]: ([6,2]) ([7,7]*[7,8])
[8,10]: ([6,2]) ([8,7]*[8,8])
[8,13]: SUM([4,13]..[7,13])
[1,14]: '  $ FECHA A VENTAS
[4,14]: 'TECLEAR NUMERO
[5,14]: 'TECLEAR NUMERO
[6,14]: 'TECLEAR NUMERO
[7,14]: 'TECLEAR NUMERO
[8,14]:  SUM([4,14]..[7,14])
[1,15]: '-------------------------------------------------------
         ---------------
[1,16]: 'COSTO POR CIRCULACION
[4,16]:  ([4,10]/[4,4])
[5,16]:  ([5,10]/[5,10])
[6,16]:  ([6,10]/[6,10])
[7,16]:  ([7,10]/[7,10])
[8,16]:  ([8,10]/[8,10])
[1,18]:  '  COSTO POR NUMERO DE VENTAS
[4,18]:  ([4,10]/[4,13])
[5,18]:  ([5,10]/[5,13])
[6,18]:  ([6,10]/[6,13])
[7,18]:  ([7,10]/[7,13])
[8,18]:  ([8,10]/[8,13])
[1,20]:  '  COSTO POR  $ DE VENTAS
[4,20]:  ([4,10]/[4,14])
[5,20]:  ([5,10]/[5,14])
[6,20]:  ([6,10]/[6,14])
[7,20]:  ([7,10]/[7,14])
[8,20]:  ([8,10]/[8,14])
```

5. CONCLUSIONES

5.1 Sugerencias para mejorar el programa

MicroC es un programa flexible, y cuanto más flexible es un programa más se puede mejorar.

El primer aspecto que se puede mejorar es el de la presentación de la hoja y el manejo de pantalla que se hace para llevar a cabo esta tarea. El diseño del manejo de pantalla de **MicroC** puede soportar funciones avanzadas tales como la de inserción y supresión de líneas, así como la utilización de manejo de ventanas que mejorarían bastante la velocidad a la que corre la hoja.

Esta técnica de manejo de ventanas puede ser utilizada para facilitar el manejo de la información escrita sobre la hoja; como la de visualizar dos o más secciones de la hoja al mismo tiempo; presentar información acerca del manejo de los principales comandos de **MicroC**; desplegar directorios de discos; entre otros.

En el apéndice **A** presentamos una rutina y una breve explicación de como crear y manejar información utilizando la técnica de ventanas.

Por otra parte si se introducen cambios en el cálculo de las fórmulas mejoraría la eficiencia de la hoja.

Otro aspecto que se presta a la investigación es la mejora del manejo de la memoria permitiendo a **MicroC** manejar hojas más grandes con la misma cantidad de memoria que antes.

He aquí una sugerencia concreta: escribir una versión de **MicroC** que pre-interprete fórmulas cuando se introducen por el usuario. Una buena parte del tiempo gastado por **MicroC** se va en el cálculo de las fórmulas introducidas. La pre-interpretación de las fórmulas haría que las conversiones de cadena a número y viceversa se realizaran una vez y posteriormente se almacenarán en las fórmulas como números reales y no cómo sus representaciones en cadena.

También se puede investigar la escritura de fórmulas en notación posfija, las cuales pueden evaluarse fácilmente con unas rutinas sencillas no recursivas.

Hay muchas características (funciones trigonométricas, logaritmos ,por ejemplo), podrían añadirse a **MicroC** para incrementar su flexibilidad. También podría añadirse un calendario aritmético.

Igualmente puede resultar útil desarrollar diferentes versiones de **MicroC** para diferentes aplicaciones. Para utilizarlo en trabajos estadísticos, incorporandole funciones que calculen la media, desviación estandar, métodos de mínimos cuadrados, etcétera, podría resultar muy útil.

Para un ingeniero sería interesante incorporar una rutina de inversión de matrices. Un administrador necesitaría, tal vez, funciones para evaluar series de tiempo, como medias de movimiento, regresión lineal y otros.

Podría resultar útil un formateado más flexible: permitir centrar dentro de una columna, insertar comas en los números, signos de pesos flotantes, signos de porcentaje, etc. Tal vez resulte útil añadir una orden de ordenación que automáticamente colocaría los renglones o las columnas en orden alfabético o numérico. Podrían añadirse otras características para aumentar la facilidad de uso de **MicroC.** Por ejemplo, la forma de alterar una fórmula o u rótulo. Una característica de edición más sofisticada, para introducir cadenas sería conveniente. Otra idea es permitir que las celdas sean nombradas usando identificadores o significados mnemotécnicos, por ejemplo, si la celda **(5,1)** contiene un tanto por ciento de interés, el usuario podría denominar a al celda **INTERES** y usar el identificador en fórmulas sucesivas, en lugar de las coordenadas de la celda.

Un proyecto muy interesante sería implementar una orden de anación, la cual desharía los efetos de la última orden introducida por usuario.

Esta sugerencias se quedan tan sólo en la superficie de las muchas mejoras que pueden realizarse.

5.2 Recomendaciones para hacer a prueba de errores sus valiosas Hojas de Cálculo

Ya sea que usted sea un principiante precavido o un seguro usuario avanzado, las posibilidades son varias de que sus modelos de hoja contengan errores ya que, según una estadística confirmada una de cada tres hojas de trabajo a nivel profesional contiene errores.

Las potenciales fuentes de error recorren la gama que va desde defectos estructurales hasta errores de mecanografía.

Para prevenirse de tales errores le podemos hacer las siguientes sugerencias:

1. PAra evitar un error de rango, usted deberá siempre incluir renglones extras en blanco en sus funciones que manejen rangos. (Por ejemplo, si usted esta haciendo una suma que va de del renglón 8 al 12 en la columna D, deberá dejar en blanco los renglones 7 y 13 en esa columna y escribir su fórmula de manera que las incluya @SUM(D7..D13), así, si más tarde insertará una línea en el renglón 7 ésta estará incluida en totales y así usted no tendrá que culpar a su hoja de cálculo.

2. Mantenga reducido el tamaño de su hoja de trabajo para hacer mucho más fácil la localizacón de errores, como también limitar sus efectos nocivos. Algunos expertos recomiendan nunca tener un modelo que ocupe más de 100K de memoria. Esto podría parecer extremadamente conservativo pero los expertos aconsejan asignar archivos a funciones financieras individuales, las cuales pueden ser consolidadas más tarde cuando se necesiten.

3. Sea consistente en la construcción y copia de fórmulas y en la señalización de rangos (Vaya siempre de izquierda a derecha y de arriba a abajo).

4. Ajuste el formato de las celdas y ancho de las columnas

para checar datos y cálculos.

5. Aprenda esta buena rutina de los contadores la cual dice: dos conjuntos de fórmulas realizando operaciones idénticas sobre datos en direcciones complementarias deben producir resultados idénticos.

6. En la construcción de un modelo, deberá siempre incluir etiquetas con anotaciones explicativas las cuales no solo le permitirán aclarar a usted lo que ha hecho sino también a cualquier otro quien pudiera utilizarlo. Una técnica es la de introducir explicaciones celda por celda de fórmulas y macros en una área separada de su hoja. Documentando una hoja fomentará la revisión continua de ésta.

7. Otro error aún más común es el no salvar su hoja de trabajo en disco. (Existe un programa en elmercado que evita esto, al salvar su hoja a determinados intervalos de tiempo).

8. Como una prueba adicional, usted puede graficar sus resultados para uan confirmación visual, si existiera una protuberancia o un hoyo en la curva de sus valores haría bien en checar su modelo.

9. Una última recomendación sería la asignacón de una sola persona para que sea la responsable del manejo de todo el archivo de hojas de trabajo de la compañia.

APENDICE A

A.1 Funciones de carácter general

A continuación describiremos las funciones de carcater general que fueron utilizadas a lo largo del programa y que por alguna razón no se incluyeron en diseño de **MicroC**, pero si formaron parte esencial para el complemento y mejor implementación de nuestra hoja electrónica.

MANEJO DE LA PANTALLA

MicroC es un programa orientado al manejo de pantalla, es decir, dan continuamente al usuario información sobre el estado del programa, mensajes de ayuda y otros datos. Toda esta información ayuda al usuario a utilizar más efectivamente el programa.

Primero diseñaremos algunas funciones para el manejo de pantalla más comunes:

```
/* Deslizamiento de lineas de lineas en blanco */
clela, b, c, d)
int a, b, c, d;
{
      union REGS r;
      r.h.ah = 6;
      r.h.al = 0;
      r.h.ch = a;
      r.h.cl = b;
      r.h.dh = c;
      r.h.dl = d;
      r.h.bh = 7;
      int86(0x10, &r, &r);
}



/* Envia el cursor a la posicion especificada */
gotoxy(x, y)
int x, y;
{
      union REGS r;

      r.h.ah = 2;
```

```
        r.h.dl = y;
        r.h.dh = x;
        r.h.bh = 0;
        int86(0x10,&r, &r);
}

.

/* Encuentra la posicion actual del cursor */
pos()
{
        union REGS r;
        r.h.ah = 3;
        r.h.bh = 0;
        int86(0x10, &r,&r);
        h = r.h.dh;
        k = r.h.dl;
}
```

Nuestro siguiente paso es escribir una función en lenguaje C llamada **crt** que ejecuta las órdenes que se muestran en la siguiente tabla:

Orden	Resultado
HOME	Envía el cursor a la esquina superior izquierda de la pantalla.
CLEAR	Borra la pantalla completamente.
ERASEOL	Borra la pantalla desde la posición actual del cursor hasta el final de la línea.
ERASEOS	Borra la pantalla desde la posoción actual del cursor hasta el final de de la pantalla.
UP	Sube el cursor una línea.
DOWN	Baja el cursor una línea.
LEFT	Mueve el cursor hacia la izquierda un espacio.
RIGHT	Mueve el cursor hacia la derecha un espacio.

BEEP Suena el altavoz del terminal (o de la computadora).

Esta es una versión de crt:

```
/* Ordenes de pantalla */
crt(c)
crtcommand c;
{
        switch(c)
        {
                case HOME:goto_xy(0, 0);
                                break;
                case CLEAR:cls(0, 0, 24, 79);
                                break;
                case ERASEOL:pos();
                                cls(h, k, h, 79);
                                break;
                case ERASEOS:pos();
                                cls(h, k, h, 79);
                                cls(h + 1, 0, 24, 79);
                                break;
                case UP:        pos();
                                  goto_xy(h - 1, k);
                                  break;
                case DOWN:      pos();
                                  goto_xy(h + 1, k);
                                  break;
                case LEFT:      pos();
                                  goto_xy(h, k - 1);
                                  break;
                case RIGHT:     pos();
                                  goto_xy(h, k + 1);
                                  break;
                case BEEP:      printf("%c", 7);
                                  break;
        }
}
```

RUTINAS QUE ENVIAN MENSAJES

Si no hay sitio en la memoria **MIcroC** utiliza la función **memout** para avisarle al usuario:

```
/* Envia un mensaje de desbordamiento de memoria */
memout()
{
        remark("Fuera de memoria");
}
```

Usamos **remark** siempre que la atención del usuario haya de ser atraída hacia un mensaje de aviso o unstrucción.

```
/* Pone un mensaje en la pantalla */
remark(rem)
char rem[80];
{
        crt(BEEP);
        center(rem, MAXCRTREN - 1);
        wait();
        eraseline(MAXCRTREN - 1);
}
```

remark usa **wait** la cual simplemente presenta el mensaje **Presione -> para continuar** y espera que el usuario pulse la tecla < **RETURN** >.

```
/* Espera que el usuario pulse cualquier tecla */
wait()
{
        char ch;

        center("Presione -> para continuar", MAXCRTREN);
        ch = get_key();
        while (ch != 13)
                ch = get_key();
        eraseline(MAXCRTREN);
}
```

La rutina **center** se usa para mostrar una cadena en la mitad de la pantalla sobre una determinada línea.

```
/* Centra una linea en la pantalla */
center(s, row)
char s[80];
int row;
{
        eraseline(row);
        posstr(s, (MAXCRTCOL - strlen(s) + 1)/2, row);
}
```

Se diseño **get_key** para que suministre la función primitiva más simple: devolver un caracter correspondiente a cualquier tecla pulsada por el usuario:

get_key llma primero a **keyy** para identificar de que caracter se trata:

```
/*Lee un codigo de 16 bits */
keyy()
{
      union REGS r;

      r.h.ah = 0;
      return int86(0x16, &r, &r);
}
```

Luego **get_key** se encarga de terminar el trabajo avisando de que tipo de caracter se trata:

```
get_key()
{
      union scan {
            int c;
            char ch[2];
            } sc;
      sc.c = keyy();
      tec_esp = 0;
      if (sc.ch[0] == 0) {
            tec-esp = 1;
            return sc.ch[1];
      }
      else
            return sc.ch[0];
}
```

RUTINAS DE MANIPULACIÓN DE CADENAS

Primero rutina **posstr** coloca una cadena en una posición dada de la pantalla:

```
/*Pone una cadena en la posicion col, ren */
posstr(s, ren, col)
int ren, col;
char s[];
{
        gotoxy(ren, col);
        printf("%s", s);
}
```

Luego la función **addchar** añade un caracter al final de una cadena:

```
/* Suma el caracter ch a la cadena s */
addchar(s, ch, max)
char s[], ch;
int max;
{
        char e[1];

        e[1] = '0\';
        if(strlen(s) < max ){
                e[0] = ch;
                strcat(s, e);
        }
}
```

Por último la función **eraseline** borra la línea especificada de la pantalla:

```
/* Borra la linea r */
eraseline(r)
int r;
{
        gotoxy(0, r);
        crt(ERASEOL);
}
```

CADENA DE ENTRADA

La siguiente función acepta una secuencia de caracteres o cadena escrita por el usuario:

```
/* Obtiene una cadena del usuario */
getstring(sp, maxl, col, rov, def)
char sp[], def;
int maxl, col, rov;
{
        char ch, e[1];
        int i;

        strcpy(sp, "");
        e[0] = def;
        e[1] = '\0';
        gotoxy(col, rov);
        if (!tec_esp) {
                if (strlen(e) > 0)
                        poestr(e, col, rov);
                strcpy(sp, e);
                gotoxy(col + 1, rov);
        }
        ch = get_key();
        while (ch != 13) {
                if ((ch == 8) && strlen(sp) > 0) {  /* 8 es <- */
                        cri(LEFT);
                        printf(" ");
                        cri(LEFT);
                        if (strlen(sp) > 0)   /* Suprime un caracter de
                                sp[strlen(sp) - 1] = '';              sp */
                }
                else if (ch == 27 && strlen(e) > 0) { /* 27 es ESC*
                        gotoxy(col, rov);
                        for (i = 1; i <= strlen(sp); i++)
                                printf(" ");
                        gotoxy(col, rov);
                        strcpy(sp, "");
                }
                else  if  (((ch  >=  'A'  &&  ch  <=  'Z')  ||  (ch  >='a'     &&
                        ch <= 'z')) || ch == '#' || ch == '%' ||
                        ch == ' ' || ch == '$' || ch =='-' &&
                        strlen(sp) < maxl && !tec_esp) {
                        if (strlen(sp) < MAXSTR) {     /* a&ade un carac
                                e[0] = ch;                ter a sp */
                                strcat(sp, e);
                        }
```

```
                printf("%c", ch);
            }
            else
                crt(BEEP);              /* Tipo ilegal */
            ch = get_key();
        }
        if (strlen(sp) == 0) {
            sp = s;
            gotoxy(col + strlen(sp), row);
        }
    }
```

La función **copy** copia n caracteres en la cadena s a partir de la i-ésima posición:

```
/* Copia n caracteres a partir de la posicion i */
copy(s, i, n)
char s[];
int i, n;
{
    int x, c;
    char y[90];

    c = 0;
    for (x = i - 1; x <= ((i - 1)+ n)- 1; x++) {
        y[c] = s[x];
        c++;
    }
    y[c] = '\0';
    s = y;
}
```

ACCESO A CARACTERES DE UNA CADENA

La función **gnbchar** busca un carcater no-blanco en una cadena a partir de una determinada posición:

```
/* Obtiene el siguiente caracter no blanco de la cadena */
char s[], *i;
{
    while(gchar(s, &*i) == ' ')
        *i += 1;
    return s[*i];
}
```

La busqueda comienza en el caracter **i**-ésimo de la cadena; si el caracter **i**-ésimo es un blanco, se pasa al siguiente caracter. La búsqueda termina cuando se llega a un caracter no blanco o al final de la cadena.

Los caracteres individuales se extraen de la cadena con la rutina **gchar**:

```
/* Obtiene un caracter de la cadena */
gchar(s, i)
char s[], *i;
{
      if(*i < 0 || *i >= strlen(s) )
            return 0;
      else
            return s[*i];
}
```

gchar devuelve el caracter **i**-ésimo de la cadena **s** como resultado de la función y en el parámetro por referencia **c** .Si **i** está fuera de los límites actuales de la cadena, rutina devuelve un caracter **NULL**.

BIBLIOGRAFIA

PASCAL AVANZADO.
TECNICAS DE PROGRAMACION
Paul a. Sand
McGRAW-HILL

LOTUS 123
GUIA DEL USUARIO
Robert Flast
Lauren Flast
OSBORNE McGRAW-HILL

APLICACIONES DEL 123
PROGRAMAS PRACTICOS
Robert Flast
Lauren Flast
OSBORNE McGRAW-HILL

USING TURBO C
Herbert Schildt
BORLAND-OSBORNE/McGRAW-HILL

ADVANCED TURBO C
Herbert Schildt
BORLAND-OSBORNE/McGRAW-HILL